CXC-DM-010

CXC Data Model

Chandra Data Model Manual, Part 1:

User Introduction to Filtering and Binning in CIAO

CIAO 2.2 Edition
Chandra X-ray Center
October 22, 2001

Contents

Part I

Introduction to the Data Model

Chapter 1

What is the Data Model?

1.1 General Description

This manual describes the filtering and binning language used throughout the CIAO data analysis package, and some of the basic tools used to do simple analysis tasks. The name “Data Model” reflects the intent that there be a high level interface which can be used on data files of different formats: we have a single abstract description (‘model’) that encompasses all the data files we use. Although the event list, source lists and exposure maps produced as output products from X-ray observations are quite different in their information content, they actually have a lot in common in their underlying structure.

An important aspect of the Data Model is that any program that asks for a data file name as input will accept a ‘virtual file’ string which will cause the program to see a filtered version of the file in question. The underlying file can be any of the formats supported by our ‘data model’ subroutine library, currently including FITS, and IRAF QPOE and IMH files.

A common use of the ‘virtual file’ syntax is to create on disk a filtered version of the input file. The **dmcopy** program is used to do this.

1.2 Organization of this Guide

This guide is divided into three parts: (I) an Introduction, (II) a guide to the DM tools, and (III) the Reference Manual. Part I contains information to assist you with getting started, a summary of the available commands, and a simple walkthrough chapter. Part II includes a set of detailed examples. Part III contains more detailed reference material that will be useful once you have become familiar with the basics of using CIAO. You should also refer to the online threads at asc.harvard.edu/ciao/documents_threads.html.

Chapter 2

Getting Started

2.1 Brief Overview

The core utility tools documented in this manual are:

dmlist: to examine the contents of data files

dmcopy: to filter or bin images or tables

dmextract: to create PHA tables

In addition, there are a number of more specialized tools which are documented in the online 'ahelp' and threads.

dmhedit allows you to edit the header keys in a file.

dmtcalc allows you to generate new table columns as arithmetic combinations of old ones.

dmjoin interpolates one table on the grid provided by another.

dmmerge merges two event lists which have the same pointing direction and roll angle (use **reproject_events** to align two event lists with different pointings).

dmgti: to make a time filter from a time ordered housekeeping file by specifying constraints.

dmregrid regrids an image, scaling and rotating it.

dmsort sorts a table.

dmstat gives basic statistics on a table.

dmimgcalc allows you to add, subtract, multiply, divide and compare two images.

dmcoords converts between Chandra coordinate systems.

dmmakepar makes a parameter file from a table header; this is occasionally needed to regenerate an 'obs.par' file from the event list, for pipeline tools which need such an input.

dmappend is a simple tool which adds a single block (table or image) from one file onto the end of another.

2.2 System Setup

Users are encouraged to read about the setup of the CIAO software and parameter files as described in the Beginner's Guide.

2.3 Data Preparation

One purpose of the data model is to handle many file formats avoiding the need for any special file preparation. The current release handles FITS, IRAF QPOE and IMH files.

2.4 Running Data Model Tools

2.4.1 The Command-Line Version

After you have initialized CIAO, the data model tools are available from the unix prompt on the command line. Some sample data files may be downloaded from http://hea-www.harvard.edu/ jcm/asc/dm/examples.tar

A trivial example: The file `evt.fits` is an event list in FITS format. FITS files consist of a number of sections, and in this particular case one of the sections is called EVENTS, containing a list of photons. The command

```
unix:  dmcopy "evt.fits[EVENTS]" copy.fits
```

generates a file `copy.fits` containing the event list data and its associated header.

The DM also considers certain other parts of the file to 'belong' to the events section: the 'good time intervals' are stored in a separate section but are automatically copied along with the events data, because in the abstract model that information is part of the 'block' of data that is being copied. In contrast, it is possible to append unrelated data sections to evt.fits, but these will not in general be copied (dmcopy has a parameter 'option=all' which will force all the sections to be copied).

As usual in Unix, the user must have write permission in the current directory, and as for most CXC tools, if the output file exists before running **dmcopy**, you must set the 'clobber' parameter to 'yes' to overwrite it.

2.4.2 GUI capabilities

Some of the capabilities of the Data Model tools are available in GUIs called *FirstLook* and *FilterWindow*. Please refer to the Beginner's Guide for information on those tools.

2.4.3 Getting Help

The help files for a given task can be displayed using the ahelp command, for example:

```
unix:  ahelp dmcopy
```

The amount of information and format can be specified using options with ahelp:

-l for a more detailed description. This is the default.

-s for a short description

-w to display the full help file in HTML format via a browser.

The detailed syntax can be found with 'ahelp dm' which points the user to a set of help files such as 'ahelp dmbinning'; these files cover a lot of the same ground as this manual.

Chapter 3

The Data Model Walkthrough

3.1 Manipulating Data

The CXC analysis and processing software is built on a common interface library called the CXC Data Model (CXCDM or just DM), which provides users with a powerful, built-in data selection and binning capability. The library is optimized for use with X-ray astronomy data analysis, but is quite general and may be used for other purposes which involve manipulating tabular and image data with lots of associated meta data.

The user doesn't need to know the details of our abstract software model, but will need to pick up some of the basic ideas to get full use out of the software. In particular, because our software supports a number of file formats, not just FITS, we can't use specifically FITS terminology (HDUs, binary table extensions, and so on), since that won't be relevant when other formats are in use.

The CXCDM will work with PROS style QPOE files as well as FITS files, and its programs will work both as IRAF tasks and on the command line like FTOOLS. Our hope is that the learning curve will be short for both PROS and FTOOLS users. CXC analysis tools use the familiar parameter interface used by both PROS and FTOOLS; that parameter interface is separate from the CXCDM, and will be covered in another document.

Unfortunately, there are some remaining compatibility problems with PROS, which was written to deal with ROSAT data and cannot handle all aspects of Chandra data; however, the timing tools in particular have been successfully used with Chandra files.

This chapter introduces you to the Data Model on two different levels. First specific examples are provided of the usage of the tools. Following that is a discussion of its design, and an introduction to its features, including comparison with FITS and QPOE terminology.

3.2 Syntax: Quick Introduction

3.2.1 Basic DM syntax

The CXCDM comes with a set of basic file manipulation tools, of which the most important is `dmcopy`. This tool, in the spirit of Unix's `cat`, performs a variety of important tasks: it copies a virtual file to a real file, optionally changing the kernel (file format). It thus provides filtering and binning capability, and also format conversion.

Data Model tools use as input a filename followed by a series of optional qualifiers in square brackets []:

```
filename[block][filter][columns/binning][opt]
```

–block identifies which section of the datafile is to be used

–filter specifies the filter to be applied

–columns specifies which columns of a table should be used (only used when output is a table)

–binning makes an image from a table (only used when output is an image).

–opt allows the user to modify the default behaviour of the tool.

Any qualifier can be omitted, but those which are used should be in the order given above.

Datafile names are case sensitive; other names are case insensitive for matching (but case sensitive for output).

Full detail on syntax can be found in Section 3.5.1, Section 3.6, and Chapter **??**.

3.2.2 Multiple input and output files

In addition to the generic DM syntax, DM tools can operate on 'stacks' of data. This essentially runs the tool in a loop operating on several files.

There are three forms of stacks:

1) Ascii file stacks

```
dmcopy @instk @outstk
```

where instk and outstk are ASCII files, containing one line per entry, each line should be a valid DM block definition (a filename, or a filtered filename). This will perform dmcopy multiple times, one on each input/output pair.

2) Grid stacks

It is possible to treat multiple 'regions' as separate input files.

```
dmextract "evt.fits[sky=pgrid(4096.0,4096.0,0.0:100.0:10.0,0.0:360.0:360.0)]" @outstk
option=pha1
```

Extracts a set of spectra in ten annuli increasing by 10 pixels radius in each step. There are two grid operators, pgrid (pie grid) and rgrid (rectangle grid). Their syntax is:

pgrid(xcen,ycen,rmin:rmax:dr,thetamin:thetamax:dtheta)

rgrid(xmin:xmax:dx,ymin:ymax:dy)

More details are given in section **??**.

3) Wild card stacks

```
dmextract "acis*evt1.fits" out.fits option=pha2
```

takes several event files matching the unix filter (but not expanded by unix; you need the quotes to prevent this) and extracts a spectrum from each.

3.3 Introduction to the tools

The following sections allow the users to familiarize themselves with some of the capabilities of the Data Model by running commands in various ways. They begin in a sequence which is likely to be part of a data analysis session: examining the contents of files using **dmlist**, filtering a file using **dmcopy**, and extracting a spectrum using **dmextract**. However, for an example of an analysis session, the user is referred to Beginner's Guide and Science Threads.

To run a program, you can either type the name, say 'dmcopy', and hit return, to be prompted for parameters, or give the parameters on the command line.

3.3.1 dmlist

Often in preparation for filtering a file, it may be necessary to examine the contents of the file such as the names of the columns. This can be done using the task **dmlist**.

1. **Listing the blocks in a FITS file.**

dmlist has an option allowing you to control which information is printed; the highest level view, showing the structure of the file, is done by:

```
dmlist file.dat blocks
```

or

```
dmlist file.dat opt=blocks
```

In our test file, this gives the result

```
--------------------------------------------------------------------------------
Dataset: file.dat
--------------------------------------------------------------------------------

     Block Name                          Type         Dimensions
--------------------------------------------------------------------------------
Block    1: HDU0                           Null
Block    2: EVENTS                         Table          20 cols x 6933     rows
Block    3: GTI                            Table           2 cols x 3        rows
```

This tells us the file has 3 blocks, consisting of a null block followed by 2 tables, the first of which has 20 columns and 6933 rows.

In general, the CXC tools operate on a single 'block' (table or image) in the file. If you want to look at the header for a block called EVENTS quote the block name in square brackets following the filename:

```
dmlist "file.dat[events]" header
```

Note that in a typical Unix shell (csh, ksh) you have to enclose the filename string in quotes, because Unix wants to do something special with the square brackets. Like all the CXC software, dmlist uses the parameter file interface, which allows you to be prompted for the parameters. If you use the interactive parameter prompting mechanism:

```
dmlist  <hit return key>
infile (): file.dat[events]
opt (): header
```

here, you don't have to use the quotation marks, since the parameter file interface doesn't do anything special with the square brackets.

2. **Inspecting the columns of the EVENT block.**

Now we probably want to look at the EVENTS table to see what quantities we can filter on. The 'cols' option in dmlist does this job:

```
unix: dmlist evt.fits cols outfile=tmp1
```

This sends the output to the file **tmp1**; it lists the columns in the table and gives their unit, type, and valid range. After the columns, any coordinate systems on the columns are shown. For instance, DM column 8 is "sky(x,y)", a "vector column" which consists of two FITS file columns x and y. The unit of "sky" is pixel; it is a 4 byte real and runs from 0.5 to 8192.5. Note that in Chandra software, the center of a pixel always has pixel coordinates with no fractional part, so the center of the lower left pixel has pixel coordinates (1.0, 1.0) but the bottom left corner of that pixel, which is the minimum value a coordinate can have, is always (0.5,0.5). The notation at the bottome of the listing shows that column 8 has a coordinate system attached: the system EQPOS(RA,DEC) which is symbolically defined in terms of sky(x,y) by an equation. Note that the 'TAN' in the equation refers to the tangent plane projection operator and not the simple trigonometric function. From this output you can easily read off the the fact that the tangent point is at (4096.5,4096.5) and has RA,DEC coordinates (23.4621,30.6603).

```
--------------------------------------------------------------------------------
Columns for Table Block EVENTS
--------------------------------------------------------------------------------

ColNo  Name                 Unit        Type             Range
   1   time                 s            Real8          83981689.6948460042: 84029906.7216549963 S/C TT corresponding to mid-exposure
   2   ccd_id                            Int2           0:9                  CCD reporting event
   3   node_id                           Int2           0:3                  CCD serial readout amplifier node
   4   expno                             Int4           0:2147483647         Exposure number of CCD frame containing event
   5   chip(chipx,chipy)    pixel        Int2           1:1024               Chip coords
   6   tdet(tdetx,tdety)    pixel        Int2           1:8192               ACIS tiled detector coordinates
   7   det(detx,dety)       pixel        Real4          0.50:     8192.50    ACIS detector coordinates
   8   sky(x,y)             pixel        Real4          0.50:     8192.50    sky coordinates
   9   phas[3,3]            adu          Int2(3x3)      -4096:4095           array of pixel pulse heights
  10   pha                  adu          Int4           0:36855              total pulse height of event
  11   energy               eV           Real4          0:  1000000.0        nominal energy of event (eV)
  12   pi                   chan         Int4           1:1024               pulse invariant energy of event
  13   fltgrade                          Int2           0:255                event grade, flight system
  14   grade                             Int2           0:7                  binned event grade
  15   status[4]                         Bit(4)                              event status bits

ColNo    Name
5:    CPC(CPCX) = (+0)[mm] +(+0.0240)* (chip(chipx)-(+0.50))
         (CPCY)   (+0)      (+0.0240)  (    (chipy) (+0.50))
7:    MSC(THETA) = (+0)[deg] +TAN-P[(+0.000136667)* (det(detx)-(+4096.50))]
         (PHI  )   (+0)             (+0.000136667)  (   (dety) (+4096.50))
8:    EQPOS(RA ) = (+23.4621)[deg] +TAN[(-0.000136667)* (sky(x)-(+4096.50))]
           (DEC)   (+30.6603)           (+0.000136667)  (   (y) (+4096.50))
```

3. **Inspecting the event information and format of the EVENT block.**

Next we use the 'data' option in dmlist to look at the actual table data. The 'rows' parameter to dmlist can be used to list only the first few rows.

```
unix: dmlist "evt.fits[events]" data rows=1:4 outfile=tmp2
unix: cat tmp2
```

```
--------------------------------------------------------------------------------
Data for Table Block EVENTS
--------------------------------------------------------------------------------

ROW    time                 ccd_id node_id expno      chip(chipx,chipy) tdet(tdetx,tdety) det(detx,dety)          ...

     1  83982738.2467849255    7    1          3    (287,518)  (4204,2220) (     4166.1630859375,     4099.5781250) ...
     2  83982738.2467849255    7    1          3    (318,558)  (4235,2260) (     4197.0996093750,     4059.6164550781) ...
     3  83982741.4876454622    7    0          4     (49,347)  (3966,2049) (     3928.5751953125,     4270.468750)  ...
     4  83982741.4876454622    7    1          4    (357,483)  (4274,2185) (     4236.082031250,     4134.4916992188)  ...
```

The output has been truncated since the rows are very long.

4. **Displaying specific columns.** It is possible to specify which columns are to be output using a virtual file filter, which we'll talk a lot more about later. The virtual file command below selects the columns time, and the x and y coordinates, again for an abbreviated list of rows. The table below shows the results.

```
unix: dmlist "evt.fits[events][cols time,sky]" data rows=1:4


--------------------------------------------------------------------------------
Data for Table Block EVENTS
--------------------------------------------------------------------------------

ROW    time                 sky(x,y)

     1  83982738.2467849255 (     4092.8640136719,     4100.8286132812)
     2  83982738.2467849255 (     4042.3723144531,     4098.6845703125)
     3  83982741.4876454622 (     4369.5009765625,     4196.4272460938)
     4  83982741.4876454622 (     4080.7888183594,     4024.1035156250)
```

5. **Filtering.** It is also possible to filter the file in making the list. The following command uses the same file and column selection but further selects only rows with a range of energy channels (pha=200:300) and a small part of the image (sky coordinates x between 3900 and 4500, and y between 3650 and 4250). Note that I was lazy and omitted to say that I wanted the '[events]' block - the DM is smart enough to guess that [events] is the most interesting block in the file and uses it if I don't specify one.

```
unix: dmlist "evt.fits[pha=200:300,x=3900:4500,y=3650:4250][cols time,sky]" data rows=1:4


--------------------------------------------------------------------------------
Data for Table Block EVENTS
--------------------------------------------------------------------------------

ROW    time                 sky(x,y)

     1  83982738.2467849255 (     4092.8640136719,     4100.8286132812)
```

```
     2  83982738.2467849255 (     4042.3723144531,     4098.6845703125)
     3  83982754.3986033350 (     4097.0952148438,     4097.5439453125)
     4  83982757.6926859319 (     4106.3100585938,     4098.53906250)
```

6. **Inspecting the header.**

It's often helpful to look at the header information in a file. We distinguish two kinds of FITS header keyword: structural and true metadata. Structural information that describes how the file is laid out is intrinsic to the choice of the FITS format and not to the scientific data; an example is the BITPIX keyword describing the data type or the TTYPE13 keyword describing the name of the 13th table column. We present this information in a file-format-independent way with the 'cols' option of dmlist. In contrast, true metadata include information like the name of the instrument used, or the nominal roll angle of the observation. (Technical note: the exact line between the two kinds of header info is, of course, a function of the abstract data model, and one could imagine a more specialized astronomical model in which the instrument name is an intrinsic piece of information).

By default, the 'keys' option to dmlist gives you only the true metadata. To see the COMMENT and HISTORY information, use 'header' instead of 'keys'; to see the structural FITS keys as well use 'header,raw'.

We truncate the output from these examples to show the first few entries.

```
unix: dmlist "evt.fits[EVENTS]" keys


--------------------------------------------------------------------------------
Header keys for block EVENTS
--------------------------------------------------------------------------------

0001 CONTENT              EVT1                          String
0002 HDUCLASS             OGIP                          String
0003 HDUCLAS1             EVENTS                        String
0004 HDUCLAS2             ALL                           String
0005 ORIGIN               ASC                           String       Source of FITS file
0006 CREATOR              acis_process_events - Version CIAO 2.0b String       tool that created this out
0007 REVISION             1                             Int4
0008 ASCDSVER             CIAO 2.0alpha Thursday, October 26, 2000 String       ASCDS version number
0009 CHECKSUM             JgZBLZZBJfZBJZZB              String       HDU checksum updated 2001-10-15T15:11:13
0010 DATASUM              3225918943                    String       data unit checksum updated 2001-10-15T15:11:13
0011 DATE                 2000-10-30T14:12:54           String       Date and t
...

unix: dmlist "evt.fits[EVENTS]" header


--------------------------------------------------------------------------------
Header keys for block EVENTS
--------------------------------------------------------------------------------

 --  COMMENT                This FITS file may contain long string keyword values that are
```

```
 --  COMMENT                continued over multiple keywords.  The HEASARC convention uses the &
 --  COMMENT                character at the end of each substring which is then continued
 --  COMMENT                on the next keyword which has the name CONTINUE.
0001 CONTENT              EVT1                           String
0002 HDUCLASS             OGIP                           String
0003 HDUCLAS1             EVENTS                         String
0004 HDUCLAS2             ALL                            String
0005 ORIGIN               ASC                            String       Source of FITS file
0006 CREATOR              acis_process_events - Version CIAO 2.0b String       tool that created this out
0007 REVISION             1                              Int4
0008 ASCDSVER             CIAO 2.0alpha Thursday, October 26, 2000 String       ASCDS version number
0009 CHECKSUM             JgZBLZZBJfZBJZZB               String       HDU checksum updated 2001-10-15T15:11:13
0010 DATASUM              3225918943                     String       data unit checksum updated 2001-10-15T15:11:13
0011 DATE                 2000-10-30T14:12:54            String       Date and time of file creation
...

unix: dmlist "evt.fits[EVENTS]" header,raw


--------------------------------------------------------------------------------
Raw Header keys for block EVENTS
--------------------------------------------------------------------------------

Key    1: C            *XTENSION     = BINTABLE            / binary table extension
Key    2: I            *BITPIX       = 8                   / 8-bit bytes
Key    3: I            *NAXIS        = 2                   / 2-dimensional binary table
Key    4: I            *NAXIS1       = 78                  / width of table in bytes
Key    5: I            *NAXIS2       = 30041               / number of rows in table
Key    6: I            *PCOUNT       = 0                   / size of special data area
Key    7: I            *GCOUNT       = 1                   / one data group (required keyword)
Key    8: I            *TFIELDS      = 19                  / number of fields in each row
Key    9: C            *EXTNAME      = EVENTS              / name of this binary table extension
Key   10: C            *HDUNAME      = EVENTS              / ASCDM block name
Key   11: C            *TTYPE1       = time                / S/C TT corresponding to mid-exposure
Key   12: C            *TFORM1       = 1D                  / format of field
Key   13: C            *TUNIT1       = s                   /
Key   14: C            *TTYPE2       = ccd_id              / CCD reporting event
Key   15: C            *TFORM2       = 1I                  / format of field
Key   16: I            *TLMIN2       = 0                   /
Key   17: I            *TLMAX2       = 9                   /
Key   18: C            *TTYPE3       = node_id             / CCD serial readout amplifier node
Key   19: C            *TFORM3       = 1I                  / format of field
Key   20: I            *TLMIN3       = 0                   /
Key   21: I            *TLMAX3       = 3                   /
...
Key  103: C             CONTENT      = EVT1                /
Key  104: C             HDUCLASS     = OGIP                /
Key  105: C             HDUCLAS1     = EVENTS              /
Key  106: C             HDUCLAS2     = ALL                 /
Key  107: C             ORIGIN       = ASC                 / Source of FITS file
```

```
Key  108: C             CREATOR      = acis_process_events - Version CIAO 2.0b / tool that created this out
Key  109: I             REVISION     = 1                    /
Key  110: C             ASCDSVER     = CIAO 2.0alpha Thursday, October 26, 2000 / ASCDS version number
Key  111: C             CHECKSUM     = JgZBLZZBJfZBJZZB     / HDU checksum updated 2001-10-15T15:11:13
Key  112: C             DATASUM      = 3225918943           / data unit checksum updated 2001-10-15T15:11:13
Key  113: C             DATE         = 2000-10-30T14:12:54  / Date and time of file creation
...
```

In this case, structural keys are marked with an asterisk.

Full discussion of syntax for **dmlist** is given in the cookbook (Section **??**).

3.3.2 dmcopy

Several examples of the capabilities of **dmcopy** are provided below. NOTE THAT THE `bin` DIRECTIVE (IN DMCOPY AND OTHER CIAO TOOLS) CREATES AN IMAGE.

1. **Making an image from a table.** The example is an event table (in the [EVENTS] block) created in a MARX simulation called marx.fits. This bins in both coordinates tdetx and tdety from 0.5 to 8192.5 in steps of 8 (so that the pixel size is increased by a factor of 8). More details about the filtering and binning selection possibilities are given in Section 3.5. This creates the output file `tx0.fits`.

```
unix: dmcopy "marx.fits[EVENTS][bin tdetx=0.5:8192.5:8,y=0.5:8192.5:8]" tx0.fits
```

The `bin` command in dmcopy creates an image format file. In this example, the input file has a size of 1.0 MB and the output file, even at this reduced resolution, is 2 MB; in X-ray data, most pixels are zero so the image format is not an efficient way to store things. To view the output you can use any FITS image viewer, for example:

```
unix: ds9 tx0.fits &
```

The header of the image file retains information on the original, unbinned coordinate system. Several other examples of **dmcopy** using the same MARX simulation are given below.

2. **Extracting the center of a FITS file.** The following command extracts the center of the image marxn.fits at half resolution. Again, the [EVENTS] block from the MARX simulation is used. In this case, the user is prompted for the parameters.

```
unix: dmcopy
Input dataset/block specification (marx.fits[EVENTS]):
         marx.fits[EVENTS][bin tdetx=4000:4200:2,tdety=4050:4250:2]
Output dataset name (tx0.fits): tx1.fits
```

3. **Filtering a FITS file both spatially and spectrally.** The following command will extract the central part of the image and a restricted set of energies. The event file is turned into a 3-dimensional image with pixels blocked by a factor of 10 in `tdetx` and `tdety` and a factor of 100 in `pha`.

```
unix: dmcopy "marx.fits[EVENTS][bin tdetx=3900:4500:10,tdety=3650:4250:10,
          pha=200:500:100]" tx6.fits
```

The output file is tx6.fits, a 40 KB file.

4. **Filtering on space while binning on energy and time.** The following command creates a file (tx3.fits) which can be used to examine the spectrum as a function of time. The spectrum is created from the region in detector coordinates tdetx between 4100 and 4300 and tdety between 3850 and 4050 with the default pixel size. The time interval 47144000 to 47148000 s is binned into steps of 100 s, and the approximate energy is binned into 100 eV bins. Displaying the resulting image reveals times of high background which are easily picked out in the energy-resolved light curve.

```
unix: dmcopy "evt.fits[x=3900:4200,y=3900:4200][bin time=83981689:84029906:100,energy=300:10000:100]" te.fits
```

3.3.3 dmextract

dmextract is a program which does a similar binning operation to that available in **dmcopy**, except that the resulting histogram is stored as a table rather than an image.

The default mode of dmextract ('opt=pha1') makes a HEASARC-compatible PHA spectral file, binning on pulse height. Its generic mode ('opt=generic') allows you to bin on any one-dimensional quantity (e.g. time, to make a light curve) or two-dimensional set of regions (e.g. a set of annuli, to make a radial profile). The output is a histogram of the data, together with histograms of counting errors and counting rates.

As an example, the following command will start with the event list `evt.fits` and create the file `source.pha` from a limited spatial region (sky coordinates within 20 pixels of a specified pixel position) using pi (pulse invariant bin) energy channels from 1 to 1024, binned in steps of 2 channels.

```
unix: dmextract "evt.fits[sky=circle(4096,4096,20)][bin pi=1:1024:2]" source.pha
```

The structure of the file can be seen with dmlist:

```
unix: dmlist source.pha cols
```

```
--------------------------------------------------------------------------------
Columns for Table Block SPECTRUM
--------------------------------------------------------------------------------

ColNo  Name                 Unit        Type             Range
   1   CHANNEL              channel      Int4           1:512                   PI
   2   PI                   chan         Real8          1.0:     1024.0         pulse invariant energy of event
   3   COUNTS               count        Int4           -                       Counts
   4   STAT_ERR             count        Real8          -Inf:+Inf               Statistical error
   5   COUNT_RATE           count/s      Real8          -Inf:+Inf               Rate
```

3.3.4 Other simple DM tools

dmappend

dmappend is a simple tool which sticks a single block (table or image) from one file onto the end of another.

```
dmappend "pspc.fits[stdevt]" marx.fits
```

This copies the block STDEVT from pspc.fits to the end of the preexisting file marx.fits.

dmmerge

dmmerge merges two compatible tables. The intent is to allow the user to merge two event lists which are segments of a single observation. Note that we don't yet have a tool to (validly) merge event lists which have different nominal pointing directions. The reproject_events tool (see 'ahelp reproject_events) can do this.

```
dmmerge  @merge.lis columnList="x,y,time" outfile=merge.out
```

where merge.lis is an ascii file with a list of input blocks or filenames (one per line). Each block will be opened with the given column list (it just creates a normal data model open with "filename[cols x,y,time]"); any file which doesn't have one of the requested columns will be skipped. The output file will have a single block, a table whose rows are the rows of the input files appended to each other. No sorting of the data is performed, so all the rows of the second file will come after all the rows of the first file.

The tricky part is merging the header keywords. dmmerge uses a lookup configuration file to select special behaviour for particular header keywords. The configuration file provided with the present release is used in Chandra pipeline processing to force a specific header style. In later releases we will also provide a simpler lookup file which won't be Chandra-specific.

The default value of the lookupTab parameter is dmmerge_header_lookup.txt in the $ASCDS_CALIB directory.

The interface and behaviour of this tool is likely to change in future releases, so be careful about using it in scripts etc.

dmhedit

dmhedit is a program to edit file headers. It's similar in spirit to the FMODHEAD ftool, but operates at a higher level of abstraction. Users are warned that the implementation in the current release has some problems, particularly with string keywords and with editing existing keys.

The input dataset/block specification can either be a single file (dmtest.fits[stdevt]) or a stack of input files (@instack).

The ASCII edit list file can contain two kinds of lines: (1) control lines, beginning with # (2) edit lines The control lines specify what to do with each of the edit lines until the next control line. The valid control lines are:

#add, #delete,

These lines indicate that the subsequent edit lines are keywords to be added at the end of the block header, deleted, or added immediately following the specified existing key. Any other control line is treated as a comment. Each edit line has the form (free format):

KEYNAME = full_value

and full_value is made up of: value, or value / comment, or value / [unit], or value / [unit] comment.

If the value is a string value, it should be included in single quotes.

An example command using dmhedit would be:

```
unix: dmhedit pspc.fits filelist=edit.lis
```

where edit.lis is an edit list file.

An example of an edit list file is:

```
#add
LIVETIME = 142.3 / [s]  Live time
INSTRUME = 'HRC-S'
#delete
PHAMAX
#add
HDUCLAS3 = 'SPECIAL'
HDUVER   = 1.2 / HDU revision
COMMENT  HDUCLAS3 is a special keyword from Goddard
COMMENT  whose value is spurious here.
HISTORY  HDUCLAS3 added by header edit, while HDUVER
HISTORY  is a made up keyword.
#add
GAUSS.POWER = -1.8  / Power law index
```

dmhedit also has a command line 'single' mode to edit a single keyword:

```
dmhedit pspc.fits filelist=none operation=add key=INSTRUME value='HRC-S'
```

dmregrid

dmregrid regrids a stack of (2-dimensional) images, by applying binning, rotation and offset to each image. The relevant parameters are

infile The input image or image stack.

outfile The final regridded output image.

bin X and Y binning specification, given in the format minx:maxx:binx,miny:maxy:biny. Can be specified as an input stack.

rotangle Rotation angle in degrees, measured counter-clockwise. Can be specified as a stack.

rotxcenter X coordinate in pixels of center of rotation. Can be specified as a stack.

rotycenter Y coordinate in pixels of center of rotation. Can be specified as a stack.

xoffset X offset to be applied to regridded image. Can be specified as a stack.

yoffset Y offset to be applied to regridded image. Can be specified as a stack.

npts Integer between 0 and 999, determines sampling level.

Input stacks are ASCII files with fields delimited by comma or space.

Example of a command using stack input:

```
unix:  dmregrid infile=@inlist outfile=newimage.fits bin=@binlist rotangle=@rotlist
rotxcenter=@xrotlist rotycenter=@yrotlist xoffset=@xofflist yoffset=@yofflist npts=0
clobber=yes verbose=1
```

All of the above parameters which can be specified as input stacks can also be specified as single values, in which case the same value will be applied to all input images. If there is a stack of input images, the final output image will be the sum of the individual regridded images. Physical and world coordinate systems are attached to the output image.

If `npts`=0, then for each pixel overlapping a regridded pixel, a polygon is created that is the intersection of the two pixels. The area of this polygon is calculated and is used to weight the number of counts within the unit pixel to be allocated to the regridded pixel. If `npts` is given a positive value, an approximate regridding algorithm is used in which `npts`×`npts` uniformly spaced points within a regridded pixel are sampled. The appropriate fraction of the number of counts within the unit pixel encompassing each sampling point is then allocated to the regridded pixel. The approximate algorithm is about an order of magnitude faster than the exact algorithm.

By specifying binning and setting all other parameters to zero, it is possible to have dmregrid do dmcopy's task (which dmcopy cannot yet do) of binning an image. Setting `npts` to something other than zero significantly speeds up the processing. Thus, for example, to bin a 512x512 image by 2, one could say

```
unix:  dmregrid infile=fullresimage.fits outfile=binimage.fits bin=1:512:2,1:512:2,
rotangle=0 rotxcenter=0 rotycenter=0 xoffset=0 yoffset=0 npts=3
```

The reason for explicitly setting some parameters to zero rather than just omitting them from the command, is that the tool will look up the values of all omitted parameters in the parameter file and use those values. That is, it will use the value that was used the last time that parameter was explicitly set. See section **??** for some more information about parameter files.

Note

dmregrid in its current version cannot handle an input binning specification stack in which a subsequent item results in a greater number of output pixels than the first item. This bug will be fixed for the next release or patch.

dmimgcalc

dmimgcalc performs basic arithmetic on images. It is a tool still in a preliminary stage. Its current abilities include adding, subtracting, mulitplying, dividing and comparing two images, with optional 'weights' on the input images. Operations on single images are not possible.

The syntax is

```
dmimgcalc image1.fits image2.fits outimage.fits op weight=<constant> weight2=<constant>
```

where op is one of: add, mul, sub, div, tst. If op equals 'tst' then the output image should be specified to be "none". dmimgcalc only tests whether the images are numerically identical. Other tests are not yet possible.

A typical command would be

```
unix:  dmimgcalc acis1.fits exposure.fits normal.fits div
```

which divides the acis image by the exposure map and puts the result into the file normal.fits,

or

```
unix:  dmimgcalc acis1.fits acis2.fits sum.fits add weight=2.5 weight2=1.414
```

which is equivalent to $(2.5 \times \text{acis1.fits}) + (1.414 \times \text{acis2.fits})$.

dmstat

dmstat computes standard statistics for a table or an image. For tables it outputs the minimum value, the maximum value, the standard deviation, the number of good values and the sum of the entire column. For an image it outputs the minimum and maximum values, the standard deviation, the sum, and the centroid. The syntax is

```
dmstat filename
```

where filename is a Data Model virtual file specification. That is, you can filter the input to the tool. Sample commands and the corresponding output are shown below.

To get information on the `time` and `pha` cols:

```
unix:  dmstat "file.fits[cols time,pha]"


time[s]
        min:    53161434.770097
```

```
        max:    53174191.333063
        mean:   53167802.580212
        sig:    3692.285028
        total:  2097257140579.023438
        good:   39446.000000
        nulls:  0

pha[adu]
        min:    43.000000
        max:    3747.000000
        mean:   860.133651
        sig:    1046.696570
        total:  33928832.000000
        good:   39446.000000
        nulls:  0
```

To get statistics on an image minus a region:

```
unix: dmstat "image.fits[\!circle(50,50,10)]"


PRIMARY(AXIS1, AXIS2)
        min:    -19.777779
        max:    5.555553
        cntrd:  ( 168.334862 ,146.626860  )
        std:    ( 75.583215 ,73.827393  )
        sum:    10806.778002


unix:  dmstat "file.fits[cols sky]" centroid=yes sigma=yes


sky(x, y)[pixel]
    min:        ( 3900.00513 3900.00854 )
    max:        ( 4411.99805 4411.96045 )
   mean:        ( 4125.71523 4117.71857 )
  sigma:        ( 113.777582 110.039969 )
    sum:        ( 123940611 123700384 )
   good:        ( 30041 30041 )
   null:        ( 0 0 )
```

There is a bug in this version of dmstat that causes incorrect extraction of WCS column names, leading to the “EQPOS” above instead of RA and DEC.

3.4 Design Overview

We have aimed to design a high level interface to the data which doesn’t depend on the details of the exact file format being used. This leads to the idea of a ‘data model’, in other words an abstract model of the data file that gives a common language which can apply to different file formats. This is possible because all of the data we use in X-ray astronomy data analysis is stored in a few common (albeit complicated) structures, so that although data products like event lists, source lists, and exposure maps are all quite different from the astronomer's point of view, they actually have a lot in common in their underlying structure. The data model provides a way of exploiting these similarities and describing the differences. Defining the data model in a way independent of the file format means that we can operate transparently on both FITS and IRAF format data files (and later can add other formats without changing the interface), and we can deal with high level structures like coordinate systems without the user having to know the latest FITS conventions for storing that information.

Although our jargon is a little different, in practice the interface is very similar to SAO's IRAF x-ray package, PROS, with which users of ROSAT data will be familiar. We've upgraded the interface to make it much more general, and the insides are completely redone so that it will work on FITS files as well as PROS style QPOE and IMH files. At the lowest level, our software calls Bill Pence's CFITSIO library for its FITS access, and Doug Tody's IRAF system libraries to read IRAF format files. However, the software does not require an IRAF installation unless you want to run the programs as IRAF tasks.

The goals of the CXCDM library are:

- to provide an easy way to filter and bin data files at runtime, extending the functionality provided in the PROS system. The user can specify a filtered or transformed version of their input file, without the individual application having to explicitly perform filtering.
- to provide a common view of multiple file formats, including FITS, IRAF/QPOE, and ASCII files, and isolate the details of those formats from the calling program. This data access layer for all our software also allows us to isolate all of the format-specific keyword conventions, keeping them up to date without having to rewrite each individual application program.
- to provide a programming interface to the data which is higher level than basic I/O libraries like CFITSIO, but is still generic.
- to support encoding a greater level of self-description in the data files while retaining back compatibility with existing conventions.

To the user, that first item is the main one of interest, although users who like writing their own analysis code will want to take a look at our programmers' guide which addresses the second item. For a more detailed look at the ideas behind the CXCDM, refer to our design overview document.

3.5 Filtering

Filtering files (i.e. selecting subsets of a data file by spatial position, energy and/or time) is one of the most important features of CXCDM.

3.5.1 Syntax

The full syntax of a data model filter is a filename followed by a series of optional qualifiers in square parentheses:

```
filename[block][filter][columns][binning]
```

The **block qualifier** specifies which section of the file to use; in FITS, it specifies an HDU (Header Data Unit). The FTOOLS syntax of [n], where n is the HDU number counting from 1, is accepted, but the name (EXTNAME/HDUNAME value) of the HDU is equally valid and in many cases will be more intuitive. If the block qualifier is omitted, the first 'interesting' block will be used - the first FITS HDU for which NAXIS is nonzero, excluding GTI and WMAP extensions.

The **filter qualifier** specifies which rows of a table or pixels of an image to use, by filtering on the values of columns or axes. A nontrivial filter example is [pha=3:20,time=100:200,250:300,450:500,ccd_id=3]. The filter on each variable is expressed as a series of acceptable ranges. Logical operators (&, <, etc) may also be used, as described in the detailed DM virtual file documentation.

The **columns qualifier** specifies which columns from a table should be used; it cannot be used with images.

The **binning qualifier** makes an image from a table, binning on any collection of columns, optionally specifying the ranges and binning factors:

[bin x=100:612:4,y=512:1024:4,time=8441014.3:8441420.8:20.2]

specifies making a 3-dimensional x,y,time cube in which each x,y pixel is 4 input pixels, and each time pixel is 20.2 seconds. The first two coordinates define the axes of the image; (in this case x,y but could be detx,dety or time,pha or ...)

Note

When specifying a virtual file on the command line, it is necessary to enclose the entire file specification in double quotes, e.g.

```
unix:  dmcopy "file.fits[bin x=100:612:4,y=512:1024:4]" image.fits
```

because the Unix shell will otherwise attempt to parse the command line instead of passing the entire argument to the DM tool. If you choose to start the tool without a file specification and be prompted for parameter values, then you should **not** use the quotes.

```
unix:  dmcopy
Input dataset/block specification ():  file.fits[bin x=100:612:4,y=512:1024:4]
Output dataset name ():  image.fits
```

Cautions

The order of the syntax is very important. For instance, you can use dmcopy to create a filtered image `evt1.out` from the event list file `Sources_evt.fits` with the command below. This will produce an image containing the data from the region x 16370 to 16390 and y 16370 to 16390 imbedded in a larger region from x 16350 to 16410 and from y 16350 to 16410 containing no counts.

```
unix: dmcopy "Sources_evt.fits[x=16370:16390,y=16370:16390][bin x=16350:16410,
          y=16350:16410]" evt1.out
```

Reversing the order of the qualifiers is **not allowed**:

```
unix: dmcopy "Sources_evt.fits[bin x=16350:16410,y=16350:16410]
             [x=16370:16390,y=16370:16390]" evt2.out
```

If the previous command is broken down into two commands like this,

```
unix:  dmcopy "Sources_evt.fits[bin x=16350:16410,y=16350:16410]" tmp1.out

unix:  dmcopy "tmp1.out[25:45,25:45]" tmp2.out
```

the result is different from the first example. First a file `tmp1.out` is created which is an image containing the events from x 16350 to 16410 and from y 16350 to 16410. This image is further filtered so that the output file `tmp2` is only the smaller image containing x from 25 to 45 and y from 25 to 45 (in the coordinate system of tmp1.out); not a small image imbedded in a larger blank image (above: `evt1.out`).

3.5.2 Virtual Files

A powerful capability of the CXCDM is being able to use a filtered version of a file without actually writing it to disk. That is, in any program using the CXCDM interface, you can pass a 'virtual file' where an input file is expected. The virtual file is implicitly created from a true disk file by filtering or binning, but does not get created on disk or even formatted into a Unix pipe. Instead, I/O calls in the program return records of the virtual file by internally accessing the actual disk file and figuring out which data passes the filter.

For instance, for a source detection program CELLDETECT you have an event list from a full resolution 32k square HRC image. In this example we use the event list for the image `Sources_evt.fits`. The output file containing the sources found is called `Sources_evt_src.fits`.

```
unix: celldetect infile="Sources_evt.fits" outfile="Sources_evt_src.fits"
```

However, because of space or run-time limitations, you want to use CELLDETECT to find the sources in only part of the image. Instead of running CELLDETECT on the full image, you can pass it a subset of that file containing only x values in a limited range, without first creating any intermediate disk file. The program sees the 'virtual' file you have specified. In this case, an image is created from the subset of the event file for X between 16350 and 16420 and Y in the same range, binned in steps of 1.

```
unix: celldetect infile="Sources_evt.fits[bin X=16350:16420:1,Y=16350:16420:1]"
            outfile="Sources_evt_src.fits"
```

To examine the output file to see what sources were found, **dmlist** can be used. First, inspect the blocks in the output file, which shows that there are two blocks in the file, with the source list being the second.

```
unix: dmlist "Sources_evt_src.fits" blocks

 --------------------------------------------------------------------------------
```

```
 Dataset: Sources_evt_src.fits
 --------------------------------------------------------------------------------

  Block Name                          Type         Dimensions
 --------------------------------------------------------------------------------
  Block    1: HDU0                          Null
  Block    2: SRCLIST                       Table         26 cols x 1      rows
```

The next step is to look at the result in the source list (block 2). In this case the RA, Dec, source counts and source count error are requested for the sources in the simulated image (in which the one recovered source is very close to the center of the image).

```
unix: dmlist "Sources_evt_src.fits[SRCLIST][cols ra,dec,net_counts,net_counts_err]" data

--------------------------------------------------------------------------------
Block    2: SRCLIST                       Table          4 cols x 1    rows
--------------------------------------------------------------------------------



--------------------------------------------------------------------------------
Data for Table Block SRCLIST
--------------------------------------------------------------------------------

 ROW    RA           DEC          NET_COUNTS   NET_COUNTS_ERR

  1  9.47086E-06 -1.24421E-05        186.0      15.6040
```

As another example, you can create a virtual image from the event list, filtering both spatially and in time. The spatial subspace is the same as above, but the time range is from 37277062.1 to 37277062.2 seconds.

```
unix: celldetect infile="Sources_evt.fits[events][time=37277062.1:37277062.2]
              [bin X=16350:16420:1,Y=16350:16420:1]" outfile="Sources_evt_src.fits"
```

Note that in this case, the order in the syntax is important. This command first goes to the events block, next filters the data in time, and then makes an image by binning over the requested X and Y region. You can't filter after binning in the same operation. In addition, once the binning is done, the data no longer contains the time information.

The CXCDM supports the full spatial 'region' filtering used in PROS:

```
unix: celldetect infile="evt.fits[events][(ra,dec)=circle(14:07:23 -00:14:23 3')][bin x,y]"
       outfile="result.fits"
```

This selects only photons whose celestial positions lie within 3 arcmin of the quoted coordinates.

Cautions

Like many powerful capabities, there are things that the user must be careful of in using virtual files.

1. If you provide this file specification to a tool:

```
"Sources_evt.fits[events][bin X=10001:12048:1,Y=10001:12048:1]"
```

this creates a 2048x2048 image.

However, consider the specification:

```
"Sources_evt.fits[events][X=10001:12048,Y=10001:12048][bin x,y]"
```

This does something slightly different. The file `Sources_evt.fits` contains the event list from a 32kx32k simulated HRC image. The specification selects events only in the limited range of x and y, but doesn't restrict the overall size of the implied image. The bin command then creates a 32kx32k image with zeroes everywhere except in the specified x and y range. This is probably not what the user wanted.

2. The filters are recorded in the file's 'data subspace'. Use '`dmlist filename subspace`' to see how the file has been filtered.

3.6 Syntax summary

The syntax is largely inspired by the IRAF/PROS filtering, regions, image section, and blocking capability. We have added intensity filtering from XSELECT and relational filtering and column selection from ETOOLS. From MIDAS we plan to add filter stacking capabilities but that isn't in the current version. We retain as much as possible of the PROS filtering syntax to keep it familiar to users of that system. We have integrated the regions syntax with it, and added new capabilities in a natural way. Keeping it compact was an important goal.

The syntax consists of a string with qualifiers surrounded by square parentheses, (Section 3.5.1) thus:

a[b][c][d][e]

Each group within square parentheses has a different meaning. The parentheses minimize the risk of an unintended string with a single mistype and make the command more readable,

Dataset names are case sensitive. All other names are case-insensitive for matching, but case-sensitive for output. (In other words, if you ask to read a quantity evENts, the software will successfully find the table EVENTS; but if you ask for an output table to be named evENts, that is exactly what it will be named.)

3.7 Multiple File Format Support

DM-aware programs will read any of the supported formats automatically, figuring out which format is being used by looking at the file. The only restriction is that the IRAF formats such as QPOE and IMH must have filenames which end with the appropriate extensions, .qp and .imh, since the underlying IRAF subroutines require this. By default, output files will use the same format as the input files, but that can be overridden by the use of the 'format' parameter provided in each CXCDM tool.

Although FITS files will be used for all archival and user data distribution purposes, for run time analysis, and for compatibility for users working in the IRAF environment, our software will support the use of QPOE event files and IRAF IMH images. The 'data model' knows which structures in QPOE and FITS files are equivalent. The various supported file formats are called 'kernels'. The ability to use one set of tools to read multiple formats removes the need for conversion tools and greatly improves interoperability. The gain is also a snag: a given high level concept may map to two quite different representations in the data file. For example, Good Time Intervals (GTIs), which represent the times on which the data has been filtered, are stored in binary table extensions in FITS files. In the QPOE files the same information is stored in the QPOE header rather than in a separate table. To allow a program to handle either FITS or QPOE, the GTI's are handled as part of the 'data subspace', discussed below. To the user, the main effect is that a given events table 'knows' about a GTI which 'belongs' to it, and operations copying or filtering the events table will automatically copy and update the GTI.

In the table below, we give some examples of concepts used in the CXCDM and how they map into the FITS and QPOE formats.

Table 3.1: Examples of CXCDM abstractions

CXCDM abstraction	FITS	QPOE/IMH
Dataset	File, may have many HDUs (Header Data Units)	1 or more files
Block	HDU	Single file
Table	BINTABLE	QPOE
Image	IMAGE	IMH
Column descriptor	TTYPE etc	QPOE structure
TIME subspace	GTI HDU	deffilt keyword
Other subspace	keywords	keywords
Coord descriptor	TCRVL keys	Opaque binary MWCS

The third kernel we expect to support is an ASCII kernel, which we anticipate will be most useful for importing user tabular data. The exact format has not been finalized, but will probably involve optional free-format FITS-style header keywords and will also support the 'rdb' Unix database format.

3.8 Data subspace

Another new concept in the CXCDM is the 'data subspace'. We are already used to the idea that it's important to remember how the data have been filtered. GTIs, PHA channel ranges, and SAOIMAGE region files are used to store this information. We think it's helpful to have a systematic approach. The CXCDM data subspace is the list of 'how this data has been selected'. This list is updated as you further filter the file, so that programs which make use of the data subspace can figure out which calibrations to apply.

The simplest data subspace filter is the logical AND of the restrictions on several columns, as in all the filtering examples given above.

More complex logical expressions, like 'this data was taken when (X1=A AND X2=B) OR (X1=C AND X2=D)' may be rare in practice, but we have such a case with Chandra's ACIS CCD imager, where we want to make a single event list for the multiple chips, but due to dropouts and saturation effects the GTIs may be different for each chip. The ability to use OR in such filters is also useful for combining data from different missions.

3.9 Header keys

Users sometimes have to look directly at the header information in data files. In this section we describe some of the special keywords that CXCDM writes to FITS files; other formats are handled in a similar way. A full description of our FITS conventions is given in a separate document.

- Grouped ('vector') columns: we support giving a single name to a related group of columns, for instance an X,Y position pair, a (start,stop) time bin, or a (value,uncertainty) pair. The MTYPEn, MFORMn, and METYPn keywords are used to do this.
- Long keyword names: we fake out the FITS restriction that keyword names can only have 8 characters by using pairs of keywords DTYPEn and DVALn, one to hold the name and the other to hold the value. DFORMn and DUNITn keywords can also be used to specify data type and unit. Of course, since all existing keywords are 8 characters or less, they don't use this convention and compatibility with existing programs is fine - they just won't recognize the new keywords, which they're not expecting anyhow.
- We record the data subspace in a block with the keywords DSTYPn, DSVALn, and DSREFn.