Sample CIAO help files:

- dmlist
- parameter
- filtering

SUBJECT(dmlist) CONTEXT(tools)

SYNOPSIS

List contents or structure of a file.

SYNTAX

```
dmlist infile optlist [outfile] [rows] [cells]
```

DESCRIPTION

'dmlist' dumps the contents or header of a file or block (a block is a subfile or FITS extension) to ASCII in an organized way. It corresponds to the FTOOLS fdump and fstruct programs, but interprets the input file at a higher level. All CXC data model formats are supported.

dmlist uses a comma-delimited list of options to select which information is displayed: blocks, keys, comments; Header, Cols, Subspace, Data, Array, Full, Struct, all, clean, raw.

Selected rows of data may be dumped: rows=min:max

EXAMPLES

(1) dmlist acis.fits full,all | more

List everything dmlist can tell you about the file and display the output using "more".

(2) dmlist acis.fits opt=blocks

See what blocks are in the file "acis.fits". (Since opt is a positional parameter, you can omit the 'opt=' if you like).

(3) dmlist "acis.fits[2]" opt=cols

See what columns are in the second block (note that we always count starting at 1).

(4) dmlist "acis.fits[events]" opt=header,subspace

Look at the header and the data subspace for the block called "events" in the file. You can choose blocks either by name or by number. If you don't specify a block, the program guesses which one you want.

(5) dmlist "acis.fits[events][pha=20:30]" opt=data rows=100:104

Look at rows 100 to 104 of the virtual file filtered to show only pulse heights between 20 and 30. Note the row numbers are those passing the filter, not the original row numbers of the underlying file.

(6) dmlist "pha2.fits[cols channel,counts]" opt=array rows=3:3

Look at the channel and counts arrays for row 3 of the file, displayed in 'array' (vertical) format.

(7) dmlist "acis.fits[events]" opt=header,raw verbose=0

Look at the FITS header instead of the DM-level header (the 'raw' option specifies use of the low level interpretation) and generate a minimal, 'bare' version of the output (verbose = 0).

PARAMETERS

name	type	ftype	def	min	max	units
infile	string	input				
opt	string					
outfile	string	output				
rows	string					
cells	string					
verbose	integer					
mode	string					

DETAILED PARAMETER DESCRIPTIONS

infile

type=string
filetype=input

The input virtual file specification.

opt

type=string

The list of options, separated by commas.

This required parameter can be one or more of the following:

Blocks - Summarize all the blocks (images or tables) in the file, one line per block.

Keys - List the ASCDM header keys for the selected block. Remember that not all the FITS keywords in a FITS file will be included - the ones like EXTNAME and CRPIX that have a special meaning for the structure of the file don't count as ASCDM keys, their values show up in other places (e.g. the output from cols). With this option, the comment and history records are suppressed. To see the low level list of all raw FITS header keywords in a FITS file, use opt=header,raw.

Comments - List the ASCDM comment keys for the block. For a FITS file, these keys are the COMMENT and HISTORY keywords.

Header - Equivalent to Keys,Comments. List both DM level header keys and comment/history records. - List the ASCDM header keys for the selected block. Remember that not all the FITS keywords in a FITS file will be included - the ones like EXTNAME and CRPIX that have a special meaning for the structure of the file don't count as ASCDM keys, their values show up in other places (e.g. the output from cols). To see the low level list of raw FITS header keywords in a FITS file, use opt=header,raw.

Cols - List the ASCDM columns for the selected block. Shows 'vector columns' like (X,Y) as pairs. This output shows you the names of the variables you can filter a table on and what their valid ranges are. If the block is an image, it gives you one column with the name of the image (images are interpreted as a table with a single row and column).

Subspace - Summarizes the data subspace for the block. This describes the filters that have been applied to the data, either by the user or in processing.

Data - Prints the data segment. The cells parameter controls how much of an array column gets printed. By default, the data is printed in an 'ornate' format showing arrays grouped in parentheses etc. If you use "Clean" option as well, the output is in a simpler format suitable for reading by other analysis programs.

Array - Prints the data segment, but with arrays printed vertically. The cells parameter controls how much of an array column gets printed.

Full - Equivalent to 'blocks,header,cols,subspace,data'

Struct - Equivalent to 'header,cols,subspace'

All - Print the info selected by the other options for all the blocks in the dataset.

Clean - When used in conjunction with the Array or Data options, produces stripped down output suitable for parsing by other programs.

Raw - Provides a lower level view of the header. 'header,raw' lists the raw FITS header keywords, not just the data model keywords. The old 'data,raw' functionality is now provided by 'data,clean'.

outfile

type=string
filetype=output

The output ASCII file to be created, if any.

The output ASCII file to be created; output is sent to the screen if this parameter is blank.

rows

type=string

Range of table rows to print (min:max)

rows=30:40 Print row 30 to row 40 rows=40 Print row 1 to 40 rows=40:40 Print only row 40 rows=" " Print all rows rows=-1 Print all rows

cells

type=string

Range of array indices to print in array columns and in images.

Range of array indices to print in array columns and in images. cells = 1:8 Print only array elements 1 to 8 of each array cells = all Print all array elements.

verbose
type=integer

Controls amount of information to print (0-5).

The verbose parameter provides debugging information; verbose = 0 is usually fine.

mode
type=string

See the parameter interface documentation.

mode=h will stop the program prompting for parameters.

SEE ALSO

dmappend(tools), dmarfadd(tools), dmcontour(tools), dmcoords(tools), dmcopy(tools), dmextract(tools), dmgti(tools), dmhedit(tools), dmimg2jpg(tools), dmimgcalc(tools), dmimghist(tools), dmimgthresh(tools), dmkeypar(tools), dmmakepar(tools), dmmakereg(tools), dmmerge(tools), dmpaste(tools), dmreadpar(tools), dmsort(tools), dmstat(tools), dmtcalc(tools), dmtcalc_expressions(tools), dmtype2split(tools), dmwritefef(tools), mtl_build_gti(tools), paccess(tools), parameter(tools), pdump(tools), pget(tools), pline(tools), plist(tools), pquery(tools), pset(tools), punlearn(tools), region(dm), reproject_events(tools), stk_build(tools), stk_count(tools), stk_read_num(tools), subspace(chandra)

VERSION

CIAO 2.1

LAST MODIFIED

15 February 2001

SUBJECT(parameter) CONTEXT(tools)

SYNOPSIS

Describes the parameter interface.

DESCRIPTION

All CXCDS tools use ASCII parameter files to get and store processing parameters. This interface is similar and extends to the IRAF and FTOOLs system and includes all their features.

Parameter files are picked up from a system install location and copied to a user's local directory. Thereafter, whenever the tools are run the version in the user's local directory is used. The environment variables that control where to search for the parameter files are (in search order) PDIRS, PFILES, and UPARM. The path listed before the ";" is the user writeable location where updated parameter files are kept. The ":" separated list of directories that follow shows where to search for the system default files.

The following fields are used for each parameter.

o) name - Name of the parameter used by tool.

o) type - data-type of the parameter. Valid values are s=string, i=integer, r=real, b=boolean, and "pset".

o) mode - controls whether the parameter is prompted for or not. Valid mnemonics are q=query, h=hidden, a=automatic (get value from "mode" parameter which can be modified with l=learn, eg "ql" means to query and learn.

o) value - default parameter value. Can be blank if no sensible default exists. If mode=*l, then after the tool is run the last value becomes the new default.

o) minimum/enumeration - minimum data value. Can be blank if no sensible value exists. Can also be a "|" separated list of values. The value can only take one of the values in the enumeration.

o) maximum - maximum data value

o) prompt - The prompt the user will see if they are prompted for the value

Lines beginning with "#" are comments and are ignored.

When prompted for a parameter, there are several special features that can be invoked

o) "Tab completion". Similar to tcsh's tab completion functionality. If the input string is a path, hitting the TAB key will list all the file+path names that match. If only one matches, it will fill in the parameter string.

o) "History". Using the UP and DOWN arrow keys you can cycle thru the previous data values for the current parameter and the parameters queried for earlier. If the value being entered has

enumerated values, these are also cycled thru.

o) "Help". If the user enters "?" as a parameter value the help file for the tool will be displayed. The user can also enter "?toolname" to get help for a different tool.

EXAMPLES

(1) my_tool

If a tool is run without any command line options then each parameter that the tool needs will be read from the user's copy of the tool's parameter file. The default parameter file name is my_tool.par. If the user does not have a copy of the tool's parameter file, one will be copied to the user's work directory (controlled by the PDIRS environment variable).

(2) my_tool parameter1=value1 parameter2=value2 ...

Each "Key=Value" Pair will set the named parameter to the given value. If the value is invalid (below min, above max, etc) the tool will prompt for a correct value. If the parameter listed is not in the tools parameter file, the tool will exit with an error.

If there are any parameters that have not been specified on the command line that have mode=q, they will be prompted for.

(3) my_tool value1 value2 ...

"Command line params"

The values are read from the command line and applied to the parameter file in the order that parameters appear in the parameter file, which is not NECESSARILY the order the tool prompts for them.

(4) my_tool parameter1=)parameter2

"Self redirection": This tells the tool to use the value stored in parameter2 for the value of parameter1, where parameter1 and parameter2 are in the tool's parameter file.

A common example of this is for the event processing tools where the event definition string is redirected from one of several listed in the .par file.

(5) my_tool parameter=)tool2.parameter

"External redirection": Similar to "Self redirection" this tells the tool to set parameter1 in its parameter file to parameter2 in tool2's parameter file.

This is very useful for scripts when chaining tools together to set the input file name of one tool to the output file name of an earlier tool in the script.

(6) my_tool parameter=))command

"Command substitution": The sh shell command after the "))"'s is evaluated and the result is returned and used as the parameter's value.

```
        The most common use of this feature is to use environment
        variables as part of parameter value, e.g.

        parameter="))echo $ASCDS_INSTALL/data/psfsize.fits"

        (7) my_tool @@parameter

        "Alternate parameter file": The default behavior of the
        parameter interface is to look for my_tool.par in the user's
        search path(s). Using this syntax the user can use an alternate
        parameter file. The alternate parameter file can be in a
        "non-searched" location,eg @@/tmp/my_tool.par or have a
        completely different name, eg @@/home/foo/my_alternate.par.

        This can be used to store parameter files locally, that is have
        some relative path in the PDIRS environment variable, but when
        running the tool from another path to force it to use the
        non-local version.
SEE ALSO

        dmappend(tools), dmcopy(tools), dmhedit(tools),
        dmimgcalc(tools), dmkeypar(tools), dmlist(tools),
        dmmakepar(tools), dmmakereg(tools), dmmerge(tools),
        dmpaste(tools), dmreadpar(tools), dmsort(tools), dmstat(tools),
        dmtcalc(tools), dmtcalc_expressions(tools),
        dmtype2split(tools), paccess(tools), pdump(tools), pget(tools),
        pline(tools), plist(tools), pquery(tools), pset(tools),
        punlearn(tools), stk_build(tools), stk_count(tools),
        stk_read_num(tools), subspace(chandra)


VERSION

        CIAO 2.1


LAST MODIFIED

        9 February 2001
```

SUBJECT(filtering) CONTEXT(dm)

SYNOPSIS

The CIAO filtering syntax

SYNTAX

```
[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=...]
```

DESCRIPTION

This help file is split up into the following sections:

- o) 1. Table filtering on columns with real data types
- o) 2. Table filtering on columns with integer data types
- o) 3. Table filtering on columns with character string data types
- o) 4. Table filtering on columns with bit data type
- o) 5. Table filtering on vector columns using region filters
- o) 6. Compound filters
- o) 7. Exclude filters

1. 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; see also ‘ahelp dmimfiltering’ for filtering files which are in image format. 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.

2. 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,

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

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

3. 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.

4. 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).


        5. 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 'ahelp regions' 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


        6. COMPOUND FILTERS

        The syntax now 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 ||.


        7. EXCLUDE FILTERS

        A very useful new 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
SEE ALSO

        binning(dm), coords(chandra), dm(dm), dmcols(dm), dmimages(dm),
        dmimfiltering(dm), dmintro(dm), dmsyntax(dm)


VERSION

        CIAO 2.1
```

LAST MODIFIED

16 February 2001