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

AHELP for CIAO 3.4

dmtcalc

Context: tools

Synopsis

Modify and create columns in a table

Syntax

```
dmtcalc  infile outfile expression [kernel] [clobber] [verbose]
```

Description

`dmtcalc' is a flexible general purpose table calculation utility. It can change existing columns add new ones using arbitrary expressions which can involve other columns − possibly smoothed − mathematical functions such as sqrt(), random numbers, values taken from the file header, and much more. It uses the same syntax as the dmgti and dmimgcalc tools: this syntax is described in detail in "ahelp syntax".

The input expression can either be given directly on the command−line or stored in an external file and accessed as a stack − i.e. "@filename". The use of a stack allows more than one column to be created or modified at one time.

When dmtcalc starts, it will make an effort to assure that any data dependencies in the expressions (referenced columns or keywords, or new columns defined in previous expressions) are satisfied, that there are not any data type conflicts within the expressions, and that there are no conflicts of array size or dimensionality. If any of these tests fail, an error message will print, and the tool will exit.

Overwriting times in the GTI blocks

dmtcalc correctly changes the start and stop columns in the file (and thus the order can change). However, it then proceeds to copy the data subspace from the input to the output file and so the values are overwritten with the original values. To override tstart/tstop, they need to be put into a new extension:

```
        infile="file.fits[gti][new_gti]"
```

Then use the reverse notation when using as a filter:

```
       infile="file_2.fits[@file.fits[new_gti][gti]]"
```

Example 1

```
dmtcalc evt2.fits evt2_hacked.fits
expression="dtime=(time−TSTART)*1.0e−3"
```

This time we have used dmtcalc to create a new column called "DTIME" in the output file evt2_hacked.fits (new or changed columns always have their names written in upper case in the output file, however they are written in the expression).

The contents of the DTIME column are set equal to

```
(time−TSTART)*1.0e−3
```

where TSTART is replaced with the value of the TSTART keyword from the header. The case of header keywords is unimportant, so

```
(time−tstart)*1.0e−3
```

would give the same result.

For this particular example DTIME can be considered to be the time since the start of the observation in kilo−seconds. In general the expressions used in these examples are just used to illustrate dmtcalc's capabilities and so should not be taken to have any real meaning for Chandra (or any other) data. More details of the allowed expressions are given in "ahelp syntax".

Example 2

```
dmtcalc evt2.fits evt2_hacked.fits expression="energy=((float)pha)*0.1"
```

Here we use dmtcalc to over−write the values in the energy column to be ten percent of the PHA values. Since the PHA values are stored as integers we convert them to floating−point numbers − by use of the term "(float)" − before the division.

If we had omitted the "(float)" term, the output would still have been automatically converted to a real since we are multiplying by a real number. It would, however, have changed the ENERGY column to Real8 (ie double) format instead of keeping the Real4 (float) format.

The name of the energy column in the output file (here "evt2_hacked.fits") will be in upper case − whatever its case in the input file − since its values have been changed by dmtcalc.

Since "expression" is a required (automatic) parameter, you may be tempted to enter something like:

```
  unix% dmtcalc evt2.fits evt2_hacked.fits "energy=((float)pha)*0.1"
```

This will fail because the parameter parser interprets this as setting a (nonexistent) dmtcalc parameter named "energy" to "((float)pha)*0.1".

Example 3

```
dmtcalc evt2.fits evt2_hacked.fits
expression="newpha=(pha+(long)(1.25e−3*(time−TSTART)))"
```

The contents of the NEWPHA column are set equal to the PHA column added to the integer part of

```
1.25e−3*(time−TSTART)
```

The data type of the output column is integer (Int4), since the PHA column is an integer and the time expression has been converted (also called cast) to an integer value (here of type "long").

Example 4

```
dmtcalc evt2.obs1,evt2.obs2 evt2.combined
expression="newpha=(pha+(long)(1.25e−3*(time−TSTART)))"
```

This does the same as the previous example except that the input is now two files − evt2.obs1 and evt2.obs2. There is only one output file − evt2.combined − which contains the NEWPHA column. The row order of the output image is determined by the order of the files in the input stack: the first files contents, then the second file, etc until all the input files have been processed. The dmsort tool can be used to change the row order of the output file if required.

Example 5

```
dmtcalc lc.orig lc.smoothed expression="rate=(rate:9−0.123)"
```

Here the file lc.orig (which is assume to be a lightcurve so contains a rate column) is converted to lc.smoothed in which the RATE column is equal to the smoothed − by 9 rows − value of the rate and then a dc offset (0.123) is subtracted from it. The value for the dc offset would have to be calculated prior to running the program − e.g with dmstat as shown in the following csh/tcsh example

```
dmstat "lc.orig[cols rate]" sigma−
set dc=`pget dmstat out_mean`
dmtcalc lc.orig lc.smoothed expression="rate=(rate:9−$dc)"
```

Example 6

```
dmtcalc evt2_bary.fits evt2_phase.fits @calcphase.lis
```

The expression can also be stored in an ASCII file and accessed as a stack. Here we use the contents of calcphase.lis to create an output file in which two new columns have created − PHASE and GPHASE − which contain phase information for the events. The contents of the stack file are:

```
unix% cat calcphase.lis
.dtime=(time−TSTART)
GPHASE=.dtime*19.794885
PHASE=GPHASE−(long)GPHASE
```

We have also included the use of a temporary variable (here .dtime) even though it is not really necessary for this particular example.

The results are only valid if the frequency of the source is 19.794885 Hz (and it's period is not changing, and you have applied the barycenter correction to the events file). For more details on using dmtcalc to add phase information to an event file see the Create a Phase−binned Spectrum thread on the CIAO website.

Example 7

```
dmtcalc evt2_bary.fits evt2_phase.fits
expression=".dtime=(time−TSTART),GPHASE=.dtime*19.794885,PHASE=GPHASE−(l
ong)GPHASE"
```

Here we repeat the previous example but give the expression directly − i.e. without the use of a stack file.

Example 8

```
dmtcalc evt2.fits evt2_hacked.fits @randpha.lis
```

```
  unix% cat randpha.lis
  newpha= pha + (long) \
    ( 1.25e−3*(time−TSTART) + 10*(#rand(1034)−0.5) )
```

Here we define the expression in an external file and use the "\" character to indicate a continuation line. The contents of the NEWPHA column are set equal to the PHA column added to the integer part of

```
1.25e−3*(time−TSTART) + 10*(#rand(1034)−0.5)
```

where #rand(1034) means a random number between 0 and 1 where 1034 is used to seed the random number generator. This seeding only happens once − ie it is not re−set for each row of the table.

The result is similar to the earlier example, except that a random element of +−5 has been added onto the calculation.

Example 9

```
dmtcalc srclist.wav srclist_filt.wav
expression="if(psfratio==#nan)then(psfratio=0.0)"
```

Here we use the support for conditional expressions to filter out the NaN values from the PSRATIO column of srclist.wav, replacing any such values by 0.0.

Example 10

```
dmtcalc srclist.wav srclist_filt.wav expression="scol=shape~='cir'"
```

This expression creates the column SCOL which contains boolean values: TRUE (1) or FALSE (0). The row entry is true if the entry shape column contains the string "cir" and false otherwise. The "~=" operator returns true if the string on the left−hand side contains the string given on the right−hand side.

Parameters

name	type	ftype	def	min	max	reqd	stacks
infile	file	input				yes	yes
outfile	file	output				yes	
expression	string	input				yes	yes
kernel	string		default				
clobber	boolean		no				
verbose	integer		0	0	5	no	

Detailed Parameter Descriptions

Parameter=infile (file required filetype=input stacks=yes)

Input file to be operated upon. A copy will be made of this file, with any additions/modifications indicated by the expression parameter.

Unless an extension is specified when the file is input, the first table extension will be used. A stack of files may be input, but they must all have the same structure − ie have the same extension name and contain the same columns − since the columns in the first file are assumed to exist − in the same order − in all subsequent files. The behaviour of the tool is not guaranteed if this condition does not hold.

When a stack is supplied, the input files are processed in the order they are supplied to create one output file. The dmsort tool can be used to change the order of rows in the output file.

Parameter=outfile (file required filetype=output)

Name of output file to create.

The output file will have the same extension name as that of the first input file. All the keywords and subspaces in the first file will be copied to the output, in addition to any new columns defined in the expression parameter.

Parameter=expression (string required filetype=input stacks=yes)

A stack of expressions to be evaluated for each row in the input file.

If many expressions are desired, then they should be put into an ascii file, and input as @filename. There is no limit on the number of expressions that can be evaluated. When providing expressions on the command line, note that spaces may be interpreted as stack separators, which can produce unexpected error messages. See "ahelp syntax" for a description of the syntax allowed here.

Parameter=kernel (string default=default)

Data Model kernel of output file.

The possible values are default, which uses the type of the input file, IRAF, or FITS.

Parameter=clobber (boolean default=no)

Clobber the output file?

Parameter=verbose (integer not required default=0 min=0 max=5)

Level of verbose information printed during processing.

CREATING VECTOR COLUMNS

In order to make a new vector column with dmtcalc, it must be used in conjunction with other CIAO tools. Here are two methods for creating a vector column "skymod(xmod,ymod)":

Method 1: Using fits_update_key

Run dmtcalc to create the new columns:

```
dmtcalc old.fits new.fits expr="xmod=(((x−...),ymod=(((y−...)"
```

Next, list all keywords beginning with M:

```
dmlist new.fits header,clean,raw | egrep ^M
```

We are interested in the MTYPE/MFORM keywords. The MTYPEn keyword lists the name of the vector and the matching MFORMn keyword gives the columns in that vector: so MTYPE2 may be "sky" and MFORM2 "x,y" to indicate the "sky(x,y)" vector. If the final pair is MTYPE7/MFORM7, create the new vector as number 8. Start chips and run

```
chips> fits_update_key( "new.fits[events]", "MTYPE8", "skymod", "" )
chips> fits_update_key( "new.fits[events]", "MFORM8", "xmod,ymod", "" )
```

(Due to a bug, this step cannot be done with dmhedit.) It's important to include the name of the block in which the columns exist (e.g. "events"). This step could also be done in a script evaluated by slsh, e.g. if edit.sl contained

```
  require("chips");
  fits_update_key( "new.fits[events]", "MTYPE8", "skymod", "" );
  fits_update_key( "new.fits[events]", "MFORM8", "xmod,ymod", "" );
```

Run "slsh edit.sl" to add the keywords.

"dmlist new.fits cols" should show that there is a vector column called "skymod(xmod,ymod)".

Method 2: Creating Temporary Files

This method avoids the need for finding out what number to use for the MTYPE/FORM keywords at the expense of creating a few temporary files. First create the skymod column and generate a new output file (temp1.fits), then filter the original sky column out of that file (temp2.fits):

```
dmtcalc old.fits"[cols x,y]" temp1.fits expr="xmod=...,ymod=..."
dmcopy temp1.fits"[cols −sky]" temp2.fits
```

The Data Model filter "[cols x,y]" indicates that only those two columns should be included in the new output file. This means that there will be only one vector and one MTYPE/MFORM pair. Update these keywords with the new vector and component names:

```
dmhedit temp2.fits filel="" op=add key=MTYPE1 value='skymod'
dmhedit temp2.fits filel="" op=add key=MFORM1 value='xmod,ymod'
```

Now paste the rest of the columns from the old file to the new one:

```
dmpaste old.fits temp2.fits new.fits
```

CHANGES IN CIAO 3.2

Exit Status

If the tool is unable to evaluate the input expression, it exits with a non−zero status.

CHANGES IN CIAO 3.0

The dmtcalc, dmgti, and dmimgcalc tools now use the same syntax for defining mathematical expressions. The "ahelp dmsyntax" file − which used to be available as "dmtcalc_expressions" − describes the allowed syntax.

WHITESPACE IN EXPRESSIONS

When an expression is given on the command line − so not as a stack file − then spaces are important. This is because spaces are treated as stack separators − as discussed in "ahelp stack" − which is unlikely to be the desired behaviour here.

So, setting

```
expression="energy= (float) pha / 10.0"
```

will not work. You have to say either

```
expression="energy=(float)pha/10.0"
```

− i.e. remove all the spaces − or surround the expression in ()'s, since this stops the checks for stack separators. This means that

```
expression="energy=( (float) pha / 10.0 )"
```

is a valid expression.

An alternative solution is to write the expression in a text file ("expr.lis" say) and then set

```
expression="@expr.lis"
```

since the white space in stack files is not important.

Bugs

See the bugs page for this tool and for the mathematical syntax support on the CIAO website for an up−to−date listing of known bugs.

See Also

dm

dmcols, dmfiltering, dmopt

tools

dmappend, dmarfadd, dmgroup, dmgti, dmimgcalc, dmjoin, dmmerge, dmpaste, dmsort, dmtype2split, mtl_build_gti, syntax

---

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