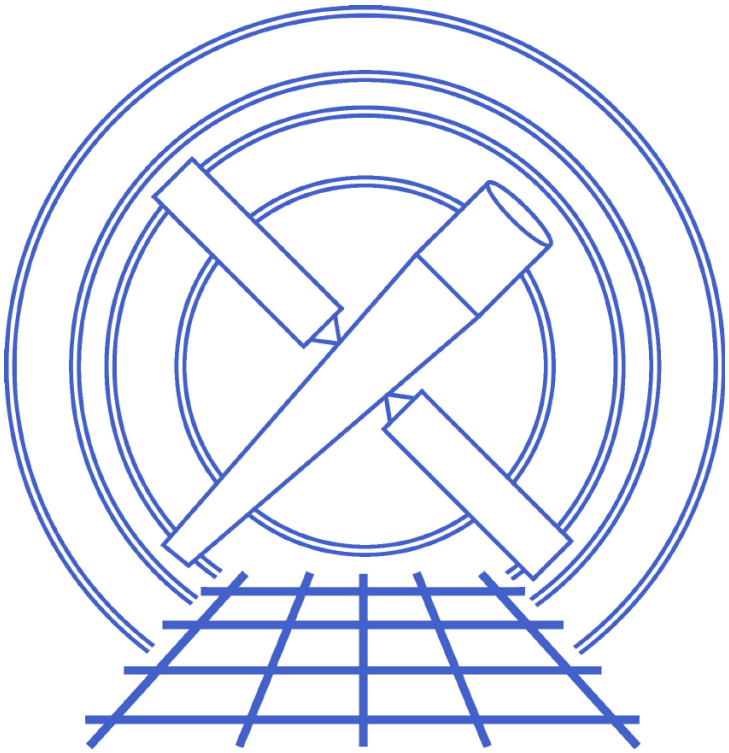
CXC-DM-013

CXC Data Model



Vol. 13

Using the ASCII Kernel

Jonathan McDowell
Chandra X-ray Center
July 29, 2005

Contents

1 Overview

This document describes the prototype ASCII kernel for the CXC Data Model. This will allow the DM tools to operate on simple text files as well as FITS and IRAF files.

The CIAO DM allows users to filter and manipulate tabular data in a convenient way. In user analysis, it's often more convenient to operate on ASCII text files which can be manipulated in editors like emacs, instead of using binary formats like FITS. (The CIAO DM also operates on images; however, the ASCII kernel does not support any ASCII image format. It does let you bin ASCII text tables to FITS images.)

The ASCII kernel allows you to do the following operations:

- Supporting DM filtering on text files which will be used to interoperate with other text-file-based tools.

  ```
  dmlist "foo.txt[time=100:1000,energy=10:20]" data
  ```

- Dumping FITS files to a form usable by other code:

  ```
  dmcopy foo.fits "foo.txt[opt kernel=text/sm]"
  ```

- Creating FITS files by making text files in emacs and then doing

  ```
  dmcopy foo.txt "foo.fits[opt kernel=fits]"
  ```

  (This can be done in an ugly way with FTOOLS' fcreate, but it needs you to make several different input files.)

- Using external text-based tools in a thread, propagating full header and ending up back in FITS:

  ```
  dmextract "event.fits[bin pi]" "pha.txt[opt kernel=text/full]" type=pha1
  text_spectrum_program pha.txt > new.pha.txt
  dmsort new.pha.txt "sort.pha.fits[opt kernel=fits]"
  ```

These different cases are best supported by several different flavors of text file. For many cases, we just want DM filtering functionality on a simple text file of columns of numbers, of the kind that astronomers typically use. However, for FITS file creation and mixed use of FITS and text in an analysis thread, we want a text format which supports full equivalence with a FITS file.

Note that in the above examples, we use a DM directive ("`[opt kernel=blah]`") to qualify an output file name - this is new, up to now we've only had DM directive on the input file names to tools. The "[opt..]" directive is the only one you can use this way, you (obviously?) can't filter or bin output files. The dmcopy tool has a "kernel" parameter, and you can use this instead (the opt kernel directive will override the kernel parameter if both are used); the advantage of the [opt kernel] syntax is that it can be used in any tool.

2 Using text files with the DM

The simplest kind of ASCII text table contains just free-format columns of numbers. Often, however, the file also contains some kind of simple metadata which we have to either read or work around. For this reason, the ASCII kernel does its best to guess what is going on in the file, but also supports a range of special options to let users describe the file's peculiarities.

We recognize lines beginning with # as comment lines, and by default treat the first such line as defining the column names. So, a simple file looking like

```
1.4 5.2 1
2.3 1.8 0.99
4   11  12
```

will be interpreted as a three-column table with columns called col1, col2 and col3, all of type double. Only the first data line is parsed to identify the columns, so to handle to all-too-common case where - like here - the first line has something that looks like an integer but later lines are floating point, we don't try and distinguish between integer and floating point in these simple files.

If you add

```
# Time Energy Factor
#  s    keV    -
# List of energy versus time

  1.4 5.2 1
  2.3 1.8 0.99
  4   11  12
```

the columns get named "Time", "Energy", "Factor", and the DM adds two comment lines containing the "s keV -" etc. (We don't have a way of specifying units or data types in this format yet; that may come later). Blank lines are ignored.

For string-valued data, note that comparisons are done ignoring trailing spaces. Strings are assumed to be space-delimited.

3 Adding FITS headers to text files

If you want to add real FITS-style header keywords, the file becomes CIAO DM's standardized Data Text Format. For a single table, all you need to do is make sure you put the END keyword in so CIAO can figure out where the data starts.

```
HKEYNO = 5.8 / a header key
END
... (data)
```

You may want to add TTYPE keywords to name the columns:

```
HKEYNO = 5.8 / a header key
TTYPE1 = 'TIME'
TTYPE2 = 'COL2'
TTYPE3 = 'PHA'
TFORM3 = '1J'
TTYPE4 = 'N2'
TTYPE5 = 'N3'
END
  78247473.5340545774    2    0         57        1
  78247473.5750945807    7    2         57       66
  78247473.5750945807    7    2         57       64
  78247473.5750945807    7    2         57       72
```

If you want to add a second HDU, you need to write

```
XTENSION='TABLE'
```

as its first keyword, following the data from the previous HDU. When the DM writes out one of these files, it will also write

```
#TEXT/DTF
```

as the first line of the file to make it clear what format it is, and allow for the possibility of version markers (#TEXT/DTF-2.0 etc) in later versions. However, when making your own files you don't need this line (at the possible minor cost of later non-back-compatibility).

This DTF format is conceived as an ASCII version of FITS. The END key is mandatory; data begins on the first non blank line following the END. New HDUs are allowed with lines beginning with XTENSION. This allows us to have full FITS table (but not image) support, and gives an easy way to mock up a FITS file (with dmcopy dtf.txt dtf.fits"[opt kernel=fits]"). You can add the usual TUNITn, TFORMn keywords.

3.1 Fixed format tables

By default, the DTF XTENSION="TABLE" tables have free format data, but you can have nicely formatted fixed-field-length DTF files using the TBCOLn and TDISPn (but not TFORMn) keywords to give starting byte positions and Fortran-style formats, in FITS XTENSION="TABLE" style.

4 Tweaking the ASCII kernel behavior

The following options are used to qualify a text file, and may be combined in a comma-separated list. The first three are used on output files to control the form of the output; you may wish to experiment to see the differences.

- `[opt kernel=text/dtf]` Force the output to be DTF style.
- `[opt kernel=text/simple]` Force the output to be raw data columns with no header.
- `[opt kernel=text/sm]` Force the output to be "SM style" with # as the default comment character.
- `[opt sep=:]` Define : (or some other character,e.g. \t) to be the separator for data fields. There are two kinds of field separator methods: whitespace, in which multiple separators in a row are treated as defining only one new field, and single-separator, in which each separator character defines a new field, so that

  ```
  14.1::23.2:15.1
  ```

  represents 4 fields, with the second one being empty.

  We specify the separator method with `[opt sep/white=:]` and `[opt sep/single=:]`. If the qualifiers are not present, the default is whitespace if the list of separators includes a space, and single-separated otherwise.

  You can use more than one character as a separator, for instance `[opt sep=":;"]` defines both : and ; to be separators.
- `[opt skip=3]` says to skip three lines at the beginning of the file. This helps handle some formats with fixed headers.
- `[opt head=#]` says that lines beginning with # (this is the default) prior to the first data line will be treated as comments, except for one special line which the colnames directive controls:
- `[opt colnames=first]` The first comment-character line is treated specially, as a space-separated list of column names. This is the default; the other options are colnames=last (the last comment-char line prior to the first data line) and colnames=none (none of the lines are treated as a colnames definition).
- `[opt quotes="\'\""]` Define a list of characters to be used as quotes to delimit string-fields. The first character in the list will be used as the default quote character for writing.

The default options for text/simple are:

```
comment='#',sep=" \t\r",quotes="\"'",colnames=none,skip=0
```

For text/sm, we have:

```
comment='#',sep=" \t\r",quotes="\"",colnames=first,skip=0
```

For text/dtf, we have:

```
sep=" \t\r",quotes="\"'",colnames=none,skip=0
```

5 What kind of file?

When DM opens a file, it has to figure out what kind of file it's reading.

First it checks to see if it is a FITS file; if not, it tries to interpret it as an ASCII file of some flavor.

1. If the kernel option has been set by [kernel=...], the kernel is matched with that option. On read, if the file can't be opened with that kernel and flavor, the open fails.
2. If kernel flavor parameters have been set (e.g. comment char, separator, etc.) these parameters are used in the autodetect below.
3. If the file does not yet exist (i.e. we're creating a file) and no kernel option is used, the selected kernel is determined from the create or copy tables as currently done in the kertable code.
4. If the file does exist (we're reading) and no kernel option is used, the kernel is autodetected as defined below.
5. We loop through all defined kernels. Right now that'll mean trying to open as FITS first, then trying ASCII if that fails.
6. When trying to open using the ASCII kernel, we apply the rules listed below.
7. The kernel tests the first line of the file (after skip if applied) to see if it begins with one of the following:

   ```
   #TEXT/SIMPLE
   #TEXT/SM
   #TEXT/DTF
   #TEXT/DTF-FIXED
   ```

   - if you are creating a file, this lets you make sure DM will interpret the file a particular way. If the first line begins with `#TEXT/` and doesn't match any of the know flavors, the open should fail.

8. Otherwise, it tries to autodetect based on the data.
9. The kernel applies the `[opt skip=n]` command line option.
10. The kernel skips any empty lines.
11. Then,
    - If the first line is # followed by a mixture of whitespace and alphanumerics, the line is interpreted as a set of column headings and the flavor is TEXT/SM.
    - If the first line looks a bit like a FITS keyword (name = value), the flavor is TEXT/DTF. More specifically, if the line is: alphanumeric string with no whitespace, followed by one set of optional whitespace, followed by an equals sign, that's enough to recognize the flavor. In particular, leading space is not allowed.
    - If the flavor is TEXT/DTF, check then or later for TBCOL/TBFORM to spot if it is DTF-FIXED.
    - Otherwise, the flavor is TEXT/SIMPLE.

6 FCREATE compatibility

FCREATE uses three special text files - a columns file, header key file, and data file - to make a FITS file. For compatibility, we support

```
dmcopy  "fcreate(colsfile,hdrfile,datafile)" foo.fits
```

7 Limitations

The ASCII kernel is particularly useful in interactive data analysis, to allow CIAO users to use the familiar DM syntax in manipulating and filtering text files they may have created for other reasons. It is not intended as a replacement for the FITS kernel in pipelines. Details of some limitations are given in this section.

7.1 Problems

1. The ASCII kernel only works for text files of limited size: maximum table length 50000 lines, maximum size of a header 80 kbyte, maximum line length 10 kbyte.
2. The ASCII kernel does not read or write images. Therefore, you can use dmextract to make a simple ASCII equivalent of a PHA file, but not one with a WMAP image.
3. If you want to create a dataset with more than one block, or write header keywords, you have to use the DTF flavor. Therefore, the Simple and SM flavors cannot be used with many CIAO tools which require multiple blocks and header keywords.