CXC-DM-007

CXC Data Model

Vol. 7

ASCII Kernel Design

Jonathan McDowell
Chandra X-ray Center
June 25, 2003

Contents

1 Overview

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

The ASCII kernel will cover a range of text-based formats. This is consistent with other kernels - for instance, the FITS kernel covers FITS-ASCII and BINTABLE flavor for table support. This also makes sense: for example, the code to support space-separated, #-headered (SM style) tables and tab-separated, above-the-dashline-headered (RDB style) tables has a 99 percent overlap, with very minor differences at the parsing/formatting stage. So it would be silly to make them two different kernels.

The major use cases for the ASCII kernel include:

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

  ```
  dmcopy foo.fits foo.txt kernel=text/sm
  ```

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

  ```
  dmcopy foo.txt foo.fits kernel=fits
  ```

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

- 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
  ```

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

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

  (Note that I propose here a new DM syntax function: specifying the kernel in the output filename with filename[kernel=foo] - this would allow us to omit explicit handling of the kernels in the program. Kenny will hate this, but the users will like it).

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.

The simpler formats don't support the full DM functionality, but that's perfectly OK. For a format which does not support e.g. coordinate systems, attempts to write a coordinate system should return dmSUCCESS - the idea is that in this format, the coordinate system was successfully written to /dev/null - and code should always be able to handle the possibility that reading from a file will find no coordinate systems on the data.

More seriously, this proposal does not include any support for image blocks. These should be added in a later release - being able to make a very small test image in emacs would be really handy - but I don't think we need to support them in an initial release. However,

```
dmcopy "foo.txt[bin x=32,y=32]" foo.fits[kernel=fits]
```

(binning from a text table to a FITS image) should work.

2 Definition of the flavors

I define the following flavors:

- TEXT/SIMPLE: Simple text table format consisting of free-format whitespace-separated columns, with no header. This format does not support the full richness of the DM's model.
  - Attempts to create more than one block in a TEXT/SIMPLE file will fail.
  - On read, blank lines and lines beginning with # are ignored. They are not created on write. However, an initial line beginning with # and containing any other text will cause the file to be read as TEXT/SM rather than TEXT/SIMPLE.
  - Columns are given the default names COL1, COL2, ...
  - All columns are scalar;
  - Column data types are determined by the first line of the file.
  - There are only two supported data types: dmDOUBLE and dmTEXT. String columns are recognized because the strings are enclosed in single quotes (we could

consider also supporting double quotes), all other data is assumed to be numeric. On write, all non-text data types are mapped to double. (Hmm.. we could also support integer, if we require that the user must type a decimal point for all columns to be interpreted as real, e.g. "11." not "11", as per Fortran literal constants...)

Example:

```
'M33' 4.8 -12
'Arp 220' 210.312 990.1
Arp 220; 210.312 990.1
```

This has three columns with two rows. The columns are named COL1,COL2,COL3 and given string, double and double (or perhaps integer) type.

If we decided to support integers, the value of COL3 in the second row is truncated to 990 because COL3 is seen as an integer column; you can change the first row to -12.0 if you want to force the column to be seen as floating point.

- NaN values are indicated by the string 'NaN'. Null values (which by default will also be NaN) can be indicated by either '-' (an isolated dash with a blank on either side; JCMLIB compatibility) or 'INDEF' (FITS compatibility). Perhaps the "[opt null=val]" DM directive can be supported on write.
- There are no subspaces or coordinate systems, and no header keys.

- TEXT/SM: As TEXT/SIMPLE, but with a header consisting of a single line of column names preceded by the # character. (this format is compatible with the SM plotting program).
  - Column names containing round or square parentheses are interpreted as defining vector and array columns: example

    ```
    #  TIME    POS(X,Y)    PHAS[3]    NAME
       1012.3  14.2 13.8   -11   2 0  "WX Hyi"
    ```

    The number of columns in the file must match the total number of component columns defined in the header line.
  - One can imagine extending this format to support additional header lines defining other column attributes (units, etc) and to support header keywords. This would not break the sort of external software that uses such files, but I don't feel this is an immediate priority; current use by astronomers extends to just a single meaningful header line with only scalar columns, and other #-lines with comments only.

- TEXT/RDB: As TEXT/SIMPLE, but with tab-separated fields, and with a header compatible with the RDB program; only scalar columns are supported. The RDB header consists of a line with dashes, above which is a line with column names.

A TEXT/STARBASE extension might be considered to support John Roll's STARBASE format, which is an extension of RDB with a more extensive header. In the Starbase extension to RDB (John Roll, cfa-www.harvard.edu/~john/starbase) the first line of the file is the table title. Unfortunately in this format there's no way to easily tell that what you have is a starbase file without parsing the whole file. In the header section, lines with tabs are keyword/value groups, and everything else is comments until you get to a line with only dashes and tabs. The PREVIOUS line is then the column names.

The data section consists of tab-separated fields. Sexagesimal format is supported (interpreted by DM as a string data type).

Multiple tables are separated by the CTRL-L character.

Some prototyping will be needed to determine if support for Starbase is reasonably easy and useful; it is not an immediate priority.

- TEXT/CDS: Support for the CDS-Strasbourg astronomical catalog data format also used by the AAS. This consists of a structured README file defining the table fields and pointing to the individual catalog files. See http://vizier.u-strasbg.fr/doc/catstd-3.1.htx for the definition. This would support a file with multiple table blocks, and limited header keyword capability, but otherwise with the same limitations as TEXT/SM. It would be really useful, but because of its rather different style might be left as a second priority.

- TEXT/FULL (or TEXT/DTF?) and TEXT/FIXED: DM (or Data) Text Format. Supports DTF, a proposed pseudo-FITS format with support for the full panoply of headers and data subspaces. See below.

- TEXT/FCREATE: We should consider supporting, at least on output, a format compatible with the FTOOLS FCREATE program.

The DM will recognize each of these flavor automatically on read; kernel options will select the flavor on output: "kernel=TEXT/RDB".

2.1 Recognizing flavors

A DTF file is recognized by with an initial "#TEXT-DTF" in the first line. If this is not present, an attempt is made to identify the file as TEXT/RDB (searching for tabs and the second line with dashes), then TEXT/CDS (if supported), TEXT/SM (looking for a first line with # and one or more words) and finally TEXT/SIMPLE.

3 DTF files: A text analog to FITS

The DTF (TEXT/FULL and TEXT/FIXED) is an attempt to support the full DM in text format. Free format tables are the default, but fixed-format fields are also supported.

It would be fairly easy to modify the syntax described below to be a more XML-like

```
<DTF>
<COL NAME=colname TYPE=datatype ... >
<DATA>
...
</DATA>
</DTF>
```

I decided not to go the XML-like route for now because the intent of DTF is to specify a format which is easy for manual data entry and inspection, and I personally don't find that XML satisfies that criterion.

Instead, I decided to leverage the familiarity of many astronomers with the FITS format, take advantage of the existing FITS kernel parsing code, and provide a natural way for users to create text files whose mapping to FITS files is natural. I propose to take the FITS TABLE/BINTABLE design and make minimal changes to turn it into a text format:

- The 2880-byte block structure is abandoned in favor of variable-length ASCII text records.
- Null primary headers are not required; image extensions are not yet supported. Thus, the first HDU is a table.
- The keyword names are not limited to 8 bytes.
- The keyword value is free-format, and quotes are optional on text values if they cannot be interpreted as valid numeric values and do not include the / character (which starts the comment section).
- The file begins with a #TEXT-DTF keyword, considered equivalent to #TEXT-DTF-1.0. Later revisions to the format may begin with #TEXT-DTF-1.1, etc.
- All headers begin with an XTENSION keyword with supported values DTF (free format data) and DTF-FIXED (fixed format data).

- The TFIELDS, NAXIS1 and NAXIS2 keywords are optional. Data begins with the first non blank line after the END keyword and ends with end-of-file or a new XTENSION keyword.
- (for discussion:) keywords must be/must not be/optionally may be prefixed with the # character. Arguments against the prefix: it's more natural for the FCREATE use case to just enter the keywords, it's more FITS-like, it's easier for the user. Arguments for the prefix: it's a natural extension of the TEXT/SM case, more compatible with other text-based tools like sm, it's easier to pick out the header sections with grep. Personally I'd like to support both sub-flavors.
- DTF-FIXED data uses the FITS TABLE keywords TBCOLn and TFORMn to specify byte position and format for each column, for the sake of old Fortran programmers like me who like nicely lined up columns. Normal DTF data is free-format and parsed on each line by space and (for text data) quote delimiters in the same way as TEXT/SIMPLE data.
- The special keyword INCLUDE, followed by a filename, directs the kernel to include the contents of that file (like a C include file).
- The special keyword pair FCREATE_COLS and FCREATE_KEYS, with no value, specify that lines between the two keywords should be parsed as FCREATE column definitions, see below.

I mocked up evt2 file with a few rows of data (the rows have been truncated) at http://hea-www.harvard.edu/~jcm/asc/data/dtf1.txt

3.1 From DTF to FCREATE

The FCREATE program, part of GSFC's FTOOLS package, uses three input files (a column description file, a header file, and a data file), all in ASCII, to create a FITS file.

Each line of the column description file has

```
name type/format unit
```

It can also have (for fixed format) character position and width of the column in the input data file.

Datatypes are (with r a multiplier giving the array size):

rL	logical
rAw	character string of unit length w, but total width r
rX	On/Off bits
rB	unsigned 8-bit byte
rI	16-bit integer
rJ	32-bit integer
rE	single precision floating point
rD	double precision floating point
rC	single precision complex value
rM	double precision complex value

Data types and formats for TABLE extensions are:

Aw	character string,
Iw	integer
Fw.d	single precision, fixed decimal point
Ew.d	single precision, exponential notation
Dw.d	double precision, exponential notation

An example column description file would be

```
time  1D s
ccd_id 1I
chipx 1I
chipy 1I
```

This format can be supported within a DTF file using the FCREATE_COLS keyword; after an FCREATE_COLS keyword, text is assumed to be FCREATE column description text until the FCREATE_KEYS keyword is found.

The header file contains header keys of the format

```
key = value / comment
```

The = and / are optional. Again, this can be supported within a DTF file as long as a leading # is not required.

The data file contains free or fixed format data in a format compatible with the DTF/TEXT flavors; only INDEF is acceptable as a NaN indicator.

The DM ASCII kernel requires a single filename to point it to all three of these files.

We can do this with the DTF FCREATE, KEYS and INCLUDE commands we can make a fourth file, dtf.file, containing:

```
#DTF
#FCREATE_COLS
#INCLUDE col.file
#FCREATE_KEYS
#INCLUDE header.file
#END
#INCLUDE data.file
```

Alternatively since #-lines are ignored by FCREATE, we can put these lines (omitting one of the includes) in the header file or the data file (the FCREATE column description file does not necessarily support comment lines).

3.2 A special fcreate DM directive

The ASCII kernel dataset open command could support the special syntax 'FCREATE(colfile,headerfile,datafile)' to avoid you having to actually make this dtf.file:

```
dmcopy "fcreate(col.file,header.file,data.file)" fc.fits kernel=fits
```

is defined to be equivalent to

```
dmcopy dtf.file fc.fits kernel=fits
```

with the contents of dtf.file as listed above. One can also do

```
dmcopy fc.fits "fcreate(col.file,header.file,data.file)" kernel=ascii
```

to create the three files.

4 DTF-COMPACT files: a proposal for the future

Here I propose a possible alternate design for a text file, which maps more closely to the DM design, with a common format for all the types of descriptor. On the other hand, it's a totally new invention, which may be considered an argument against it.

DTF-COMPACT files are like FITS files in that their blocks are organized as header-data units, with header sections followed by data sections. Each line in the header section begins with the character #, except that after the first line of the file (which must start with #TEXT-DTFC), blank lines may occur as desired in either the header or data sections and will be ignored.

commands can be continued onto subsequent lines if the last non blank character of the line is a backslash and the next line is # followed by space.

The data section of a DTF block begins with the first non-blank line following a #END line. It ends with the end of file or the next #TABLE or #IMAGE line.

The choice of # as a special character is motivated by its widespread use as a comment character in scripting languages and in astronomical software. Thus, many programs will handle the data sections of simple (single-table, no-text-column) DTF files correctly and will simply ignore the DTF header.

4.1 DTF descriptors

Columns are defined by the following compact syntax, intended to appeal to astronomers who don't mind the complicated cases being a bit cryptic as long as they don't have to type much for the simple cases:

```
#COL:pos colname(cptname,cptname)(n1,n2...)[unit]:fmt:type
```

or by a more verbose syntax, designed to appeal to programmers who like an easily parsed, explicit and verbose style:

```
#COL colname(cptname,cptname)(n1,n2...) TYPE=datatype UNIT=unit DESC=desc POS=pos FORMAT=fmt
```

```
#COL:42 aspimg(5,5):i4[erg/s] / Fluxed image
```

```
#COL aspimg(5,5) TYPE=i4 UNIT='erg/s'  DESC=Fluxed image; POS=42 FORMAT=F5.2
```

The order of the #COL definitions in the header corresponds to the order of the columns in the table.

Keys are defined by

```
#KEY instrument = acis
```

or

```
#KEY instrument value=acis
```

Vector and array values are supported:

```
#KEY CEL_NOM(RA_NOM,DEC_NOM)  value=(235.82,-20.13) unit=deg
#KEY COEFFS(5) value=[1.3E-5,218.2,-40.1,3E-6,-1.001E-12]
#KEY POSLIST(X,Y)(2,2)[pixel] = [(0,0),(4,5),(3,3),(1,1)]
```

As for FITS, we recommend that only letters, digits and underscore be used in column names.

Arraying is indicated by square parens, vectorizing by round parens, and vectorizing parens are always inside arraying parens. Multidimensional array elements are listed in the order (1,1),(1,2)...(1,n),(2,1)....(m,n). [TBR].

Array specifications are distinguished from vector component names by the fact that they are integers. (Therefore, names that are purely composed of digits are not allowed).

The general form of the long form descriptor syntax is

```
#desctype name(cptnamelist)(arrayspec) attribute=value attribute=value ...
```

Either the cptnamelist or the arrayspec may be omitted. If the cptnamelist is omitted, the descriptor is a scalar or scalar array descriptor (as opposed to a vector descriptor). If the arrayspec is omitted, the descriptor is a simple scalar or vector descriptor rather than an array one.

Attribute values which are strings may be enclosed in single quotes. In which case a literal single quote in the string may be escaped as pair of single quotes:

```
DESC='Stephan''s Quintet'
```

(consistent with FITS practice).

Alternatively, the quotes may be omitted. In this case the string is terminated by a semicolon character.

The valid descriptor types are:

- KEY Header key
- COL Table column
- COORD Coordinate system
- FILTER Data subspace filter
- AXIS Physical axis group coord system on an image

The attribute types are:

- TYPE: The data type for the field. Valid values are i2,i4,r4,r8 corresponding to 2 and 4 byte integers and 4 and 8 byte reals.
- UNIT: The descriptor unit.
- VALUE: The descriptor value (for KEY and FILTER only)
- FORMAT: The suggested display format for the field.
- POS: (for COL only) The starting character position in each row of the field (the first character is 1). If this attribute is absent, the field is free-format. If POS is specified for any column it must be specified for all of them (this requirement may be relaxed in the future).
- CPT: (for FILTER only) the data subspace component, default 1.
- REFVAL (for COORD only) the reference value in coord units; FITS CRVAL
- REFPOS (for COORD only) the reference position in parent units; FITS CRPIX
- SCALE (for COORD only) the pixel size in coord units per parent unit; FITS CDELT
- MAP (for COORD only) the transformation type; FITS CTYPE last half
- PARENT (for COORD only) the parent descriptor (thing being mapped). For non-COORD descriptors, the parent is implicit.

The general form of the compact syntax is

```
#DESC:qual  name(cptname,cptname...)(n1,n2...)[unit]:fmt:type=value / description
```

Any of the attributes not used may be omitted. Spaces are allowed between elements of the syntax. The only required space is that between #DESC:qual and the name.

Here the attributes are:

- The descriptor qualifier, an integer after a colon following the descriptor type name. For COL, this is the POS attribute. For FILTER, this is the CPT attribute. Ignored for other descriptor types.
- The unit, in square parens following the name and array specification. This syntax is motivated by the familiarity of the CFITSIO keyword convention. (For compatibility with FITS, this field may also appear after the / denoting the start of the description, but we note that the idea of encoding this information in a comment field is considered a horrible idea born of desperation even by those who proposed its use in FITS. The field may also appear after the value and before the descriptive /.)
- The display format, preceded by a colon.
- The data type, preceded by a colon; if omitted, it is guessed from the display format.
- The value, preceded by an equals sign. For FILTER, the value is a set of colon-ranges or spatial regions in the DM style. For COORD, the value is a string specifying the transformation. For COL, the optional value is a colon-range specifiying the valid range of values and optionally the NULL value in the format min:max or min:max:!null (can also be specified with RANGE= and NULL= parameters). If the value field contains the / character or the [character, the field must be enclosed in single quotes to prevent it being interpreted as a description field.
- The description, following the text ' / ' (slash with space on either side), as in FITS keywords.

4.2 Specifying the data subspace

The VALUE field of a FILTER descriptor is specified by:

- A range list of values:

```
#FILTER PHA VALUE=1:20,40:50,100:
#FILTER PHA = 1:20,40:50,100:
#FILTER TIME = 84721032:84721040,84721050:84729800
```

- A region string, for a 2D filter:

```
#FILTER SKY(X,Y) VALUE=circle(4096,4096,20)*box(4100,4200,20,40)
#FILTER SKY(X,Y) = circle(4096,4096,20)*box(4100,4200,20,40)
```

- A table reference:

```
#FILTER TIME VALUE=TABLE GTI3
#FILTER TIME = TABLE GTI3
```

As always, the complication with the subspace is the case of multiple components. Really you'd like to define the descriptor once and given the values for all of the components, but that would be unwieldy. A special syntax for defining filter component values, or requiring them to be in a separate table, also seems clunky but might be considered. Instead, it seems less ugly to accept the repeated reference ot the descriptor:

```
#FILTER CCD_ID  CPT=1 VALUE=7
#FILTER TIME    CPT=1 VALUE=TABLE GTI7
#FILTER CCD_ID  CPT=2 VALUE=6
#FILTER TIME    CPT=2 VALUE=TABLE GTI6
```

or (short format)

```
#FILTER:1 CCD_ID=7 / For CCD 7...
#FILTER:1 TIME=TABLE GTI7  / Use time intervals in GTI7
#FILTER:2 CCD_ID=6
#FILTER:2 TIME=TABLE GTI6
```

In future we might allow (short format)

```
#SUBSPACE (CCD_ID=7,TIME=GTI7)|(CCD_ID=6,TIME=GTI6)
```

but that will not be supported in an initial release.

4.3 Specifying coordinate transformations

Some examples in compact form:

```
#COORD ENERGY[keV] = 0.05 + 14.6 * PI
#COORD EQPOS(RA,DEC)[deg] = (11:04:28.2,-20:30:11.2) + \
# P-TAN (-0.492",0.492") * ( SKY(X,Y) - (4096.5,4096.5))
```

The general forms are

```
#COORD name =  refval + scale * transform ( parent - refpos )
```

In this format, FITS WCS 2-D projection transforms are preceded by 'P-' (for 'Projection') to prevent confusion between the TAN projection and the tangent mathematical function.

Some examples in verbose form:

```
#COORD ENERGY UNIT=keV PARENT=PI REFVAL=0.05 REFPOS=0 SCALE=14.6 MAP=LINEAR
#COORD EQPOS(RA,DEC) UNIT=deg PARENT=SKY REFVAL=(239.1822,-20.5042) REFPOS=(4096,5,4096.5) \
                     SCALE=(-1.38E-4,1.38E-4) MAP=TAN
```

4.4 INCLUDE command

The keyword #INCLUDE filename causes another file to be included, as if it had been pasted in at that point in the text. This is useful for cases when data sections need to be in separate files. The INCLUDE keyword may occur outside of header sections.

4.5 Other DTF keywords

The DTF-COMPACT file must begin with

```
#TEXT-DTFC
```

or

```
#TEXT-DTFC-1.0
```

(later revisions of the format may require the second form, with an updated version string). Other text may follow:

```
#TEXT-DTFC-1.0  DM Text Format (Chandra X-ray Center, 2003)
```

If the #TEXT-DTF key is missing, software may ignore other DTF keywords and interpret the file as a TEXT flavor file.

The rest of the file is a series of blocks (DTF blocks, which map to DM blocks). Each DTF block should begin with

```
#TABLE blockname
```

or

```
#IMAGE blockname
```

In the first DTF block this may be omitted; the default blockname is the name of the file, and the default block type is TABLE. The TABLE/IMAGE keyword is required in the second and subsequent block names and indicates the beginning of a new block.

The blank keyword (# followed by a space) indicates a comment line.

It's often inconvienient to type in a lot of keys with the #KEY keyword in front each time. The special keyword #KEYS indicates that following lines contain #KEY definitions until a #END keyword is read. This also improves compatibility with FCREATE.

The #FCREATE_COLS keyword indicates that subsequent lines contain column definitions in FCREATE format, until another DTF keyword is found. This is discussed in the FCREATE section.

The header section of a DTF block must end with

```
#END
```

Other # keywords are ignored; warning messages may be issued by software for unrecognized keywords, but they should not cause errors. This allows software to handle upgraded versions of the format in a limited way. An attempt will be made to ensure that such upgrades will still produce a reasonable result if newly defined keywords are ignored, although this cannot be guaranteed.

4.6 An example file in compact format

An DTFC evt2 file with a few rows of (truncated) data is at http://hea-www.harvard.edu/~jcm/asc/data/dtfc.txt

4.7 Data section

The format of the data section is as for a TEXT flavor file, except that if POS attributes were defined for the columns the file must be in fixed format.

In the data section, # keywords (other than #-space comments) are not permitted and may be silently ignored. #-space comments are permitted, ignored, and encouraged.

4.8 Images

The #IMAGE keyword indicates that the data section contains a single array object and that newlines are not significant; data values are read in in the canonical order. Example image:

```
#TEXT-DTFC
#IMAGE arp220
#KEYS
INSTRUMENT = 'ACIS' / Instrument
# Data is blocked by 8 and a small subsection 10x5 image extracted
#COL COUNTS(10,5)[count]:i2  / Image counts
#COORD SKY(X,Y) = (4090.5,4090.5) + (8.0,8.0) * ( (#1,#2) - (0.5,0.5) )
#COORD EQPOS(RA,DEC)=(11:01:28.3,-00:00:14.5) + \
   P-TAN(-0.492",0.492")*(SKY(X,Y)-(4096.5,4096.5))
#END
0 0  0  1  0 10  0  2  0  1  1  1
0 10 12 8  4  3  1  2  1  0  0  2
0 2  11 9 15 21 88 60 16  5  3  1
1 5  13 20 21 92 513 182 14 10 8 3
4 3  10 16 12 40 112 80  33  5 2 1
```