CXC-DM-003

CXC Data Model

Vol. 3

Data Model Library Architecture

Jonathan McDowell
Chandra X-ray Center

December 27, 2001

Contents

1 Goals

The current CDM has a lot of useful functionality, but the internals reflect some early implementation compromises that impede both debugging and adding new abilities. The goal of this redesign is to generate a CDM2 which has the same interface as the CIAO2.2 CDM and passes the same regression tests, but has a cleaner internal design with the following properties:

- Implementation to reflect the original CDM (1994) design, plus modifications due to lessons learned (see 'historical' section).
- Possibly add hooks that may be useful in NVO related work. There are no specific design elements for this yet, but as NVO work gathers steam the DM design will be reviewed for possible modification.
- Consistent and documented design, maintainable and commented code, so that others in CXC can maintain the code.
- Stable and public kernel interface so that others in the community can write new kernels.

2 New Structure

The most important difference is in the layering. In the old structure, kernel routines were cast in terms that were closer to FITS concepts. Thus, grouping of multiple axes and coordinates had to be stitched back together at the DM layer. The new kernel will operate in terms of the DM structures.

The first design decision is to choose between two models of kernel interaction:

- Allow kernel routines to directly fill DM structures, for maximum efficiency.
- Retain a strict layering in which DM structures are filled only via interface routines. This makes it easier for others to write kernels.

I really want it to be easy to write kernels, so I chose the second. However, in our new kernel interface, we will call DM layer routines, as opposed to the old rule in which kernel routines did not call the DM. This implies a new layer diagram.

The original DM layer diagram was:

```
-------------------------------------------
|    C  Application     |  F77 application|
-                       -------------------
|                       |  F77 wrappers   |
-------------------------------------------
|     DM public layer                     |
-------------------------------------------
|    ETOOLS                       |
---------------------------------
| FKER          |   | IQKER       |
---------------------------------
| CFITSIO  |            | IRAF  |
------------            ---------
           |            |
-----------------------------------
| Region      |  EDSUTIL  | WCSLIB|
-----------------------------------
```

In the first DM rewrite, the ETOOLS layer was removed. In the new model, we will have the following layer diagram:

```
-------------------------------------------
|    C  Application     |  F77 application|
-                       -------------------
|                       |  F77 wrappers   |
-------------------------------------------
|     DM public layer (DMP)               |
----------------         ----------------
| FKER          |        | IQKER        |
------------------     ------------------
| CFITSIO | PKI   |    | PKI      | IRAF |
-----------      ------           |-------
          |  DM Internals (DMI)   |
    ------------------------------------
    | Region  | DM Utils (DMU) | WCSLIB|
    ------------------------------------
```

In the old model, both the kernels and the ETOOLS layer made calls to the EDSUTIL utility routines which contained a grabbag of generic algorithms and X-ray-astronomy conventions. In this model, the kernels call a Public Kernel Interface (PKI) which in turn access a DM Internals (DMI) layer. The DM public layer will call both PKI routines and private DM Internals routines. The PKI routines are simply the public part of the DM Internals interface and access the same internal structures. In order to make it easy for people to write new kernels which put and get to/from the internal structures, the PKI needs to be well defined and well matched to these structures, but the DM public layer, which needs to a different kind of bookkeeping and to perform efficient filtering, may need more direct access routines that are not suitable for public use, so it isn't restricted to the PKI.

The small number of EDSUTIL routines and existing kernel routines which are generic will be replaced by a DM Utils layer.

A CXC Tool application has further relevant structure:

```
---------------------------------------------------------------------------
|     CXC  Tool                                                           |
-----------------------------------------------------------------------------
|     DM public layer                    | DMTLIB   | STKLIB | PARAM | DSLIBs |
--------------          ---------- ------------------------------------------
| FKER          |         | IQKER        |                    | DSLIBs|
------------------     ------------------                    ---------
| CFITSIO | PKI   |     | PKI     | IRAF |
-----------       ------        |-------
          |      DM Internals    |
    -------------------------------------
    | Region      |  DM Utils   | WCSLIB|
    -------------------------------------
```

In this diagram, DSLIBs represent the whole set of CXCDS libraries. The DM is separately distributable, and its Core Tools (e.g. dmlist, dmcopy) are being kept as independent as possible of the rest of the CXCDS. Therefore, the DM Core Tools will call the DSLIBs only through the PARAM library. In the current separate distribution, which is not advertised but has been made available to a limited number of users (J. Davis/MIT; Palermo group, who are using the Fortran wrappers), the current CXC PARAM library is replaced by an older version which is not dependent on the other CXCDS libraries, and I will lobby for such a library being formally maintained.

The DMTLIB is used for several DM tools. In the new design, the DMTLIB is expected to be fully absorbed into the DM public layer.

The STKLIB handles stacks. The status of this library's relationship to the DM also needs to be reviewed. Because of the existing code base, I propose that the STKLIB remain as is, but new DM routines be provided to open and manipulate stacks of datasets; these new routines would be phased in slowly. These new routines may not be implemented in the first (CIAO 3.0) phase of the redesign; the main impact on lower layers of the DM is that (1) we may want to have a mode in which you have a 'dmmerge on the fly', so that EOF on a table silently triggers reading from the next table in the dataset stack (but this would imply incorporating all the header merging machinery too) and (2) we want to have the ability to apply a stack of filters to a file without reopening it. This could be done either by a filter design which assumes you are always working with stacks of datasets, or by making the user open a dataset unfiltered and providing an API routine to apply a filter to a row. The former approach may imply adding a lot of extra API get/put routines for the multiple-filter case. The latter approach causes much less grief to the existing API and I propose that solution.

3 DM functionality

All routines in the current DM API will remain, and all current functionality will be supported.

The following DM limitations will be addressed in the redesign:

- Header keyword ordering. There are some limitations on accessing keywords by number that prevent the keyword insertion routines in dmhedit from functioning correctly. This is because of the convoluted way that the current DM layer interacts with the kernels and should be easy to fix in a cleaner design.
- Header keyword grouping. We add the ability to define and manipulate named groups of keywords, to reflect the substructure in our headers.
- Column deletion and insertion. This will always be inefficient on FITS files, but should be supported nevertheless.
- Row insertion and deletion. This will require some modification of the kernel interface, and will also be intrinsically inefficient for FITS.
- Support for explicit buffer flushing. I will also investigate providing the ability to inhibit flushing (for PRISM editor application) but I'm not promising, as this is intrinsically problematic.
- Improved and documented error handling
- Review of functionality now handled by [opt ...] syntax; complete documentation of [opt ...] syntax.
- Improved documentation of bit handling. Support for easier syntax for bit filtering (this is a separate enough problem that it does not have to form part of the initial redesign).
- New functions to manipulate data subspace, deleting its members and testing whether a row is within the subspace of another block.
- Useful wrapper routines to handle primary header handling.

The following future upgrades are planned:

- The ASCII kernel is a high priority and a separate document will describe this.

- Very preliminary ideas for an IDL kernel, which would allow DM filtering and dataset manipulation from the IDL command line, have been discussed with V. Kashyap (CfA) and D. Lenz (RS Inc.). We have established a basic approach and some design drivers (memory management in IDL wrappers, use of the CALL EXTERNAL routine to call DM C code).
- Shared memory and/or pipe interface. CFITSIO and XMM-SAS report limited success in usage of shared memory applications to date, but it seems likely that some form of shared memory or direct bytestream kernel will be of use in the future.
- Integrated support for associating uncertainties with columns.

3.1 Kernels and converters

It's been suggested that we abandon the kernel paradigm, work purely in FITS, and have separate (other format) to-and-from DM converters. Since we want to keep FITS and the DM cleanly layered, there is actually not much design difference implied. The QPOE and ASCII kernels would be replaced with QPOE-DM and ASCII-DM converters, which would contain much of the same code as the kernels. The FITS kernel would stay as is. What we'd lose is the ability to work with ASCII files and mix and match generic unix and DM tools; we'd have to keep converting from FITS back to ASCII to apply ASCII methods to the data. We'd also lose the option to develop a kernel for a format which supports DM constructs that are not fully supported in FITS.

I remain convinced that support for the kernel paradigm is the right thing to do for the long run. That doesn't stop users writing a converter from their favorite data format to FITS and then running the DM tools in purely FITS mode; that's an entirely legitimate thing to do.

4 History of DM development

4.1 Development of the code

The initial concept of the DM was elaborated in 1994. At that point, the CXCDS concept involved a core IRAF component, and the DM offered a path to also support FITS event lists. During the PDR/CDR process, the decision was made to carry out multiple object-oriented design cycles, first with D. van Stone and then with P. Patsis, rather than develop prototype code. This limited the usefulness of the design process. After CDR, it was decided to try and build the DM by reusing the much more limited ETOOLS library already under development. This caused considerable awkwardness in the design and made it deviate significantly from the original concept. Frequent developer turnover made progress very slow except of course for the period when Mike Noble was lead, and thanks to him the DM did successfully support CXCDS development starting in 1998, and the DM tools were welcomed by the community following the public CIAO release in 1999. I then became the developer as well as the science lead, and removed an entire layer from the design, deleting over 10K lines of code. Nevertheless, the existing design still does not fully map to the original concept and is structured in a way that's hard to add new features. Thus, a major redesign is critical to provide a robust DM for long-term maintenance.

In the longer term, the development of an NVO protocol may replace parts of the DM. Nevertheless, having a clean DM will be an essential basis for my own analysis of the NVO issues.

4.2 CDR documents

Section 4.7.7 of DS01 describes the data analysis API. In the language of section 4.7.7, the translation wrappers map to the DM Fortran wrappers, the data manipulation libraries map to the DM public and internal layers, and the DM virtual file syntax implements the API mini-language for describing generic analysis objects. The 'adaptation wrappers' do not exist (I prototyped wrappers for IRAF but it was decided that DS could not support the overhead of supporting them) but may arise if the IDL kernel is ever implemented. In Fig 4.7-14 the kernels were the IRAF IMIO and the TBTABLES kernels. The IRAF kernel is still supported; the TBTABLES kernel was abandoned as ST Tables can be replaced in IRAF with FITS, which has become much more important to the community in the interim. The 'generic data model library' and 'science data model library' layers have been merged, since there was no added value from the lower layer (I think it was code for the ETOOLS layer). The DDF layer is another name for the DM kernel layer. The addition of further kernels is compromised in the current design by the poor match of the current kernel API to the DM fundamental design.

The promised features of the Data Model in 4.7.7 were

- 1D and 2D filtering - implemented
- stacking - supported in stack library
- coordinate systems - supported
- coordinate conversions - supported
- uncertainties - not implemented; a harder problem than anticipated.

- The instantiation of data products in different formats = supported
- support for preferred columns - supported, although most data products don't make full use of it.
- interpolation support - as described, implemented in the CIAO 2.2 dmjoin tool.
- mission independence and genericity of the DM - supported
- 'quantity' support - implemented as DM 'descriptors'
- QPOE support - implemented, although not as fully as FITS
- FITS support - implemented
- EDF support - concept abandoned, merged with QPOE kernel
- ST Tables support - not currently implemented
- IRAF images - implemented
- The shared memory kernel has not yet been implemented.
- OTS libraries: IRAF libraries used; TBTables, PROS, XRAY, EVTIO libraries not used.

4.3 Modifications to the original concept

The discussion in DS01 was not a direct reflection of the original DM design; it reflected the ETOOLS code reuse compromise.

In terms of the original design, generic (vector array) descriptors have now been implemented, but the simple cases (scalars, non-array vectors) often have separate code since they were coded first; in many cases that code could be simplified by treating them as special cases of the generic.

There are two main changes I feel are needed to the original DM concept: a way to modify data subspaces to drop quantities when their filters get too complicated, and a modification of the implementation of 'element types', which were to be used to handle uncertainties.

When filtering, there typically comes a point in data analysis where propagating parts of the history is no longer interesting. I therefore propose a new syntax option to allow editing of the data subspace. For instance, you may want to apply the time filter for CCD4 to a file without also applying the ccdid=4 filter. This syntax will be equivalent to the [cols ...] syntax, but for the columns of the data subspace.

Secondly, I now propose to treat complicated element types using the existing vector column mechanism, rather than add an extra layer of structure. The DM's perceived to be complicated enough without adding further dimensions.

Apart from these changes, the 1994 DM document continues to reflect my view of the correct low-level generic interface.

5 The DM public layer

The DM public layer (DMP) API will be largely unchanged, although DMTLIB routines will be added.

The DM public layer's internal structures will be largely unchanged. However, the PKI will now deal with the mapping from DM structures to kernel structures - previously, the DM layer split things up into kernel-level (actually ETOOLS) concepts and then passed them to the kernel interface. In the new design, it is the kernel's responsibility to translate between DM and kernel concepts.

Some code that was previously in the kernels will now be in the DM layer. Specifically, the row filtering code will be at the DM layer; this will cause minimal performance hit, if any. This will avoid duplicating the filtering code in the kernels, duplicating all the filtering information (ranges and regions) in the kernels, and allows more intelligence in the filtering (easier access to high level info about the objects being filtered). Performance will be improved by having the kernels return more than one row at a time when filtering. At present, when filtering, one row at a time is returned to the top layer. In the new design, a buffer of rows will be returned and an index will be used to mark which rows are good. A memory copy is already done when returning multiple rows in an API call, so there is no downside to this approach and the ability to call FITSIO to return multiple rows at once should result in a measurable speedup. It will be recalled that the current row filtering code was written under heroic time pressure by Mike and was always intended as a stop-gap approach; I have done some rework since then but the original skeleton is still constraining us.

In contrast, much of the code that was in the DM layer for composing vector columns and basis keywords will now be relegated to the DM internals layer and called by the kernels. This avoids the problems caused when the kernels pick information apart in their layer which then has to be stitched back together in the DM layer. In general this fits with the philosophical approach of fitting data into the DM concepts as soon as possible and having the internal work done in terms of those concepts, which will make a lot of the internals more robust.

The coordinate and DSS manipulation code will largely remain at the DMP layer, although the writing and reading parts of the code will be split between the KER and DMI layers for the reasons mentioned above for vectors.

6 The kernel layers

The kernel layer internal structures will be altered to avoid caching DM object values a second time. The internal structures will be limited to containing kernel-specific information.

To aid in copying objects, the different kernels will share a common handle type. Some PKI routines will accept a dmKernelData handle containing kernel-specific data for an object. The kernel routines will test the common kernel_id member of this structure, and if it's the right kernel, the data will be copied, otherwise it will be ignored. For example, FITS ASCII and binary tables both map to a DM table. In the new design, the block will have a kernel-specific pointer that says "I was originally a FITS ASCII Table". If it's copied to a QPOE, that information will be ignored, but if it's copied to another FITS file, it will be picked up and (unless overridden by a specific directive) the resulting FITS file will be made an ASCII table instead of the default binary kind. This provides a mechanism for making sure that a copy of a FITS file to a FITS file preserving as much of the original file's idiosyncrasies as possible, while retaining the capability to copy it to a QPOE file in a clean way.

7 The PKI

Here are the existing Kertable routines and their PKI analogs. The names 'Object' and 'Property' will be changed to 'Block' and 'Key' to make the DM nomenclature consistent.

Data-type families will be made type-generic, so that

```
kPutProperty_double( block, name, double val, status )
```

becomes

```
kPutKey( block, name, dmDOUBLE, void* val, status ).
```

This reduces the number of separate PKI routines.

Some descriptor-specific routines will be made descriptor-generic, so that kSetColumnUnits becomes kSetDescriptorUnit and works to set the units for keys and columns and coords.

This basically moves a switch statement into the kernels, but decreases the number of kernel routines; this decision may be revisited.

Some routines that access by name will be changed to access by number or handle, so that renaming the object or (in some cases) multiple objects with the same name will not cause problems. Attention will be given to make sure that this doesn't cause new problems when reordering objects (which is not well supported in the current design anyway): careful distinction will be made between raw object order and virtual-file object order.

etBool (*kAccessDataset) (char *dsname);	TBD
etBool (*kCanCreateDataset)(char* dsname);	TBD
void (*kCreateDatasetFunc) (char *dsname, kdsHandle *kds, etStatus *status);	Same?
void (*kDeleteDatasetFunc) (char *dsname, etStatus *status);	Same?
void (*kRenameDatasetFunc) (char *dsname, char *newdsname, etStatus *status);	Same?
void (*kOpenDataset) (char *dsname, etBool update, kdsHandle *kds, etStatus *status);	Same?
void (*kCloseDatasetFunc) (kdsHandle kds, etStatus *status);	Same?
void (*kFlushDatasetFunc) (kdsHandle kds, etStatus *status);	Same?
etBool (*kAccessObjectFunc) (kdsHandle kds, char *objname);	Not needed
void (*kCreateObjectFunc) (kdsHandle kds, char *objname, edsObjType objtype, kobjHandle *kobj, etStatus *status);	Same
void (*kGetObjectNamesFunc) (kdsHandle kds, char ***objnames, long *nobject, etStatus *status);	Replace with GetNoBlocks and GetBlockNameByNo?
void (*kGetOpenObjectName) (kobjHandle kobj,char **objname,etStatus *status);	Change to access by No
void (*kGetObjectTypeFunc) (kdsHandle kds, char *objname, edsObjType *objtype, etStatus *status);	Same
void (*kDeleteObjectFunc) (kdsHandle kds, char *objname, etStatus *status);	Change to access by No
void (*kRenameObjectFunc) (kdsHandle kds, char *objname, char *newobjname, etStatus *status);	Change to access by No
edsObjType (*kOpenObjectAs) (kdsHandle kds, char *objname, edsObjType eObject, kobjHandle *kobj, etStatus *status);	TBD
void (*kCloseObjectFunc) (kobjHandle kobj, etStatus *status);	Same

void (*kFlushObjectFunc) (kobjHandle kobj, etStatus *status);	Same
etBool (*kAccessProperty) (kobjHandle kobjh, char*);	TBD
void (*kDeleteProperty) (kobjHandle kobjh, char*, etStatus*);	TBD
void (*kPutProperty<type>) (kobjHandle kobjh, char* propname, <type> val, etStatus* status);	Add desc and unit and comment arguments
	Make type-generic
void (*kGetProperty<type>) (kobjHandle kobjh, char* propname, <type>* val, etStatus* status);	Add extra info, make type-generic
void (*kPutPropertyComment) (kobjHandle kobjh, char* propname, char* comm, etStatus* status);	Make descriptor-generic
void (*kGetPropertyComment) (kobjHandle kobjh, char* propname, char** comm, etStatus* status);	Make descriptor-generic
void (*kGetPropertyNames) (kobjHandle kobjh, char*** propnames, long* numprop, etStatus* status);	Replace with access by number
void (*kGetPropertyType) (kobjHandle kobjh, char* propname, etDataType* proptype, etStatus* status);	Access by handle
void (*kGetNewKWIndex) (kobjHandle kobj, char* strIndexKey, long* n, etStatus* pStatus);	Deleted, done within kernel
etBool (*kGetKWIndex) (kobjHandle kobj, char* strKWName, char* strIndexKey, long* n, etStatus* pStatus);	Deleted, done within kernel
char* (*kET2KernelDatatype) (etDataType etType, etStatus* pStatus);	Deleted, work in DM types
etDataType (*kKernel2ETDatatype) (char* strKernelDatatype, etStatus* pStatus);	Deleted
void (*kTotalTableRows) (kobjHandle kobjh, long* numrows, etStatus* status);	Same
void (*kTotalTableColumns) (kobjHandle kobjh, long* numrows, etStatus* status);	Same
etBool (*kAccessColumn) (kobjHandle kobjh, char* colname);	Not needed

void (*kCreateColumn) (kobjHandle table,char *colname,etDataType datatype, long length, long string_length, long* axlen, long naxes, char *units, char *format, int var, char* desc, kcolHandle *col, etStatus *status);	Same?
void (*kDeleteColumn) (kobjHandle kobj, char* colname, etStatus *status);	Access by handle
void (*kRenameColumn)(kobjHandle table, char* colname, char* newcolname, etStatus* status);	Access by handle
void (*kOpenColumn) (kobjHandle table, char *colname, kcolHandle *col, etStatus *status);	Access by handle, name table held at DMP layer
void (*kOpenColumnByPosition) (kobjHandle table, long colpos, kcolHandle *col, etStatus *status);	Combined with routine above
void (*kGetOpenColumnName) (kcolHandle col, char** str, etStatus* status);	Descriptor-generic
void (*kGetColumnType) (kobjHandle table, char *colname, etDataType *coltype, etStatus *status);	Descriptor-generic
void (*kGetColumnLength) (kobjHandle table, char *colname, long *length, long** axlen, long* naxes, etStatus *status);	Descriptor-generic
void (*kGetColumnStringLength) (kobjHandle table, char *colname, long *string_length, etStatus *status);	Descriptor-generic
void (*kGetColumnUnits) (kobjHandle table, char *colname, char **units, etStatus *status);	Descriptor-generic
void (*kGetColumnFormat) (kobjHandle table, char *colname, char **format, etStatus *status);	Descriptor-generic
void (*kGetColumnNames) (kobjHandle table, char*** names, long* num, etStatus* status);	Not needed
void (*kSetColumnUnits)(kobjHandle table, char* colname, char* format, etStatus* status);	Descriptor-generic
void (*kSetColumnFormat)(kobjHandle table, char* colname, char* format, etStatus* status);	Descriptor-generic
void (*kGetRows)(long rownum, kcolHandle* colhandles, long numcols, long buflen, kcolBuffer colbuffer, long* nread, int invert, etStatus* status);	To be redesigned

void (*kPutRows)(long rownum, kcolHandle* colhandles, long numcols, long buflen, kcolBuffer colbuffer, long nwrite, etStatus* status);	To be redesigned
void (*kGetAxLen)(kobjHandle, long, long*, etStatus*);	Descriptor-generic
void (*kGetNDim)(kobjHandle, long*, etStatus*);	Descriptor-generic
void (*kGetImageType)(kobjHandle, etDataType*, etStatus*);	Descriptor-generic
void (*kGetSection)(kobjHandle, long*, long*, long, void*, int,etStatus*);	Same?
void (*kSetAxLen)(kobjHandle, long, long, etStatus*);	Descriptor-generic
void (*kSetNDim)(kobjHandle, long, etStatus*);	Descriptor-generic
void (*kSetImageType)(kobjHandle, etDataType, etStatus*);	TBD
void (*kPutSection)(kobjHandle, long*, long*, long, void*, etStatus*);	Same?
void (*kAddIntervalToFS)(kobjHandle table, etDataType datatype, void *starts, void *ends, long numintervals, char* intname, char *colname, long cpt, etStatus* status);	Delete; filter at DM layer
void (*kMakeTabFilter)(kobjHandle objh, char* tabname, char* name, char** colNames, void* mins, void* maxes, etDataType dtype, char* unit, long nvalues, etStatus* pStatus);	Redesign as generic DSS writer
void (*kGetTabFilter)(kobjHandle objh, char* tabname, char** actual, char** name, char** colNames, void** mins, void** maxes, etDataType dtype, char** unit, long* nvalues, etStatus* pStatus);	Redesign as generic DSS reader
void (*kAddBinSpec)(kobjHandle objh, char *axisName, double min,double max,double step,etStatus* status);	Delete; bin at DM layer
void (*kPutWCS)(kobjHandle kobjh, char* name, char* ttype, char** cptNames, long* colNums, char* unit, etDataType dType, void* crpix, double* crvals, double* cdelt, long dim, double* params, long nparams, char system, etStatus* pStatus);	Same?

void (*kGetWCSNames)(kobjHandle kobjh, char*** ppstrWCSNames, long* numCoords, etStatus* pStatus);	Replace with generic WCS handle reader
void (*kGetWCSInfo)(kobjHandle kobjh, char* name, char** type, char** unit, char*** cptNames, char* system, long** colNums, etDataType* dtype, void** crpix, double** crvals, double** cdelt, long* dim, double** params, long* nparams, etStatus* pStatus);	Redesign and combine with above
void (*kSetColumnRange)(kobjHandle objh, char* name, void* vmin, void* vmax, etStatus* pStatus);	Descriptor-generic
void (*kGetColumnRange)(kobjHandle objh, char* name, void* vmin, void* vmax, etStatus* pStatus);	Descriptor-generic
void (*kSetColumnBin)(kobjHandle objh, char* name, void* vmin, etStatus* pStatus);	Same?
void (*kGetColumnBin)(kobjHandle objh, char* name, void* vmin, etStatus* pStatus);	Same?
void (*kSetColumnNull)(kobjHandle objh, char* name, void* vmin, etStatus* pStatus);	Descriptor-generic?
void (*kGetColumnNull)(kobjHandle objh, char* name, void* vmin, etStatus* pStatus);	Descriptor-generic?
void (*kSetColumnDesc)(kobjHandle objh, char* name, char* desc, etStatus* pStatus);	Descriptor-generic
void (*kGetColumnDesc)(kobjHandle objh, char* name, char* desc, long maxlen, etStatus* pStatus);	Descriptor-generic
void (*kSetHints)(char* name, char* value);	Same
void (*kAddRegionFilter)(kobjHandle objh, char** cptnames, void* region, long cpt, etStatus* pStatus);	Delete, filter at DM layer
void (*kSetTablePref)(kobjHandle objh, char** names, long n, etStatus * status);	Same
void (*kGetTablePref)(kobjHandle objh, char*** names, long* n, etStatus * status);	Same
long (*kGetNoKeys)(kobjHandle objh, etStatus* status);	Same
void (*kKeyPrint)(kobjHandle objh, long keyno, char* buf, long maxlen, etStatus* status);	Same?
int (*kNullPrimary) (kdsHandle ds);	TBD

void (*kUpdateFilter)(kobjHandle block, char* buf);	TBD
void (*kGetComment)(kobjHandle objh, char* name, long no, char** tag,char** comment, etStatus* status);	Same
void (*kSetArraySize)(kcolHandle col, long nvals);	Same?

8 Work packages

Here I describe the components of the DM and the extent of the proposed work. The IRAF-QPOE kernel is not addressed here; to handle it, the changes would be pretty mechanical after doing the FITS kernel. The ASCII kernel could be written in parallel with the FITS kernel rewrite, or left till later.

The main pieces of work are:

- (A) remove duplication of dm/kernel cached info
- (B) header key parsing and composing
- (C) header key caching
- (D) table filtering
- (E) image axes
- (F) image filtering

(A) Duplication of dataset, block and descriptor info: these structures exist at both DM and kernel layers. The kernel layer structres have some kernel-specific info but also duplicate much of the info from the DM layer. I will move the DM structures to the DMI layer at the bottom, and having both DM and kernel layers access them - either directly or (at some efficiency cost) via wrapper routines. This will eliminate a lot of code that's required to keep them in sync, and make it easier to write new kernels. The removal of incompatible terminology (properties for keys, objects for blocks) will make the code more maintainable, too.

A typical change to be made is the method of closing a block. Currently, the dmBlockClose routine calls a dmpBlockClose routine which does work at the DM layer and also calls a kernel->BlockClose routine to do kernel-specific work. In the new design, dmBlockClose will call kernel->BlockClose which will in turn call dmpBlockClose, now at the DMI layer. The argument to the kernel routine was a dmkBlock (kernel block) and is now a full dmBlock; the dmkBlock structure is simplified because it no longer needs to duplicate and stay in sync with the dmBlock structure; and a few lines of code in the kernel routine are changed to distinguish between dmkBlock and dmBlock references. Most of the work is in the repackaging and moving around, rather than in lines of code actually changed.

(B) Header parsing: There is a lot of nasty code in dmbasis.c. The problem, as ever, is that ETOOLS picked apart the compound information at the kernel layer - e.g. DTYPE/DVAL pairs, making it hard to put them back together. This is even worse at key write time, where the convention of DTYPE/DVAL should be at the kernel layer and invisible at the DM layer. Much grief to keep this working in the current design, and it stops me fixing the header caching problem below. The new design lets the DM work in terms of platonic ideal DM keys, and the fact that FITS may use multiple keywords to store them is entirely hidden at the DM layer as it should be.

(C) Header caching: The problem with editing keywords is that information about keywords is kept both at the DM and the kernel layer, but changes in keyword order are not propagated. We do want to have a separate kernel header cache in FITS, because of the way the different keywords interact with one another. So we need to update both the DM and FITS order at the same time. This requires changes to the kernel interface routine for header keys, which are already really broken because of the header parsing issues described above.

(D) Table filtering: When filtering, the current design passes a request for one row down to the kernel; the kernel reads one row and filters it, and passes the result back up to the DM, filling another row in the row buffer. The buffer is then memcpy'd to the output. Instead, I propose to fill the DM row buffer directly with data, do the filtering there and mark each row as good or bad, and then memcpy the good data row by row to the output. This will reduce the number of calls to CFITSIO, and eliminate a whole set of kernel filter infrastructure that duplicates the DM layer filters. Testing of these changes (see below) may be split up - first put in the infrastructure and test that you can still get data in and out without filtering. Then debug the filtering and test that. Then put back in the hooks to subspaces and coordinates, and test that. At that point, you have really tested all of the new parts A-D. The image stuff (E,F) is just as important, but is a more (not entirely!) separable problem.

(E) Image axes: this is bound up with header parsing. The worst problem is the coordinate systems. The ETOOLS legacy is that the information about the individual axes and their coordinate systems is broken apart and passed up to the DM layer as lists of names, which then have to be matched up again - at the FITS level we have axis numbers to let us do this more easily and robustly. The code that stitches together information about the different axes is necessarily complicated anyway, as it must support logical, physical and world coordinate systems, and the presence or absence of them on any particular axis, and spotting that the presence of an RA/Dec world system implies that the physical system must be sky coordinates even if there aren't any keywords to say so, and other special cases. However, the situation is made even harder in the current configuration and the current code is one of the most confusing and least maintainable parts of the DM. Further, it has proved impossible to reliably support image filtering on logical and physical coordinates. The rewrite will allow me to provide this support robustly and leave behind a still arcane but somewhat more maintainable algorithm.

(F) Image filtering: Once the new support for image axes is in place, I can make image filtering work properly. This requires minor changes to the filtering code, and the move of the filtering code to the DM layer (as for tables).

I propose to develop the new DM in a phased manner. However, the new DM will not have the capabilities of the old DM until the final phases. Because of the interaction of different components of the old design, it would be a lot of wasted work to keep the revised DM fully functional at every step along the way - it's better to have some things temporarily broken. Of course, while this is happening, the old DM should remain the version seen by ciaox! At each phase, specific features of the DM will be tested.

The proposed test/putback phases are:

	Theme	Test Capability
Phase A	Blocks	List blocks
Phase B/C	Keys	List header
Phase D1	Table Read	List table data
Phase D2	Table Write	Copy table (no filter)
Phase D3	Filter	Filter table
Phase D4	Subspace/Coords	Propagate subspace; check coords
Phase E	Images	Image data and coords
Phase F	Binning	Bin table to image, filter image
Phase G1	Complete	Cleanup, dmcopy/dmlist regression pass
Phase G2	Full Debug	Test against CIAO tools, debug

Here is a summary of the code components of the DM, and how major the changes will be. Of course this is a bit misleading, since even for the pieces that say "80% rewrite" much of the work will be rearranging blocks of code, or mechanically going through and replacing a pointer to one structure with a pointer to another structure (e.g. kernel block to DM block). Segments with significant rewrite are called out with an asterisk.

Component	Subdirectory	LOC now	% rewrite	Notes
DM Parser	filter	2000	10	Minor cleanup
DM Subspace	filter	9000	10	Minor cleanup
DM Coords	coords	2000	10	Cleanup, Sync with kernel
DM Image axes	block	1000	80*	(A) Major simplification after new kernel
DM Images	descriptor	500	20	Cleanup
DM Blocks	block	5000	30*	(A) Sync with rest of rewrite
DM Buffer	misc	2000	80*	(D) Rewrite table filtering
DM Keys	descriptor	5000	80*	(B) Remove header parsing to kernel
DM Set/Get	descriptor	5000	10	Cleanup
DM Descrips.	descriptor	2000	50*	(C) Add header key stuff from kernel
DM Dataset	dataset	600	20	Cleanup, kernel layer change
Kertable	kertable	500	50*	(A) PKI routine API

Ker Subsp.	ftfilter	2300	80*	(D) Move filtering to DM layer
Ker Coords	ftcoords	1700	30*	(B,E) New header parsing
Ker Images	ftimages	2200	25(*)	(F) Move filtering to DM layer
Ker Blocks	ftsymtab	1000	70*	(A) Rewrite - simplify
Ker Blocks	ftobject	1000	70*	(A) Rewrite - simplify
Ker Buffer	fttables	1500	70*	(D) Move filtering to DM layer
Ker Keys	ftproplist	4200	40*	(B,C) Rework to do more parsing at kernel
Ker Tables	fttables	4000	40*	(A,D) Don't cache info at kernel layer
Ker Dataset	ftdataset	500	20	Cleanup, kernel layer change
Utils	edsutil	5000	40	Cleanup and additions