Revised Coordinate Systems Design for ASC Data Model SDS-12.0

Jonathan McDowell

March 29, 1998

Contents

1 Vector columns

Experience now suggests a slight modification to the use of vector columns as described in the abstract design. For maximum ease of use with existing code, I propose a new function

dmDescriptor* dmGetCpt(dmDescriptor* v, long* cptNo);

and a new implementation suggestion: Creating a scalar column involves making the descriptor, tying it to an ETOOLS column, and entering it in the DM's list of columns (accessed by TableGetNoCols and TableOpenColumnNo). I propose that creating a vector column of dimension n should make the n column descriptors, tie them to the ETOOLS columns, enter them in the name index (for dmTableOpenColumn) but not in the list of columns. It should then make the vector descriptor, which would have pointers to the individual scalar descriptors and would be entered in the list of columns. It would also make the MTYPE and MFORM keywords. So, TableOpenColumnNo would see only the vector column, not its components, but TableOpenColumn would be able to find the component by its name and use it for scalar column operations. The new routine dmGetCpt would give you access to the components without knowing their names, by returning the scalar descriptor for a given component number of a vector descriptor.

However, coordinate systems could attach directly to the vector column descriptor, not to the individual column descriptors.

When binning on a vector column, the resulting image should have an axis group of dimension the same as the vector column (see below).

When reading a column, in many cases of old files we want to interpret a pair of columns as a vector column but it has no grouped name. In this case, or in a case where dmColumnCreateVector has a blank name entry, we make a spurious name by combining the component names in a parenthesized comma-delimited list: e.g. component names RA, Dec go to vector column name '(RA,Dec)'.

With this implementation we need to:

- implement dmColumnCreateVector
- implement dmGetCpt
- Fix dmGetScalar to either fail, or use the first vector component, if given a vector descriptor
- Delay implementation of dmGet/SetVector(s) since we can still use dmGet/SetScalar(s) on the individual components
- Adapt dmBlockOpen to scan for MTYPE and MFORM keywords and create vector columns accordingly (but see also coord issue below, of making RA-TAN/DEC-TAN a vector col automatically)
- Adapt binning to handle case of binning on a vector column.

- Issue: can we let the user make new groupings with a dmDescriptor* dmVectorGroup(dmDescriptor** list, long dim, char* name) ? This would have an error if any list entries were vectors.

2 Images and Arrays

The definition of axis groups in the abstract design needs to be modified slightly. My new view is presented here. Types of descriptor are:

- Data Descriptor (Column Descriptor, Key Descriptor, Image Data Descriptor)
- Subspace Column Descriptor
- Coord Descriptor

A descriptor has:

- the Common Descriptor Properties: name, unit, description, data type, display format
- the Element Definition Properties: element dimensionality d, element type t
- the Array Specification Properties: array dimensionality n, array dimensions n_i.
- the Attached Coordinate List, a set of pointers to Coord Descriptors
- the Axis Groups (described below), present only if $n > 0$.
- the Auxiliary Data; a pointer to actual data for Data Descriptors or Subspace Column Descriptors, or to the transform info for Coord descriptors.

The transform info for Coord descriptors includes a pointer to the 'parent' descriptor as follows:

- Transform type
- Transform values (CRPIX, CRVAL, CDELT)
- Transform parameters
- Parent descriptor
- Parent descriptor axis group number (zero if the coord is attached to the descriptor itself rather than to one of its axes).

With the Axis Groups, we do two things: (1) group the axes in the same way that vector columns group table columns; (2) attach the logical to physical coordinate transformations which rescale the binned image pixels to 'original' pixels, and provide names for the axes. These axis groups are (for the time being) each of dimension 1 or 2. They correspond to the **logical coordinate** system in IRAF WCS language.

The Axis Groups information consists of

- The number of groups, ng
- The number of groups actively defined so far, ng1
- The dimensionality of each group, g_i (must be less than or equal to 2 for the current implementation)
- For each group, a (possibly null) pointer w_i to a Coord Descriptor of transform type linear.

Initially ng = n (the number of axes), $g_i = 1$ for all i, w_i is null, and ng1 = 0. At all times the sum $\sum_{i=1}^{ng} g_i = n$. Calls to dmCreateAxisGroup (see below) should increment ng1 and set w_{ng1}; if the call has dim=2 then n_g is decremented and g_{ng1} is incremented. An alternate acceptable implementation would be to set ng = 0 initially and only create axis groups when the explicit call is made; this makes the inquiry routines to get at axis group info a little less friendly, though.

When can a Coord Descriptor be the parent of another Coord Descriptor? We don't want to chain coord descriptors usually, for efficiency reasons and FITS compatibility reasons. However, it makes sense for the world coordinates to be chained to the physical coordinates. Hence we make the following rule for dmCoordCreate: if the parent descriptor is of type dmCOORD and the parent descriptor's own parent descriptor axis number is zero, dmCoordCreate should return an error to the effect of 'Can't chain coordinates'.

In FITS and IRAF, the axis group dimensions for those axis groups of dim 2 or more must be stored as header keys. The keyword AXISD4 = 3 indicates that the 4th axis group has dimension 3. AXISDn keywords with value 1 should be omitted.

In general, the default component names should be AXISi and the default full names should be AXISi if the axis group has dim 1, AXISGj otherwise (where j is the axis group number). These default names are not written to the data file. If non-default names are used, they should be stored as AXISGn = '....' for the group name and AXISn = '...' for the component name.

- *Implement axis group descriptors*
- *Implement dmImageCreateGroups, dmArrayCreateAxisGroup*
- *Check the usual dmGetUnit, dmSetDesc, etc. work for axis group descriptors.*
- *Adapt dmSet/GetScalar to fail for axis group descriptors*
- *Issue: should the dmImageCreate, dmImageCreateAxes, dmImageCreateGroups calls have dmBlockImageCreate, etc counterparts with dmDataset* arguments?.*

3 API revisions

3.1 Coord Create routines

We need to modify dmCoordCreate_T to add an extra argument char** cptNames between the arguments unit and dim.

We need to add

dmCoordCreateLinear_T(dmDescriptor* dd, char* name, char* unit, T crpix, double crval, double cdelt);

This wraps the simple, common case of a linear coord transform of dimension 1 so you can use literal constants instead of having to create a variable with pointers, e.g.

```
dmCoordCreateLinear_l( pha, "Energy", "keV", 0, 0.15, 0.01 );
```

We need to modify dmCoordCreateI to only support 1D:

dmCoordCreateInteger_T(dmDescriptor* dd, char* name, char* unit, Type crpix, long crval, long cdelt);

If you need to make an integer coord descriptor for a vector column, with an n-dimensional linear transform, you can call dmCoordCreateInteger_T on all of the individual components. You'd then like to be able to group the resulting descriptors in a single vector coord descriptor using something like dmVectorGroup described above.

To create an image physical system, we have

dmDescriptor* dmArrayCreateAxisGroup(dmDescriptor* dd, char* name, dmDataType type, char* unit, char** cptnames, integer dim)

(R1.7) Creates an axis group of dimension dim on the next available set of axes. Returns an error if we run out of axes. Creates and returns a dmCOORD descriptor whose parent is the just created axis group, with the given name and component names and element dimensionality dim. Initializes the transform to the unit transform. The transform can then be reset with dmCoordSetTransform_l. This version works on descriptors of type Image Data.

(R3.0) Support for table cell images.

dmDescriptor* dmArrayCreateAxisGroups(dmDescriptor* dd, char** name, dmDataType* type, char** unit, char** cptnames, long* dim, long ngroups)

(R1.7) Like the above, but lets you define multiple groups at once.

Note: dmArrayCreateAxis is deleted from the API, as are dmArrayGetAxisGroupCoord and dmArrayGetPixelCoord, dmImageCreateAxes, dmImageCreateGroups.

long dmArrayGetNoGroups(dmDescriptor* dd)

(R1.7) Returns the number of axis groups.

3.2 CoordCalc

int dmCoordInverse_T(dmDescriptor* dd, double* value, T* inverse)

(R1.8) Return the inverse coord transform given a dmCoord descriptor and a value.

3.3 Unifying Image/Table creation semantics

Currently dmDatasetTableCreate maps to dmImageCreate, while dmBlockCreate is useless for non-null images. We'll revise the API so dmDatasetTableCreate is renamed dmTableCreate, and dmBlockCreate for tables is replaced by a new dmDatasetCreateTable and dmDatasetCreateImage. This has the advantage that the dmDatasetCreate routines have a dmDataset* argument like the other routines beginning with dmDataset in their name. We should also rename dmDatasetTableOpen/dmDatasetTableClose to dmTableOpen and dmTableClose.

dmBlock* dmDatasetCreateImage(dmDataset* ds, char* name, long* axes, long naxes)

(R1.7) Create image within dataset.

dmBlock* dmDatasetCreateTable(dmDataset* ds, char* name)

(R1.7) Create empty table within dataset.

4 Physical system

The IRAF WCS model treats the special case of the **physical coordinate system**, which is a linear 1-D mapping from a logical coordinate axis.

In IRAF-based FITS files the mapping from logical to physical is done with LTVi and LTMi_j keywords.

In the ASCDM, we introduce a coordinate transform descriptor, whose parent is the axis group.

Rule: A coordinate descriptor whose parent is an axis group must be of transform type linear or nD-linear.

The physical system represents 'original' binning of the data, say the original instrument pixel size.

Rule: On rebinning an image, if no physical system exists, one is created equal to the logical system of the original image. On reading an image, if there is one coordinate system and it is linear, it is interpreted as the physical system.

The physical system is indicated to the DM by the special transform name 'physical', which is otherwise the same as 'linear'.

4.1 WCS systems

The abstract design indicates that the world coordinate systems should be mappings from physical to world, not logical to world. Remarkably, this turns out to be how IRAF handles things internally too. However, externally files record the logical to world system. Therefore, on reading a header one must recreate the physical-to-world system from the logical-to-world plus logical-to-physical.

An alternate implementation is to make all the coordinate systems have the axis group as their parent, in other words all are logical-to-world, matching the header info (the FITS approach). I think we should probably go with the IRAF approach.

For the first linear transform, on FITS axis n,

- CTYPEn gives the name of the coordinate
- CUNITn gives the unit of the coordinate
- CRPIXn, CDELTn, CRVALn give the crpix, cdelt, cval values defining the linear transformation from logical to world coordinates.

For subsequent linear transforms, use CiTYPn, CiUNIn, CiRPXn, CiDLTn, CiRVLn where i is 1, 2, ... m-1 for m subsequent transforms.

The index i = 1 is reserved for the physical transform. The LTM and LTV keywords should also be read and written for back compatibility (eg saoimage).

For a 2D transform on a vector column: suppose the FITS columns of the vector column are nos. 3 and 4. Then we write CUNIT3, CUNIT4, CRPIX3, CRPIX4, CDELT3, CDELT4, CRVAL3, CRVAL4 as before (or CiUNI3/CiUNI4 etc as appropriate), but the CTYPEs are more complicated - you have to follow the WCS rules explained in the DM FITS Conventions memo. You also need to write the MTYPE and MFORM keywords for the names.

4.2 WCS systems on columns

WCS systems on columns are handled the same as WCS systems on axes. The FITS keywords are different (see the FITS conventions document).

For columns, the physical transform is considered to be the identity.

When binning a set of columns to an image:

- Scalar columns map to scalar axes; vector columns map to axis groups.
- A coord system on a column maps to a coord system on the corresponding axis, corrected for binning as necessary.
- Unless the image has binning by factor 1 and lower left corner position equal to (1,1,...) in the units of the table columns, create a physical coordinate transform which relates the new logical system (the image system) to the original table values. If there is no WCS, make the physical system the WCS. For instance,

  ```
  dmcopy foo.fits[bin time=100:200:10,pha=20:40:2] bar
  ```

  would make a 10 x 10 image with two axes, with coordinate system

  ```
  CTYPE1 = TIME
  CTYPE2 = PHA
  CDELT1 = 10.0
  CRPIX1 = 0.5
  ```

```
CRVAL1 = 100.0
CDELT2 =  2
CRPIX2 =  0.5
CRVAL2 = 20.0
```

Here we assume no WCS previously on TIME or PHA; we put the physical system in CTYPE etc.

On the other hand

```
dmcopy foo.fits[bin pos=box(100 200 20 40):2] bar
```

with coord keywords

```
MTYPE1 = 'POS'
MFORM1 = 'X,Y'
MTYPE2 = 'EQPOS'
MFORM2 = 'RA,DEC'
TTYPE5 = 'X'
TTYPE6 = 'Y'
TCTYP5 = 'RA---TAN'
TCTYP6 = 'DEC--TAN'
TCDLT5 =    -0.04
TCDLT6 =     0.04
TCRPX5 =  512.0
TCRPX6 =  512.0
TCRVL5 =  101.3
TCRVL6 =  -20.3
```

would make

```
MTYPE1 = 'POS'
MFORM1 = 'X,Y'
MTYPE2 = 'EQPOS'
MFORM2 = 'RA,DEC'
AXISD1 =    2
CTYPE1 = 'RA---TAN'
CTYPE2 = 'DEC--TAN'
CDELT1 =     -0.08
CDELT2 =      0.08
CRPIX1 =   256.0
```

```
CRPIX2 =   256.0
CRVAL1 =   101.3
CRVAL2 =   -20.3
C1TYP1 = 'X'
C1TYP2 = 'Y'
C1DLT1 = 2.0     / Physical system
C1RPX1 = 0.5
C1RVL1 = 100.0
C1DLT2 =  2.0
C1RPX2 =  0.5
C1RVL2 = 20.0
```

with the physical system in C1TYP.