# ASC Data Model Abstract Design: SDS-3.0

Jonathan McDowell

February 26, 1998

## Contents

# 1 Introduction

The Science Data Systems Group at the AXAF Science Center (ASC) has been studying the problems and limitations of current astronomy data analysis systems. The result of that study is a proposed generic Data Model for astronomical data. The ASC Data Model describes the common structure for the data to be analysed by our interactive analysis tools. The same structure should also be used for pipeline processing. In this document I present the proposed ASC Data Model from the science requirements point of view. This first section describes some aspects of the model, which is presented in detail in later sections.

## 1.1 Overview of the Model

Our data model has a number of high level goals:

- Create data files which are more fully self-describing, while retaining back compatibility in the sense that existing archival FITS files will be interpreted correctly.
- Systematize the treatment of filtering, uncertainties, units, and coordinate systems, unifying the current approach which involves a large number of special cases.
- Allow programs to use both FITS and native IRAF file formats interchangeably, by supplying a format-independent interface layer.
- Allow users to write their own programs easily by providing a subroutine interface which makes accessing the data easy and removes the need for the user to worry about the details of the file format.
- Support advanced virtual file and filtering operations by providing a uniform convention for recording the way a file has been filtered.

The intent of the model is to describe an abstract representation of a generic astronomical dataset and to layer extra structure onto existing file formats to make them more fully self-defining. Our datasets include both binned (image) and tabular data, corresponding to the IRAF IMH and QPOE formats or the FITS IMAGE and BINTABLE formats. In this document I will make explicit parallels to the FITS format since it is the externally defined export and exchange format. An important aspect of the design presented here is that existing FITS files will be interpreted correctly by the data model. This is achieved through careful use of default values for keywords in the mapping to FITS.

## 1.2 What is a Data Model?

A data model is an abstract description of our datasets. (Datasets may be files, or groups of files that we want to consider as a unit). It tells us the different properties and attributes a dataset can have (e.g. 'a dataset consists of a header and a table or an array; a table has n columns each with a name, a data type, a unit, ' ... etc.) This description of the data is possible because all of our many different datasets can be thought of as special cases of a very small number of basic types of dataset. In using the data model to describe a dataset, then, we have a way of defining that dataset which makes explicit its differences from all other datasets. Furthermore, the data model contains no information about the storage format of the data. Thus our definition of the structure of a dataset is completely separated from the way in which that structure is implemented on disk - we distinguish between information that is truly part of the scientific data and information that is bookkeeping or specific to the file format. This makes it easy to support multiple file formats with the same data model. The data model can be implemented as an API which lets you access and manipulate the data using the concepts of the data model.

The data model gives the application writer an interface to the data which is independent of the details of the file format. It also provides a standardized structure and language which brings out the similarities between different kinds of dataset. This standardization is an important advance beyond the standardization provided by particular data formats such as FITS.

- We make the treatment of the data independent of the choice of disk file data format, thus allowing the algorithm to concentrate on the science and making it easy to support the open architecture of different data formats. It means that applications writers don't have to worry about the specifics of the data format, those are hidden in the interface subroutines.

- The model layers extra structure onto the concepts implicit in the underlying data formats.

- We describe all data in a common structure; by imposing a uniform description we can support generic tools. We have a way of describing a general data file independent of the specific structure of the file (PHA file, event file, etc). This means that when you make a new kind of file, existing tools can still do something with it. FTOOLS does this at a certain level, allowing basic filtering of generic tables, and can be thought of as having a very simple data model consisting of table columns with no extra attributes. Our software will go well beyond this, dealing with coordinate systems and other auxiliary quantities in a standardized way.

- Further, all this makes the data more self-describing.

- We explicitly tie information relating to each image axis or table column to that axis or column. In FITS, there is some of this: a keyword like CRVAL4 tells you the coordinate value for axis number 4. However, there are a lot of other keywords that don't do this and could - for instance, TSTART gives the start time for a dataset, but there is no explicit expression of the fact that this quantity is related to the TIME column in the data.

- Note that a single data model table may correspond to many FITS tables. For instance, the Good Time Intervals, which in the data model are just the ranges for one axis of the data subspace, have to be kept in a separate table in the FITS file. At the moment FITS files often have an assortment of tables in them, some of which are related to each other and some of which aren't. Using the data model will help us make much more sensible decisions about which FITS tables to group together in a single file. For instance, for an EVENT file it helps us realize that the Good Time Intervals are truly just an auxiliary piece of information describing the main table, while ROSAT Temporal Status Intervals are (at least on the data model I present here) a separate data object that has meaning separately from the EVENT data.

- The concept of a data subspace lets us unify the treatment of good time intervals, spatial regions, and filter ranges. This makes these concepts independent of whether a particular column contains temporal, spatial or spectral info, and lets us be much more systematic about asking the question 'to what range of data values does this dataset apply?'.

- Grouping together of header keywords helps us propagate related info more easily, makes it easier to specify the definition of new files in terms of old ones, and improves user readability of headers.

- The existence of a data model helps us include support for new features (e.g. uncertainties) in a systematic way, so we don't have to deal with hundreds of special cases each time. This applies both to the new features we add now and to future features in later versions of the model - in other words, having an overall data model reduces overhead in including new functionality, because it's clear how to add that new functionality in a way that will work throughout the system.

- The separates out the science description from the details of a data format, allowing us to define clean mappings to different data formats. This makes it easier to support new data formats, since the I/O is so well isolated.

How can we be all things to all systems? The crucial idea is the concept of a data model. By this we don't mean a model of a specific dataset, like a spectral fitting functional model, we mean a model of the *concept* of astronomical data. More specifically, we mean an abstract description of the structure of our data separate from its implementation in a particular disk storage format. We note for the software-literate that this abstract description can be - but does not need to be - given a manifest software implementation as an object or set of objects in an object-oriented language. Once we have our data model, we can map it to the particular disk data formats we wish to support. This allows the same code to read FTOOLS FITS files or PROS QPOE files and 'see' them (after a translation layer) as identical sources of information. The individual tool will not usually need any explicit 'if FITS then' code, and will not even know what type of file is being read.

In principle such a data model could be arbitrarily complex with many special cases. Actually it turns out that almost all our data can be described by a single kind of object, perhaps with a few simple flavors. This fact is what gives FTOOLS its strength: much X-ray data analysis can be accomplished by fairly general manipulations of FITS binary tables. We take FTOOLS' advance one step farther by separating our unified data description from the specifics of the FITS format (2880-byte blocks, indexed keywords, storing the structure of the main data as header keywords, no units on keywords, etc), which are not relevant to any of the science algorithms. This separation turns out to be extremely powerful, and allows us to do a lot more than just support multiple data analysis contexts.

The existing package tool kits (FDUMP, TPRINT, etc..) will work on our files but may lose the extra layers of meaning provided by our data model. We will therefore provide a new set of infrastructure tools which will do generic operations on our files. This IS a significant amount of extra work, but we believe our plan builds on existing infrastructure experience in important ways. For instance, we wish to unify and extend the PROS concepts of filters, regions and good time intervals into a single selector concept; this will greatly increase the flexibility of filtering.

## 1.3 Summary of motivation

By generalizing our approach, we can get by with fewer distinct tools. By writing the tools using our data model, and modern software approaches (careful layering, self-describing data, etc.), we can make each tool more flexible, able to do sensible things with data that is in slightly different formats, or even data representing entirely different physical quantities. We try and strip the algorithm to its bare bones and encode the specifics of the data in the self describing data files, not in the compiled code. By designing in low level support for operations on multiple data files, we make easier the task of doing the same operation across such sets of data files and, if desired, combining the results. The existence of a unified data model will make communications among programs, and between programs and GUIs, easier to systematize. By including uncertainties, upper limits, units, etc., in our data model, we standardize their treatment and so allow generic tools to operate on them.

## 1.4 Problems and Solutions

In this section I discuss various limitations we've come across in the way current systems handle abstract data manipulation. I concentrate on examples from PROS and FITS since they are the systems I am most familiar with.

- PROBLEM: PROS regions are handled in a different way from time, PHA filters.

- SOLUTION: Introduce the idea of a Data Subspace which handles filters on all data axes in a uniform way. The user can specify a spatial region anywhere they can specify a PHA or time filter. The Data Subspace for a data object records the way that object has been filtered. If you like, it is the filter that has been applied to the data so far. The Good Time Intervals are part of this filter.

- PROBLEM: Making a detector coordinate image is messy at the moment (PROS keyx, keyy syntax).

  ```
  qplist "test.qp[pi=40:90]" region="c 2048 2048 20"
  ```

  lists photons in a given sky region and PI range, but

  ```
  display "test.qp[pi=40:90]"
  ```

  does not take a region argument - you can't display it. To list photons in a detector coordinate region,

  ```
  qplist "test.qp[key=(detx,dety),pi=40:90]"
                     region="c 2048 2048 20"
  ```

  which is ugly because the specification of the region and the statement that the region applies to detector coordinates are separated.

- SOLUTION(1): Make regions part of the virtual file syntax, so you can do:

  ```
  qplist "test.qp[(detx,dety)=c 2048 2048 20,pi=40:90]"
  ```

  - this is much more coherent.

- SOLUTION(2) The scientist thinks in terms of 'detector position' and 'sky position' as single attributes of the data. Make our software able to work on two-dimensional items named 'DETPOS' and 'SKYPOS' to allow a natural system of

  ```
  qplist "test.qp[detpos=c 2048 2048 20,pi=40:90]"
  ```

  Make the data model support 2D objects with a name for the object and for each of its components (e.g. object name SKYPOS, component names RA and DEC). This makes it easy for a programmer to make a file which knows that it contains a bunch of SKYPOS each of which consists of an RA and a DEC. Current files don't have any way of letting the software know which columns are paired together as positions.

- PROBLEM: No standard way to record how the data has been filtered on PHA or PI.

- SOLUTION: The Data Subspace does this automatically. Thus the software will know where to look to find out which (energy-dependent) point spread function would be matched to the current image - it looks for a PI axis in the image's data subspace.
- PROBLEM: Lots of data products in my directory. Which one is the one I want?
- SOLUTION: An Observation Index file which is a stack of data products for a particular observation. This lets the software know automatically which file contains the exposure map and the events file for the observation, so the user doesn't have to type the filename explicitly. This file will be provided as a standard data product. The Data Model will support this by allowing table columns to have a special data type 'file' so that software can easily identify the presence of such an index to other files.
- PROBLEM: Some data manipulation tasks need you to go back and forth between header keywords and table columns, but header keywords in FITS don't contain as much information as table columns (short names, no units, no vectors). Examples: we wish to combine event lists from ACIS chips I2 and I3, which have header CHIP_ID values giving the chip ID, getting an event list with an extra CHIP_ID column in which each row is either I2 or I3. Or, we wish to combine tables of sources detected with three different cell sizes, to make one table with a CELL_SIZE column. The resulting table needs to know the units in which CELL_SIZE is measured. Actually, it would currently have to be CELL_SIZ since the header keywords can only be 8 characters.
- SOLUTION: The data model will support the extra information. The I/O library handles a convention to write this to FITS in a way that is back compatible with existing data. The tool program can ask for the same information about a keyword that it would for a column entry, so the code is more uniform - fewer special cases.
- PROBLEM: We have a blocked sky image and want to know about both the original plane pixel coordinate system and the celestial spherical coordinate system. FITS only supports one set of WCS keywords for an image.
- SOLUTION: For array objects, allow an 'axis pixel' coordinate system and an 'axis coordinate' coordinate system to retain both sets of information. An alternate approach would be to allow arbitrary numbers of coordinate systems for each object, so that for instance one could attach a galactic coordinate system to the image as well. I have decided that this is not such a good idea, and is better handled by using a coordinate conversion program to transform the celestial systems.
- PROBLEM: Want a single program to browse and plot all kinds of data files, labelling axes sensibly.

- SOLUTION: Each axis in the data is liable to have both a local and a 'world' value: pixel position and celestial position, mission time in seconds and calendar date, pulse height and nominal energy. The data model treats all of these as generic coordinate systems, so a plotting program can recognize them automatically. Example: pulse height versus time image, with nominal energies and calendar dates automatically labelled.
- PROBLEM: Want to support a table with images embedded in one of the columns, for instance aspect camera records.
- SOLUTION: Introduce a data model convention to handle this case, which is supported to a limited extent in FITS by the multidimensional array TDIMn syntax; further simplify by considering an ordinary image to be a special case of a table with one row and column.
- PROBLEM: Want to create datasets such as an array of x-ray colors versus best fit parameters, and invert to make an array of best fit parameters versus x-ray colors.
- SOLUTION: Provide data model support for arrays whose elements are themselves n-dimensional.
- PROBLEM: Want to deal with upper limits properly.
- SOLUTION: The interface to the data files should be able to cope with any data item being either a detection, an upper limit, or a detection with uncertainty. Other software, however, will see the uncertainty ranges as separate columns and won't know that a particular value is an upper limit.
- PROBLEM: We have a set of PSFs which were created at XRCF at different energies; they are labelled with a 'header keyword' ENERGY. We wish to plot the FWHM of the PSFs versus energy. In an existing system, one would run the calculate-FWHM program on each PSF file separately, capturing the results and running a table creation program to combine them in a single result table (or noting them down on paper and typing them back in!); plotting the results might not be trivial either.
- SOLUTION: We should be able to do this with three commands: one to stack the PSF files on energy, creating an index file consisting of a table of energy versus filename, a second to run the calculate-FWHM program on the stack, and the third to plot the resulting file. In our system, the added bonus is that if the calculate-FWHM program also calculates uncertainties, these will be picked up by the plotting program.
- PROBLEM: In a derived file like a light curve, we may make many columns (raw counts, background counts, net count rate, etc.) even though the basic concept is of time versus net count rate. We want our plotting software to plot the two columns of most interest by default. Also, indexing operations may also be carried out on event list columns of 'most interest'.

- SOLUTION: Define 'preferred' columns (axes) of the table, which will rank a subset of the columns in an order which may be different from the order of the columns in the table. A plotting program which plots two quantities against each other will then take the first two preferred columns if such exist, otherwise it will take the first two columns in the table column order.

## 1.5 Informal Introduction to the Data Model

In our model, each dataset consists of an ordered set of 'Datablocks or simply 'Blocks'. An ASC Block consists of a table whose columns may be scalars, vectors, arrays, or ranges. Header quantities may be attached to the table as a whole, or to individual columns or to the data subspace. An important special case of a Datablock table is called an Image, and we will often consider Datablocks to be of two types, Table and Image (even though strictly speaking an Image can be treated as a kind of Table).

A table consists of a header, together with a set of rows and columns. I will refer to the intersection of a row and a column as a 'cell'. We expect that some of our tabular data products will contain small embedded images. For instance, aspect camera data will include 6x6 pixel images of each fiducial light in every row of the table (the row represents the information from one aspect camera exposure). Also, we may include small 'postage-stamp' images of sources in our source list data product. This suggests a theoretical simplification: if an image can be in a cell of a table, we may consider an image on its own to be just a table with one cell. So, instead of two different fundamental types of data, we have a single type - the 'table-which-can-contain-images'. We can then specify that a table with one row and one column may, if desired, be stored on disk using an image format instead of a binary table format. We have thus moved the distinction between a table file and an image file to a different level: an image is a component of a table, rather than its peer. We still, of course, need to have interfaces to operate on images, so this simplification is minor in practical terms.

The header in a FITS file is a heterogeneous collection of information. Some of the keywords describe the file's structure, while the remainder are metadata: data which apply to the file as a whole, but are true science data rather than descriptive of the file structure. We want to layer extra structure on the file so we can tell the difference between these types of header keyword. Some of the metadata has particular importance: it describes how the data in the table columns was selected. We treat this kind of information in a systematic way and isolate it conceptually as the table's 'data subspace'.

The figure below gives a schematic example of a complicated table.

## 1.6 Table columns

FITS already provides support for vectors and arrays in table columns. However, there are several enhancements we need. Particularly for the case of positional data, we want to have paired table

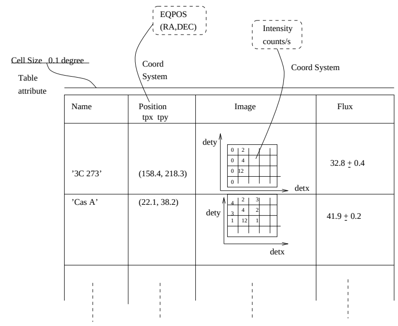


Figure 1: Example of a complicated table. The table is a source list containing 'postage stamp' images of each source. The position column has a coordinate system attached to it, the flux column has uncertainties, and the whole table has metadata such as the source detection cell size.

columns: for instance, DETX and DETY paired as DETPOS, or RA and DEC paired as EQPOS, with both the individual and the pair names available in the file. We also want to support uncertainties and upper limits, which implies something like having a column FLUX and a column FLUX_ERR (no problem right now) together with a structure which ties the two together as a single object (Flux with error). Both of these enhancements, and the desire for back compatibility, lead us to a system with a low level (FITS) set of raw columns and a high level (Data Model) set of columns, with one high level column mapping to several low level columns.

## 1.7 Table Attributes

Table attributes are the equivalent of header keywords. Unlike FITS header keywords, we support the various Quantity attributes such as units, etc. FITS allows 'indexed keywords' which are really 1-D arrays of keywords: we want to support this at a higher level, and add support for 'vector keywords', e.g. grouping together RA and DEC as a single high level table attribute EQPOS.

We'd also like to specify some attributes as belonging to specific table columns rather than to the table as a whole. These are called column attributes. Similarly, the data subspace may have its own attributes: livetime is an example.

## 1.8 Binned Data

An event list table consists of values which represent precise points in an n-dimensional space. In contrast, we often deal with binned data in which the values represent cells of finite volume in the space. The simplest example is a histogram with equal size bins, but we also have datasets with logarithmic bins or even arbitrary bins (e.g. those chosen to match the position of sharp features in a spectrum). A binned data column can use the same mechanism as the uncertainties for a normal column, since it just involves specifying a range.

## 1.9 Arrays and Images

When we have binned data with ordered, equal size, contiguous bins, the column of data may be defined implicitly by specifying the start value and step size. Suppose we have a table whose columns include three binned data columns and two point data columns, one of which happens to be a 3D position:

```
C1          C2            C3           C4       C5
[0.5:1.5)   [10.0:11.0)   [4.8:4.9)    1082.2   (0.0, 18.3, -812.3)
[1.5:2.5)   [10.0:11.0)   [4.8:4.9)     182.3   (4.3, 12.2. -712.3)
....
[0.5:1.5)   [11.0:12.0)   [4.9:5.0)    1211.2   (2.1, -1.2, -271.3)
[1.5:2.5)   [11.0:12.0)   [4.9:5.0)    1232.1   (6.2, -4.2, -0.023)
....
```

Here the rows are ordered so that C1 changes most rapidly, followed by C2 and then by C3 so that the grid of cells in the three dimensional C1, C2, C3 space is traversed in a regular order. We can replace this table by one in which only the values for C4 and C5 are included explictly. The information about the binned C1, C2, and C3 datasets are stored in the descriptions of the structure of quantities C4 and C5. C4 is a normal 3-dimensional image; the pixels of the 3-dimensional array of values in the C4 column are mapped to values of C1, C2 and C3, which are called the axes of the image. C5 is a more complicated object, an image whose pixels are vector-valued. Support for objects like C5 (arrays of vectors) is new, but gives added consistency to the data model. Arrays of vectors are useful, for instance, when the varying centroid position of a source is measured as a function of several parameters.

## 1.10 Elements and Quantities

The building blocks of our data are called Elements and Quantities. Let's consider a simple physical quantity: the energy of the Fe K line, which we wish to store as an object FE_K_ENERGY. Suppose we have measured it to be $6.4 \pm 0.3 keV$. We will store the name FE_K_ENERGY and the unit keV as part of a Quantity of real data type. Associated with this Quantity is a single Element of dimension 1, which consists of a Value, the number 6.40, and an Uncertainty Range, the three sigma range [5.5:7.3]. We store the range since this lets us easily handle the case of upper limits: an upper limit is just an element for which the lower bound of the uncertainty range is zero or negative. If we get a whole set of measurements of FE_K_ENERGY, we retain the single Quantity and associate many Elements with it.

A more complicated Quantity/Element combination would be EQPOS, the equatorial position of something. EQPOS has a name, a unit (degrees), and two new things: a dimensionality, equal to two, and a set of component names (RA and Dec), one for each dimension. The corresponding

Element is an Element of dimensionality 2. It has two Values and an Uncertainty Ranges of dimensionality 2. A Quantity of dimensionality 1 is called a Scalar, while a Quantity of dimensionality 2 or more is called a Vector.

Another type of Element is the pure Region Element, which has an Uncertainty Range but no Value. Region Elements are used to describe filters, regions, intervals, etc.

Multiple Elements associated with a single Quantity are called Arrays. Arrays of scalars are familiar; arrays of vectors are more complicated, but are sometimes needed. The simplest kind of array is a one-dimensional array, which simply has a given number of elements. Note the difference between an array with dimensionality 1 and dimension n, and a vector with dimensionality n and dimension 1.

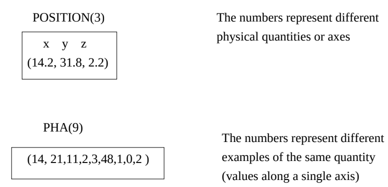


Figure 2: Difference between a vector and a 1-D array. In the first case, each component has a name (e.g. 'y'); you would plot the n-tuple as a single point in n-dimensional space. In the second case, the different components do not have names. You would plot this as 9 different points along a 1-dimensional space. We also use arrays of vectors: for example, PSF centroid position versus energy and off axis angle.

## 1.11 Stacks

To work more effectively with multiple sets of data, we introduce the concept of stacks. The simplest stack is just a list of files. However, a more powerful kind of stack is a table one of whose columns contains filenames: in other words, we have a list of files which is **labelled** by the other columns. As an example, let us consider a set of point spread function calibration images which have been taken at some quasi-random set of energies and off axis angles and have similarly random filenames PSF42, PSF13, PSFA1, etc. We make a table PSFSTK as follows:

```
ENERGY  THETA   PSF_FILE
real    real    file

0.3     42.1    PSF42
0.3      0.1    PSFA1
....
5.2      0.2    PSF13
```

This gives us a 'library' of PSFs which we can look up as a function of the two parameters ENERGY and THETA. If the ENERGY and THETA parameters are table attributes (header keywords) in the individual PSF files, we can imagine a program which would make this stack file PSFSTK automatically by saying: look at all the files in this directory, and for each file with a table attribute OBJECT whose value is equal to 'PSF', add a record to the stack labelled with the values of the table attributes ENERGY and THETA. I will call this operation 'stacking (a set of tables) on ENERGY and THETA'.

We then define a 'stack operation' at the tool level as follows: if the effect of a tool T on a non-stack file F is to make a multi-line table T(F), then the effect of the tool on a stack is to make a new stack table where each entry F in the stack column is replaced by the name of T(F). If the effect of T is to make an output file with a single line, then the entry F is replaced by the contents of that line (so the output file is no longer a stack but a single table).

To continue the earlier example, consider two tools T1 and T2, where T1 takes the histogram of the image pixel values, and T2 returns a one-line table containing the centroid position and total counts. Running T1 on PSF42 makes a new file PSF42_IMHIST (say) with several rows and columns. Running T2 on PSF42 makes a new file PSF42_CTR with several columns but only one row:

```
XCEN  YCEN  TOT_CNTS
real  real  integer

42.3  121.2 141412
```

Then running T1 on PSFSTK should make a new stack as follows:

```
ENERGY  THETA   IMHIST_FILE
real    real    file

0.3     42.1    PSF42_IMHIST
0.3      0.1    PSFA1_IMHIST
....
5.2      0.2    PSF13_IMHIST
```

as well as making all of the individual IMHIST files. But running T2 on PSFSTK should make a single file

```
ENERGY  THETA   XCEN     YCEN   TOT_CNTS
real    real    real     real   integer

0.3     42.1    42.3     121.2  141412
0.3      0.1    52.1    1109.1   32821
```

```
....
5.2       0.2   9212.2    104.2    1821
```

The power of this is that it allows us to do aggregate analysis easily: we can now use the generic plot tool to plot, say, XCEN versus THETA to see how those two parameters vary with each other.

The functionality of stacks is really at a higher level than that of the data model described in this document, but it is presented here for background.

# 2 Some general requirements

## 2.1 Data Model and files

We require that the data model reflect the structure of our science data as generally as possible. Our paradigm for analysing data involves applying tools (programs) to one or several input data files, and generating output files. Data files may be 'standard data products' whose structure and contents are predefined in detail by the ASC, 'user-derived data files' which follow our general paradigm but whose detailed structure is specified by the user, and 'compatible data files' which are produced by external analysis systems (including archives of older missions) but which are sufficiently similar in structure that our software can recognize them. It turns out that almost all our data can be described in terms of instantiations of a single kind of object, which I will call an ASC block or ASC Table. There is also a special flavor of ASC block called an ASC Image which is treated separately in some cases.

A requirement is that the division of our data into separate files should 'make sense' to the scientist, logically related information being kept together. An obvious way to do this in the object-oriented paradigm is that each file should contain exactly one ASC Table. However, this will not work in all cases.

We require that the data model allow the applications programmer to ignore the details of the specific file format conventions (e.g. FITS, QPOE) but also allow some measure of override access to the specific file format writing kernels. At least two kernels should be supported by the model, to support writing and reading FITS files and IRAF files. By IRAF files I mean IMH files, PROS QPOE files, ST Table files, and possibly ETOOLS EDF files.

## 2.2 Compatibility Requirements on FITS kernel

We require that as many of the following existing archival FITS datasets should be readable by the FITS kernel as valid Science Datasets: Event lists and XSPEC-type PHA and response matrix files for the following missions: Einstein, ROSAT, ASCA, and XTE. This imposes requirements on the FITS keywords used to map data model structures.

## 2.3 The native data model in FITS

FITS files contain a set of independent Header Data Units (HDUs). There are several flavors of HDU but the most important ones are IMAGE and BINTABLE. We will consider a FITS file containing only IMAGE and BINTABLE HDUs. The HDU consists of a header and a data section.

- The header consists of an arbitrary number of header cards.
- A header card contains a keyword (an 8 character case-insensitive string), a value (of one of a number of data types), a comment (a string which is usually ignored by software), a data

type (which is not given explicitly, but may be deduced from the formatting of the value), and a card type (deduced from the keyword name).

- The card types are: Mandatory cards, standard reserved cards, local reserved cards, and ordinary cards. By local reserved cards I mean cards whose keywords are not reserved from the point of view of the FITS standard but which are given a reserved meaning in some extra convention to which the particular FITS file adheres. An example is the WCS convention which has been proposed for inclusion in the standard and reserves the meaning of several extra keywords.

- An IMAGE data section consists of an n-dimensional array of numerical values. Associated with the IMAGE data is the data type of the array elements, the number of dimensions, and the size of each axis. This information is contained in reserved header cards; scaling and unit information about the data and coordinate information about the axes may also be associated with the image in this way.

- A BINTABLE data section consists of a set of columns. Each column has a data type and a name, and possibly a unit and various coordinate information. All the columns have the same number of entries.

## 2.4 Interaction of Data Model and other infrastructure

The data model will affect the other infrastructure parts as follows:

- The filtering language describes a virtual dataset in terms of a preexisting one. This description should be complete in the sense that it fills in all the interface requirements of the data model.
- The data model I/O should work on filtered (virtual) datasets. Although the filtering routines will use the data model, the interface to the data model and the filtering are somewhat separable, but we may want to integrate them for implementation efficiency.
- Parameter files should have a rich enough syntax to fully specify a data model scalar or vector. There should be a way of reading parameter files to instantiate the parameters as data model objects.
- Otherwise, the data model stuff should be pretty much decoupled from work like the composite tools, the fitting model language, etc.

# 3 The ASC Data Model, SDS Version 1.2

In this section I present the SDS design for a data model, developed by Jonathan McDowell based on extensive discussions with Martin Elvis, David Van Stone and Peter Patsis. The model represents a very general kind of table, whose columns can contain vectors or multidimensional arrays, with associated coordinate systems and other metadata. Further metadata can be associated with the table as a whole or with individual columns, and a 'data subspace' indicates the range of values for which the table is valid.

*This version 1.1 (Mar 1997) has minor differences from the version 1.0 document of May 1996. It includes the addition of unsigned types and variable length array columns. The FITS implementation and the API have been split out into separate documents; the API is substantially revised.*

*Version 1.2 (1997 Mar 5) includes handling of null primary arrays in FITS files and the idea of a kernel marker in the data descriptor to support this.*

A note of explanation is required for object-oriented fans (others may skip this paragraph). A Rumbaugh diagram for our design, shown below, indicates that only a small number of distinct objects are used. However, I feel that the Rumbaugh methodology, at least as I have been made to understand it, obscures understanding of the true structure of our data in which multiplicity and aggregation of instances play a key role. I therefore use a slightly different kind of diagram, which I will call a structure diagram, which includes structural components which are not necessarily distinct objects in the OO sense, and which shows separate instances of an object if (and only if) the object is instantiated in a separate role. All of the associates represent 'has a' relationships.

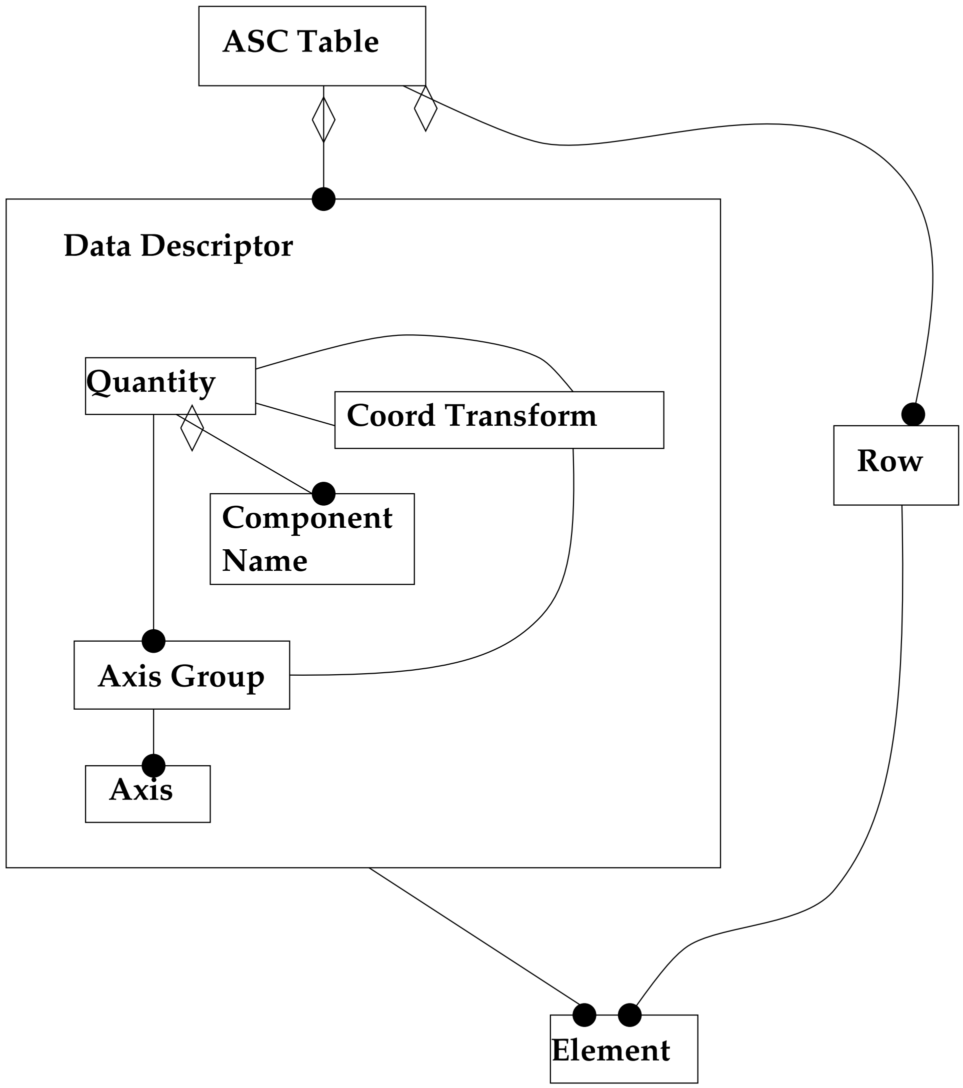


Figure 3: Rumbaugh diagram for ASC Table/block Data Model, showing the fundamental different object classes. The Data Descriptor class is a composite whose components are shown within an enclosing box.

In less technical language, I'm trying to present an abstraction of a particular kind of scientific dataset. The diagrams I show are an attempt to illustrate the different components that go to make up this abstraction. Each box is one of these components, and a line going out of the bottom of one box into the top of another indicates that the second box is a component of the first. A symbol by the end of the line indicates the number of such components that will exist. For instance, in the Table Model diagram the symbol c appears next to both the line from Table Data to Column Data Descriptor and the line from Table Row to Column Data Cell. This indicates that there are the same number of Column Data Cells in a Table Row as there are Column Data Descriptors in a Table Data, and that we will represent this number as $c$ (it happens to be the number of columns in the table). If no multiplicity is indicated on a line, there is exactly one of the components attached to the parent object.

Text underneath a horizontal line below the name of the object indicates parameters (attributes) associated with that object. For instance, the ASC Table has a Name. A double line at the bottom of an object box indicates that the structure of the object is complex and there's a whole separate diagram for it later on. For instance, DSS Data Descriptor is a particular kind of Data Descriptor object; the Data Descriptor has a diagram of its own, and the text below defines what kind of a Data Descriptor a DSS Data Descriptor is (For instance, unlike a general DD, it can't be an Array.)

All multiple subcomponents are considered to be ordered. In other words, there is a defined order of the columns in a table, a defined order of header keywords in the header, and a defined order of the axes in an array. One may refer to a column by its name (e.g. TIME) or by its number (e.g. Column 4). A single ASC Table Column may map to several columns in an underlying table format (e.g. FITS BINTABLE), and in general the numbering of an ASC Table component is distinct from the numbering of the corresponding structure in the underlying data format.

### 3.1 ASC Table

The highest level object is the ASC block. It consists of three main parts: the Table Data proper, the Header, and the Data Subspace. Each of these contains Data Cells made up of arrays of Elements which contain the actual data, and Data Descriptors which provide metadata about the meaning of the Elements.

The ASC block has a single attribute of its own: the table name.

## 4 Table Data Section

### 4.1 Table Data

The Table Data Section represents a table with $r$ rows and $c$ columns. The intersection of a row and a column is a Data Cell; all the Data Cells in a column have the same structure, and contain the same type of data; they are described by the Data Descriptor for that column. The different columns in a row may have different structures.

Associated with the Table Data section is an ordered list of Preferred Columns, as a hint to generic software which only operates on a given number of columns without specifying specific column names.

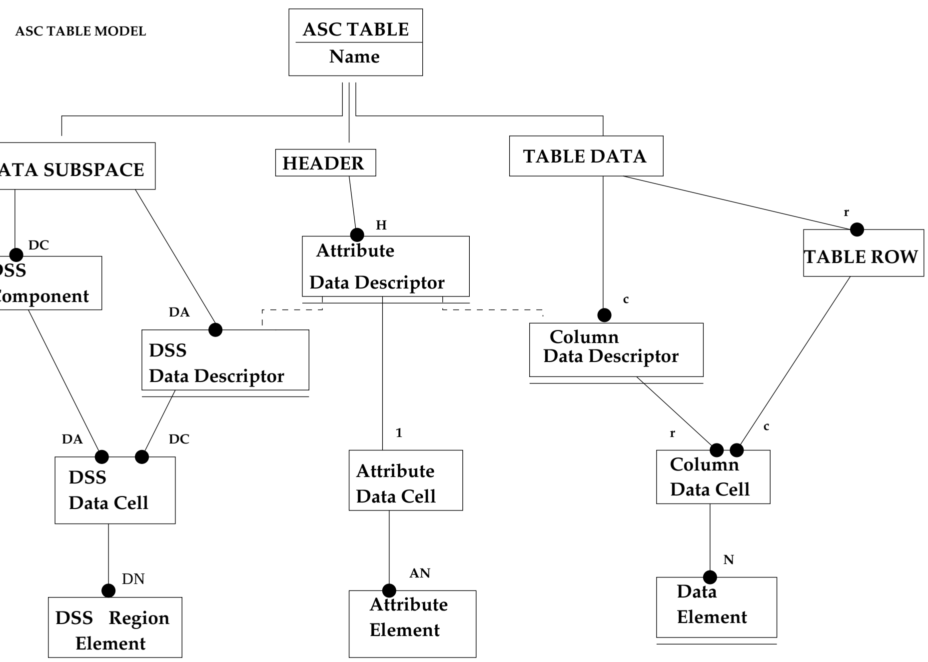


Figure 4: Data Model 1: Overall structure of the data model, showing the ASC Table, used as the highest level data object encapsulating all others.

## 4.2 Quantity

The Quantity object is proved to describe the structure and properties of named quantities. It is basically a structure which provides a name, a unit and other descriptive information, and specifies a data type. It is the abstraction of the FITS BINTABLE's TTYPEn, TUNITn, etc., header keywords for a column.

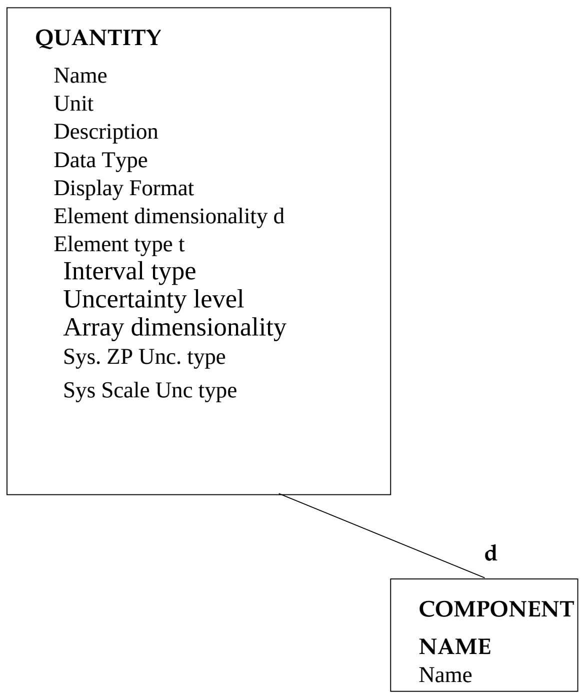


Figure 5: Data Model 2: The Quantity object, used to describe the structure and properties of named quantities (header attributes, table columns, data subspace axes, world coordinates
.

We provide the following information for each quantity (much of which may be omitted in actual disk storage if it is equal to the default value):

- The quantity name, a character string. Any ASCII character string shall be supported, but at least for now we shall require that for Quantity objects used in the Data Descriptor, the string shall consist of only alphabetic upper or lower case letters a-z,A-Z; numeric digits, 0-9; the symbols +,- and underscore (_). In particular, spaces are not permitted (except trailing spaces which are not considered to be significant). At this level, case is significant, although we anticipate that user access routines will not be case-sensitive and recommend that names be unique within a table even when case is ignored. The idea here is that we may want to name something MaxVoltage instead of MAXVOLTAGE so that the software knows how to print it nicely, but we don't want to require that the user has to get the capitalization right when searching for it. So case is remembered, and returned correctly, but matches are case insensitive.

- The quantity unit. A character string which specifies the physical unit. Should comply with the HEASARC/OGIP format on unit strings or the JCMLIB specification for unit strings.

- The Description is a string which is used to label human readable output such as ASCII print files and graphical axis labels. It is a longer name which may include spaces and other special characters, including backslash. I suggest the use of TeX escape sequences which are supported by some graphics libraries such as SM, for instance '\ alpha' for $\alpha$. Thus a Quantity might have the name 'RA' and the description '\ alpha (J2000.0)'.

- The Display Format indicates the preferred output format for a single data value associated with the quantity in a text browser. It is required that the Display Format can be returned as a string Fortran format specification compatible with the TDISPn keyword in a FITS file. This optional information may be provided as a hint to browsers to let them format tabular output efficiently. For instance, a quantity stored as a 4 byte integer might be known to only take values less than 1000, allowing a display format of 'I4' instead of the larger 'I10' needed by an arbitrary 4 byte integer. Pixel coordinates might be displayed as 'F8.3' while a time specification in seconds might require the greater precision of 'F20.6'. However, the display format may be absent or the browser may choose not to use it, it's just provided to help make the output pretty.

- The Data Type indicates the type of data in an associated Element. Supported data types shall include:
  - Integer, 2 byte
  - Integer, 4 byte

– IEEE Real, 4 byte. The specification of IEEE here indicates that it must be possible to return the data in IEEE format, and it must be possible to store IEEE special values such as NaN and -Inf. How the data are actually stored internally or in a data file is an implementation detail.

– IEEE Real, 8 byte

– Logical, 1 byte

– String, specified fixed number of bytes $s$.

– Datablock Reference, with specified number of bytes $s$. A Table Reference is a string which contains a pointer to another named Datablock. A datablock reference 'Table1' refers to a table named 'Table1' in the same directory. The table is assumed to be in a file of the same name, but we will also support a syntax 'filename[tablename]' to refer to a table named 'tablename' in file 'filename'. A reference 'old.fit[Table1]' refers to a table named Table1 inside a file named old.fit in the current directory (although we will recommend that only one ASC Table be stored per file, an arbitrary FITS or QPOE file may in principle contain several unconnected extensions which would be interpreted as separate ASC Tables. ) A reference 'subdir/Table1' refers to the first table in a file Table1 in a subdirectory subdir. In other words, we will use the Unix-compatible directory character / to indicate relative directory paths. We will further specify that higher level paths shall follow the URL convention, so that

```
URL:ftp://sao-ftp.harvard.edu/pub/jcm/asc/test.tab[Table1]
```

refers to a table Table1 in a file test.tab with the specified access via anonymous ftp. This is not to be taken as a requirement that our software should support accessing such URLs, I just want to specify a standard way to store them.

– *Unsigned Byte, 1 byte*

– *Unsigned Integer, 2 bytes*

– *Unsigned Integer, 4 bytes*

*In addition, the following data types are under consideration for support:*

– *Extended Unsigned Integer, 6 bytes*

– *Extended Real, 12 bytes (needed for full time precision)*

*The Bit datatype supported by FITS will not be implemented; it will be interpreted as Unsigned Byte instead and promoted to a multiple of 8 bits. Complex datatypes will not be supported.*

*Table 1: Codes for Data Types*

| *Data Type* | *API routine suffix* | *FITS CFORM* | *FITS TFORM* |
|---|---|---|---|
| *Integer/2* | *s* | *'I'* | *'I'* |
| *Integer/4* | *i* | *'J'* | *'J'* |
| *Real/4* | *r* | *'E'* | *'E'* |
| *Real/8* | *d* | *'D'* | *'D'* |
| *Logical* | *q* | *'L'* | *'L'* |
| *String* | *c* | *'A'* | *'A'* |
| *Block Ref* | *br* | *'R'* | *'80A'* |
| *Unsigned/1* | *b* | *'B'* | *'B'* |
| *Unsigned/2* | *su* | *'IU'* | *'2B'* |
| *Unsigned/4* | *iu* | *'JU'* | *'4B'* |
| *Unsigned/6* | *ie* | *'IE'* | *'6B'* |
| *Real/12* | *de* | *'DE'* | *'J' + 'D'* |

- The Element Dimensionality specifies the dimensionality $d$ of all Elements associated with this Quantity. The default is $d = 1$.
- The Element Type can be Value (V); Value with Uncertainty (U), Value with Fixed Uncertainty (UF), 2D Region (REG), Bin (BF), Bin Start (SF), etc. The different element types are discussed in full in the section on elements. All Elements associated with the Quantity must be of the same Element Type.
- An associated Interval Type may be defined: see the discussion of Intervals.
- A Kernel marker. This is a placeholder to support extra information needed to reconstitute a clean file for a particular kernel. For the FITS kernel, this marker is needed for header keywords in a binary table. The kernel marker values 'HP', 'HT', 'HB' mean that a header attribute belongs to the null primary header, to the main bintable, or to both, respectively.
- An associated Uncertainty Level may be defined, between 0.0 and 1.0. The range is then considered to describe the uncertainty at the given percentage confidence interval. The default value is 1.0. A common use is to give a value of 0.68 for an element of type U, in which case the uncertainty range is interpreted as 1 sigma errors.

  A possible alternative use for the uncertainty level would be to indicate for a 2D region representing an X-ray source region the fraction of counts within the region. (A separate U element might give the source centroid and position uncertainty).
- Array Dimensionality $n$ specifies the dimensionality of the array of Elements making up a single Cell associated with the Quantity. If $n > 0$, there must be an Array Specification associated with the Quantity; if $n = 0$ there is no Array Specification. All Cells associated with the Quantity must have the same array dimensionality and array specification.

- If $d > 1$, there are a set of $d$ Component Names which identify the names of each component of the associated elements. For instance, we might define a 2-dimensional Quantity with name SKYPOS and component names RA and Dec. If $d = 1$ then the single component name is defined to be identical with the Quantity name.

We name certain special cases:

- 
  - A Quantity with $d = 1$ and $n = 0$ is known as a Scalar Quantity.
  - A Quantity with $d > 1$ and $n = 0$ is a Vector Quantity.
  - A quantity with $d = 1$ and $n > 0$ is a Scalar Array Quantity.
  - A quantity with $d > 1$ and $n > 0$ is a Vector Array Quantity.

- *Finally, a descriptor may have a Comment field. This Comment field consists of arbitrarily many 72-byte text strings each with an associated 8-byte tag. The default value of the tag is the string 'COMMENT'. Other values of the tag are not guaranteed to produce valid files for all kernels, although 'HISTORY' and blank are valid for FITS files. The Comment field text may appear in the underlying file header anywhere following the appearance of the descriptor name and preceding the next descriptor name.*

## 4.3 Array Specification

An Array Specification describes the arrangement of a set of N elements into an n-dimensional array. The $n$ axes of this array, $i = 1, ...n$, have dimension (size) $n_i$, so that

$$\prod_{i=1,n} n_i = N$$

The elements $E(p_1, ...p_n)$ of the array are labelled by array pixel numbers, which are an ordered n-tuple $P = (p_1, ...p_n)$.

## 4.4 Array Axis

Each axis $i$ of the array is defined by a given dimension (size, number of pixels) $n_i$. We adopt the FITS (and Fortran) convention in which the pixel numbers start at one, and in which a default storage order is implied in the following sense: an Element Number $e$ is defined equal to

$$e(P) = p_n + (p_{n-1} - 1) * n_n + ((p_{n-2} - 1) * n_n n_{n-1} + ...$$

or

$$e(P) = \sum_{i=1}^{n} \left( (p_i - 1) \prod_{j=i+1}^{n} n_j \right)$$

where $\prod_{j=n+1}^{n} n_j$ in the final term of the sum is interpreted to be equal to one. A mechanism will be supplied to return the elements in element number order. In FITS files and in Fortran arrays, the array elements must be actually stored in element number order.

## 4.5 Axis Groups

We add a little extra structure to the array to group axes which may have common coordinate transforms. In our model we consider something like detector pixel position to be a single, two-dimensional, quantity; if we have a data cube of detector pixel position DETX, DETY versus energy E we wish to emphasize the fact that DETX and DETY are related to each other in a way that they are not related to E. In this view, the three dimensional data cube DETX,DETY,E is instead a two dimensional array with two axes DETPOS and E, in which the first axis is itself two-dimensional. This first, two-dimensional axis may have a coordinate system on it which applies a two-dimensional spherical rotation, or it may have a mask on it which specifies a two-dimensional region, in each case requiring that treatment of DETX and DETY be coupled. In contrast, we do not expect to get situations where we must treat DETX and E in a coupled way (if we do, they will have to be treated at a higher level).

The Array Specification adds the concept of Axis Groups. In the example above, the three dimensional array has two axis groups, one a two-dimensional axis group containing the first two axes and another one-dimensional axis group containing the third. We can label the array by axis group pixel numbers $P_G = ((p_1, p_2), p_3)$, an ordered pair of a two-dimensional detector position pixel and an energy bin.

Say there are $g$ axis groups each of dimensionality $g_m, m = 1, ...g$. We have

$$\sum_m g_m = n.$$

## 4.6 Coordinate Transform

A Coordinate Transform maps the Elements associated with one Quantity to the Elements associated with another. A simple example is the mapping of mission time TIME in seconds to Julian Date JD in days. We define this transformation by choosing a reference value of TIME (usually 0.0) and the corresponding reference value of JD (the JD when TIME is equal to 0.0; say 2445200.0), and defining the transformation relative to this reference value. If TIME is the correct time in seconds since the reference value, then the transformation type is LINEAR and the transformation scale is 1.0/86400.0 (the number of days in a second); this completely determines the transformation. If TIME is a spacecraft clock with glitches and resets, the transformation may be a lookup table or a polynomial with a more complicated definition.

In general, we consider a coordinate transform to link two Quantities which have the same Element Dimensionality $d$. One Quantity is referred to as the Pixel Quantity and one as the World Quantity (this does not necessarily imply that the Pixel Quantity has units of pixels; the names evoke the FITS keywords CRPIX and CRVAL). The transform consists of a Coordinate Transform Specification which has a Transform Type, a set of $d$ Transform Scales $\Delta_i$ ($i = 1, ..d$), and associated Transform Parameters specific to the transform type. It also has a Reference Pixel Element and a Reference World Element, which are Value Elements (Elements of type V, see below) corresponding to the Pixel and World quantities. In our example above, the Pixel and World quantities are TIME and JD, and the Pixel and World Elements have values 0.0 and 2445200.0.

The point here is that we choose to represent an arbitrary transformation by a local linear transform about a reference point, plus higher order corrections. This has three advantages: it maps directly to the FITS CRPIX/CRVAL/CDELT convention; it ensures that we have a defined 'center' for our transformation, which can be used as a default location by an application; and often the transformations we use are linear, and don't require any higher order parameters, so it makes the usual cases simple.

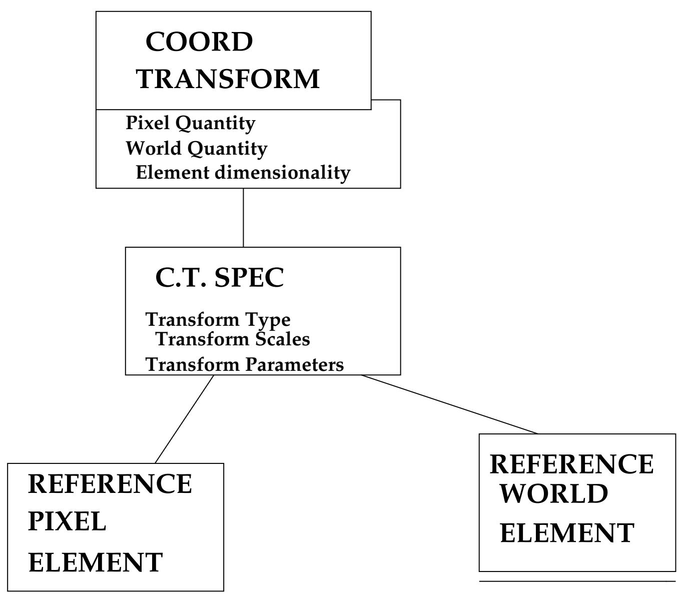


Figure 6: Data Model 3: Use of the Coordinate Transform. The Coordinate Transform object includes the reference pixel element and the reference world element, as well as the world Quantity. The World element is then defined in terms of the Pixel element and quantity with which the Transform is associated.

## 4.7 Column Data Descriptor

A Data Descriptor provides information about a quantity which we're going to provide values for. The simplest, minimal Data Descriptor is a Data Quantity which is a scalar Quantity. More complicated Data Descriptors provide support for vector Quantities, for Arrays (a Quantity with an Array Specification), and for associated coordinate and axis quantities.

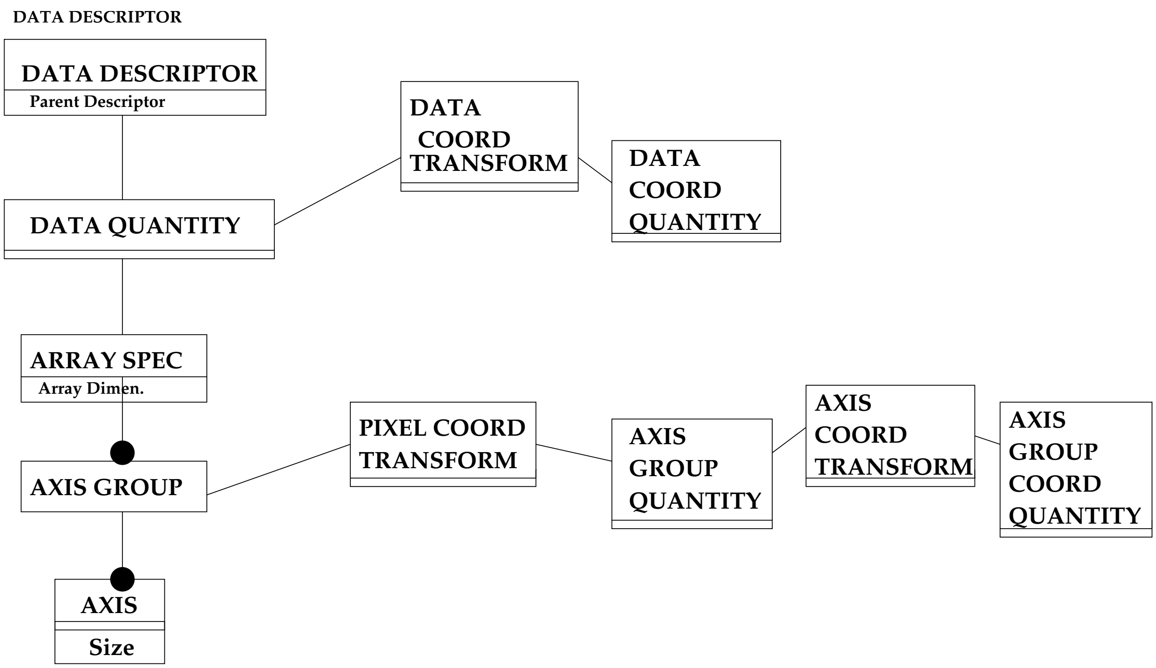


Figure 7: Data Model 4: The Data Descriptor object, which consists of a named Quantity and Array Specification with which data cells will be associated, as well as implicitly defined associated Quantities which label the axes and provide coordinate systems.

Every Data Descriptor has a single Data Quantity. A Data Descriptor whose Data Quantity is a Vector Array Quantity is called a Vector Array Descriptor, and so on. If the Data Quantity is an Array Quantity ($n > 0$) then there is an associated Array Specification with Axis Groups and Axes. A Scalar Data Quantity in a Column Data Descriptor is the abstraction of the FITS keywords TTYPEn, TFORMn, TUNITn, etc. An Array Data Quantity corresponds to the FITS BINTABLE multidimensional array convention for TFORM values. Vector Data Quantities do not correspond to any existing convention in FITS.

Associated with the Data Quantity there may be a Data Coordinate Quantity linked to it by a Data Coordinate Transform. For instance, a table may have a column TIME with values included explicitly in the table cells. The TIME column may have associated with it a quantity JD which gives the Julian Date. The individual values of JD are not stored explicitly, but are implied by the JD to TIME coordinate transform. JD is a Data Coordinate Quantity associated with the Data

Quantity TIME. The Data Coordinate Quantity and Transform are the abstractions of the FITS keywords TCTYPn and TCRVLn, TCRPXn, etc.

A Column Data Descriptor with a Data Quantity which is an Array has an Array Specification with one or more Axis Groups. Each Axis Group may have an associated Axis Group Quantity, related to it by a coordinate transform called a Pixel Coordinate Transform which must be of transform type LINEAR. The Axis Group Quantities are the labels of the axes of the array. For instance, we may have a Data Quantity PSF which is a three dimensional array with axis groups $g_1 = 2$ and $g_2 = 1$, associated with Axis Group Quantities DETPOS ($d = 2$, component names DETX and DETY) and ENERGY ($d = 1$). The element dimensionality of the Axis Group quantity must be the same as the dimensionality of the Axis Group.

Further, the Axis Group Quantities may themselves have associated Axis Group Coordinate Quantities related to them by Axis Group Coordinate Transforms. Consider another example in which the Array has $n = 2, g = 1, g_1 = 2$ and the single Axis Group Quantity is SKYPOS with components X and Y representing the X,Y sky pixel coordinate positions. We may associate with it an Axis Group Coordinate Quantity EQPOS with components RA and DEC, linked by a coordinate transform of type TAN, representing the actual equatorial sky positions. The Axis Group Coordinate Quantities are the abstractions of CTYPEn in a FITS image, while the lack of support for Axis Group Quantites themselves (such as SKYPOS X,Y) is an unfortunate limitation of current FITS practice.

## 4.8 Interval

An Interval defines a contiguous subset of the data values of the appropriate data type. Intervals are only meaningful for data types where a well defined ordering of the data values exists. For integer and real types this is the usual ordering; for string types this is defined to be the ASCII ordering.

The most general Interval is a minimum value, a maximum value, and an interval type. Possible interval types are closed, open, semi-open lower, and semi-open upper, denoted as [a:b], (a:b), (a:b], and [a:b) respectively. These are defined as:

$$\begin{aligned} x \in [a:b] &\quad \Leftrightarrow a \leq x \leq b \\ x \in (a:b) &\quad \Leftrightarrow a < x < b \\ x \in (a:b] &\quad \Leftrightarrow a < x \leq b \\ x \in [a:b) &\quad \Leftrightarrow a \leq x < b \end{aligned}$$

The semi-open intervals are useful for ensuring that boundary values are not counted twice. For integer and string data types, the only possible type of interval is Closed. This is also the default interval type for real data types.

## 4.9 Elements

The actual data for the table is stored in Elements. An Element must be associated with a Data Descriptor and its Data Quantity. A single Element contains values for one instance of the Quantity. For example, if the quantity TIME has element type Value with Uncertainty (VU) and element dimensionality 1, then an Element associated with TIME has one value of the TIME and one uncertainty for that value (in our terminology, an uncertainty is a potentially complicated object, which for now we assume is just an Interval - a single range of values, by default corresponding to a maximum and mininum 100 percent confidence range, but whose properties may be enhanced in a later revision of the data model). If the quantity DETPOS has element type Value (V) and element dimensionality 2, then a single Element of DETPOS has two Value Elements. The simplest kind of element is an element of type Value and dimensionality 1, which is a single value (numeric or string according to the associated Quantity's data type.)

The special element type REG applies only to 2-dimensional elements and is a string defining a region in PROS Regions syntax. With the exception of this element type, all d-dimensional elements consist of uncoupled element components for each of the d dimensions. The most general element component is a Value plus its Uncertainties or Ranges. We propose to support three different uncertainties: statistical, systematic zero point, and systematic scale. In addition, we define a total uncertainty which is a function of these three. We also use the same paradigm to record bin ranges. Our approach is to treat the systematic uncertainties as separate add-ons, with our default description being a single value and uncertainty, which is to be interpreted as a statistical uncertainty if the systematics are present and as a total uncertainty if they are not.

Frequently in a table column we wish to have different values and statistical uncertainties in each cell, but a single pair of systematic uncertainties for the whole column, expressed as an absolute zero point error and a fractional (percentage) scale error. An uncertainty which is constant over all cells is called a fixed uncertainty. Each uncertainty has a significance level; usually we quote Gaussian 1-sigma uncertainties (uncertainty level 0.68), although in X-ray astronomy 2-sigma (level 0.90) uncertainties are also common, and we may also have occasion to use full range (level 1.00) uncertainties.

The uncertainties can be represented in many different ways. Each representation is given a single-character code.

- If no uncertainty at all is present, the code is V (Value).
- The most flexible representation is the Interval Uncertainty (I) which uses an Interval to define the minimum and maximum values within the significance range. If the minimum value is zero or less, the measurement is termed an upper limit. If the Value component has value $v$, and the Interval has min and max of $v_1$ and $v_2$, then for a closed interval type

  $$v_1 \leq v \leq v_2.$$

  Note than the range center $(v_1 + v_2)/2$ is not necessarily equal to $v$.

- A second, more common representation is the Two Sided Uncertainty (T), in which the offsets $\sigma_+, \sigma_-$ from the nominal value are given. This has the advantage that it may be often used as a Fixed Uncertainty. In terms of the Interval Uncertainty,

$$v_1 = v- = \sigma_-, v_2 = v + \sigma_+.$$

- The One Sided Uncertainty (U) is the same as the two sided, but both upper and lower uncertainties are equal.

$$v_1 = v - \sigma, v_2 = v + \sigma.$$

- The Bin (B) is the same as the one sided uncertainty, but the full bin width $w$ rather than the half bin width $\sigma$ is given. This is more usually employed when the interpretation is a binned dataset rather than an uncertainty.

$$v_1 = v - w/2, v_2 = v + w/2.$$

- The Bin Start (S) is the same as the Bin, but the Value is deemed to be the start of the bin rather than the center:

$$v_1 = v, v_2 = v + w.$$

  This representation is often used for light curves.

- The Range (R) is the same as the Interval Uncertainty but there is no associated Value. If a Value is required, it is assumed to be $v = (v_1 + v_2)/2$.

- The scale uncertainty is always represented as a single nonnegative dimensionless real value (K) so that the implied range around the value $v$ is

$$v_1 = v(1 - K), v_2 = v(1 + K).$$

- We also want to support a two sided scale (L) with different upper and lower scale errors, which arises when we take the logarithm of a quantity with different upper and lower uncertainties.

- Finally, sometimes data is just provided in the form of detections and upper limits. We define an element type Z which consists of a value $v_d$ and a limit flag $f$, with the meaning

$$\mathrm{if} f \mathrm{then} v = v_d \mathrm{else} v_1 = 0, v_2 = v_d.$$

  However, I don't propose that we support this element type initially.

Each of these range types can be Fixed, in which case we append the letter F to the element type. We will further require that elements in a Table Column have a fixed Interval Type for all cells of the column.

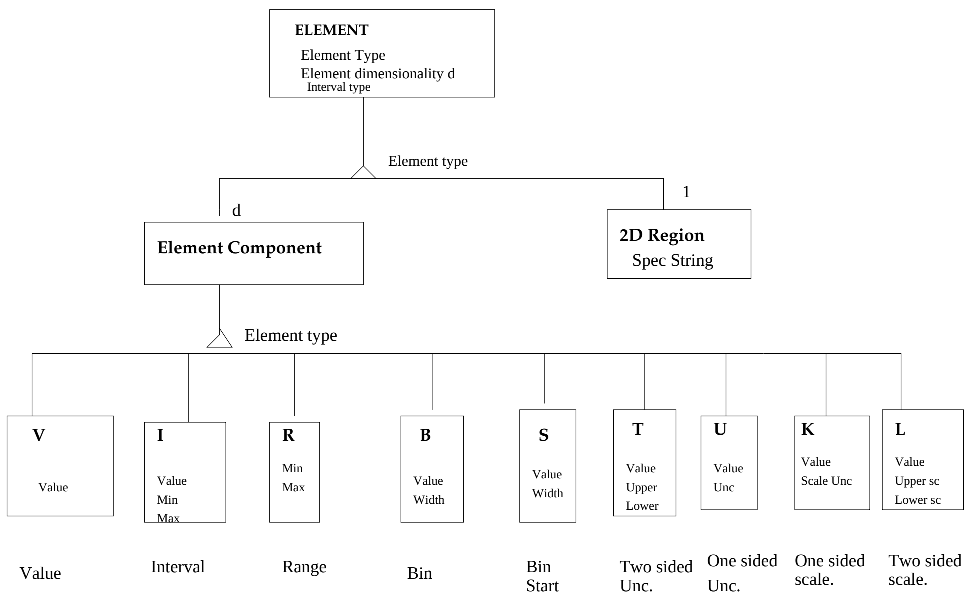


Figure 8: Data Model 5: The Element object, used to store the actual values. There may be many elements described by a single Quantity. There is one Element component for each dimension of the element, except if systematic uncertainties are included in which case there may be up to three Element components for each dimension.

We will later add to the Quantity object a systematic zero point uncertainty type and a systematic scale uncertainty type, the default values of which are null (not present). The legal values are the same as for the Element type, and if they are present the usual values are UF for the zero point uncertainty and KF for the scale uncertainty.

## 4.10 Region Description

For the two dimensional region descriptions we would like to support those in current systems, namely:

- Bitmap: appropriate for a binned dataset, provides a list of the pixels in the region.
- Polygon: an ordered list of n points describing a closed polygonal region.
- Shape: A parameterized shape, including the cases Circle, Annulus, Ellipse, Box, Pie.

We can describe the Shape with the following parameters:

- Shape type: elliptical or rectangular.
- Shape center x0, y0.
- Shape radial range r1, r2, interval type. If r1=0, have a Circle or Box. If r1>0, have an Annulus or annular box.
- Aspect ratio a, ratio of major to minor axis. If a=1 have a circle or square; if a<1 have an ellipse or rectangle.
- Shape orientation theta0; measures angle between major axis and x axis. Irrelevant if a=1.
- Shape azimuthal range theta1, theta2, interval type. The default is theta1=0 and theta2=360 deg. Any other value gives you a pie or sector (for shape type elliptical; shape type rectangular may not support sectors).

## 4.11 Table Data Cell

A Data Cell is associated with a Data Descriptor and contains one set of Elements for that Data Descriptor. The number of elements in the cell is equal to the number of elements in the array specification for the data descriptor; in particular, if there is no array specification (data quantity array dimensionality equal to zero) there is exactly one element in the cell. The elements in the cell can be accessed via pixel number or element number as discussed in the section on array specifications and axes.

### 4.12 Table Row

In a Table Data section, there is some specified number $r$ of Table Rows. Each Row may be thought of as containing one Data Cell for each of the Column Data Descriptors. More precisely, there is one Data Cell associated with each combination of row and column.

# 5 Data Subspace

## 5.1 Introduction

What distinguishes a photon event list from a table in an ordinary database? The rows of the event list represent individual, asynchronous events. They cannot be interpreted without knowing the filter through which those events were selected. Suppose we detect photons only between times 100 and 200. Is this because the source flared during that time, or because the satellite was only looking during that time period? To be more precise, if you just have an ordinary table of rows, what you are missing is the information about what rows would NOT have been allowed in the table - in the photon event list case, which events would NOT have been detected. We are then led to the concept of the data subspace: in the space of all possible data, what subspace is being sampled by the current table?

This idea is closely connected with the idea of filtering. The data subspace is simply the filter that has been applied to the data. However, we're not just talking about user filters applied during processing, but also implicit filters applied by the act of observation at a particular time with a particular instrument. If the user then filters the data further, the new data subspace is simply the intersection of the filter with the old subspace.

If two datasets are merged, the new data subspace is the union of the old ones. In this case, however, we lose some information: the data subspace paradigm doesn't retain information about which of the original subspaces a particular row belonged to. This is the usual problem with binning data together, which we can illustrate with a familiar example: combining two pulse height spectra. Suppose we have two event lists E1 and E2 with the following data, representing events from two different ACIS chips which are distinguished by different ranges of detector position DETPOS:

```
E1 subspace: DETPOS=[0:1024,0:1024]
E1 table:
  DETPOS  PHA  TIME
  100 245  8   4922.2
  231 928  17  4812.5
  ....

E2 subspace: DETPOS=[1024:2048,0:1024]
E2 table:
```

```
  DETPOS PHA TIME
 1241 621 22 4924.3
 1782 212  7 4092.2
 ...
```

If we extract two PHA histograms P1 and P2, retaining only pulse heights from 2 to 100 and selecting a region near the boundary of the chips where we think there is a source, we get:

```
P1 subspace: DETPOS=[1000:1024,800:825], PHA=[2:100]
P1 table:
 PHA  COUNTS
 2    0
 3    4
 ....
100   1

P2 subspace: DETPOS=[1024:1124,800:825], PHA=[2:100]
P2 table:
 PHA  COUNTS
 2    1
 3    2
...
```

If we then merge these two datasets to form P3, we get:

```
P3 subspace: DETPOS=[1000:1124,800:825], PHA=[2:100]
P3 table:
 PHA   COUNTS
 2      1
 3      6
 ....
```

A tool to build the XSPEC response matrix would then check the DETPOS region to see which chips were involved. In the case of P3, it would see that 20 percent of the region was on one chip and 80 percent on the other, and would average the two response matrices in that proportion. We have lost any information about which counts came from which chip. If instead we merge the lists E1 and E2 to form a new event list which retains the DETPOS column, and then filter on position and PHA but don't bin to make the histograms, we get E3:

```
E3 subspace: DETPOS=[1000:1124,800:825], PHA=[2:100]
E3 table:
```

```
 DETPOS   PHA
 1012 814  8
 1182 803 18
 ...
```

although the data subspace is the same as for P3, the information about which chip is involved for a given event is still available via the DETPOS value for the given event.

In general, any tabular data may have a data subspace which describes the range of data for which the table applies. The quantities in the data subspace are not necessarily the same as the quantities in the table itself - see the example of P3 above in which DETPOS is in the data subspace but not in the table.

## 5.2 Unions of subspaces

A more complicated case of merging subspaces is when we wish to use 'incompatible' filters. For example, perhaps the second chip has unreliable data in PHA channels 2 to 10, so we want to apply a different PHA filter to it. We filter E1 and E2 with different filters and then merge them to make E4:

```
E4 subspace:
    Component 1: DETPOS=[1000:1024,800:825], PHA=[2:100]
    Component 2: DETPOS=[1024:1124,800:825], PHA=[11:100]
E4 table:
 DETPOS   PHA
 1012 814  8
 1182 803 18
 ...
```

When two filters (subspaces) are unioned (logical OR), we describe them as different 'components' of the subspace.

What if the different filters involve filtering on entirely different quantities? Consider the case when E1 is filtered on PHA and E2 is filtered on TIME.

```
E5 subspace:
    Component 1: DETPOS=[1000:1024,800:825], PHA=[2:100]
    Component 2: DETPOS=[1024:1124,800:825], TIME=[4823.2:4890.1),[5012.4,5100.0)
E5 table:
 DETPOS   PHA TIME
 1012 814  8  4902.54
 1182 803 18  4823.80
 ...
```

To simplify the treatment, we note that we can make the quantities involved in the two components the same by adding the trivial filters TIME=$[-\infty : \infty]$ and PHA=$[-\infty : \infty]$ to components 1 and 2 respectively. Doing this lets us store a single list of the quantities involved in a data subspace, instead of requiring us to maintain separate lists for each component.

## 5.3 General definition

1. A Data Subspace (DSS) D consists of $DC = 0+$ Data Subspace Components (DSS Components) $C(i), i = 1, DC$ and a list of $DA = 0+$ Data Subspace Data Descriptors or Data Subspace Axis Groups $A(j), j = 1, DA$. (*Note: The notation* $n_C = 0+$ *means that there are zero or more of the entities in question, and that the number of entities will be denoted by* $DA$.) There is usually only one DSS Component in a DSS, i.e. DC=1. The name Axis Group reflects the fact that the data subspace could be represented by an array with those axis groups (although the pixel values of that array are not defined).

2. A Data Subspace Data Descriptor or Data Subspace Axis Group is a named object which has the same properties as the generic Data Descriptor defined above, particularly including a name and a dimensionality. An example of a data subspace axis group might be TIME, or POSITION. However, a Data Subspace Data Descriptor may not have associated array Axis Group Quantities, or array Axis Group Coordinate Quantities. Further, it must have array dimensionality 1. An important distinction between the DD for Table Data and the DD for a Data Subspace is that the array dimension $n_1$ is to be interpreted as the maximum dimension for any data cell, rather than the actual dimension for each data cell (see below). However, the Data Subspace Data Descriptor is allowed to have a Data Coordinate Transform and a Data Coordinate Quantity.

3. A Data Subspace Component $C(i)$ consists of $DA$ DSS Data Cells $RV(i, j)$, one for each axis group of the parent data subspace.

4. The Data Cells of a data subspace component consist of $n_R = 0+$ Region Elements $R(i, j, k), k = 1, n_R(i, j)$. An example of such a Data Cell is a set of Good Time Intervals, or a spatial mask consisting of several components. The different Data Cells corresponding to different DSS Components may have different values of $n_R$, unlike the Data Cells for different rows of a Table Data section which must all have the same array sizes. Since there is usually only one DSS component, this doesn't usually matter.

5. A Data Cell may be defined implicitly as a World Coordinate Data Cell. For instance, if the Data Subspace Axis Group is pixel sky coordinate position SKYPOS (X,Y), and this has a Data Coordinate Quantity EQPOS (RA,DEC) related to it by a Data Coordinate Transform, then we may express the Data Cell as a set of region elements attached to EQPOS (the Data Coordinate Quantity) rather than SKYPOS (the Data Quantity) - say, a circle expressed as

‘(c 14:04:11 -00:23:12 6.2’)’, i.e. a 6.2 arcmin circle around the specified sexagesimal RA and Dec, instead of ‘(c 4212.2 5123.2 42.1)’ in pixels. I haven’t included this explictly in the diagrams; in the FITS implementation I have suggested parallel keywords DSn and DSCn for regions expressed in the pixel and world systems respectively.

6. A Region Element $R(i,j,k)$ in a data subspace data cell is a range element if the dimensionality of the corresponding Data Subspace Axis Group is 1, and is a 2D Region Element in the dimensionality of the corresponding Data Subspace Axis Group is 2.

   From a set-theory point of view,

   $$RV(i,j) = \cup_k R(i,j,k)$$

   and

   $$C(i) = \cap_j RV(i,j)$$

   and

   $$D = \cup_i C(i) = \bigcup_i \left( \bigcap_j (\cup_k R(i,j,k)) \right)$$

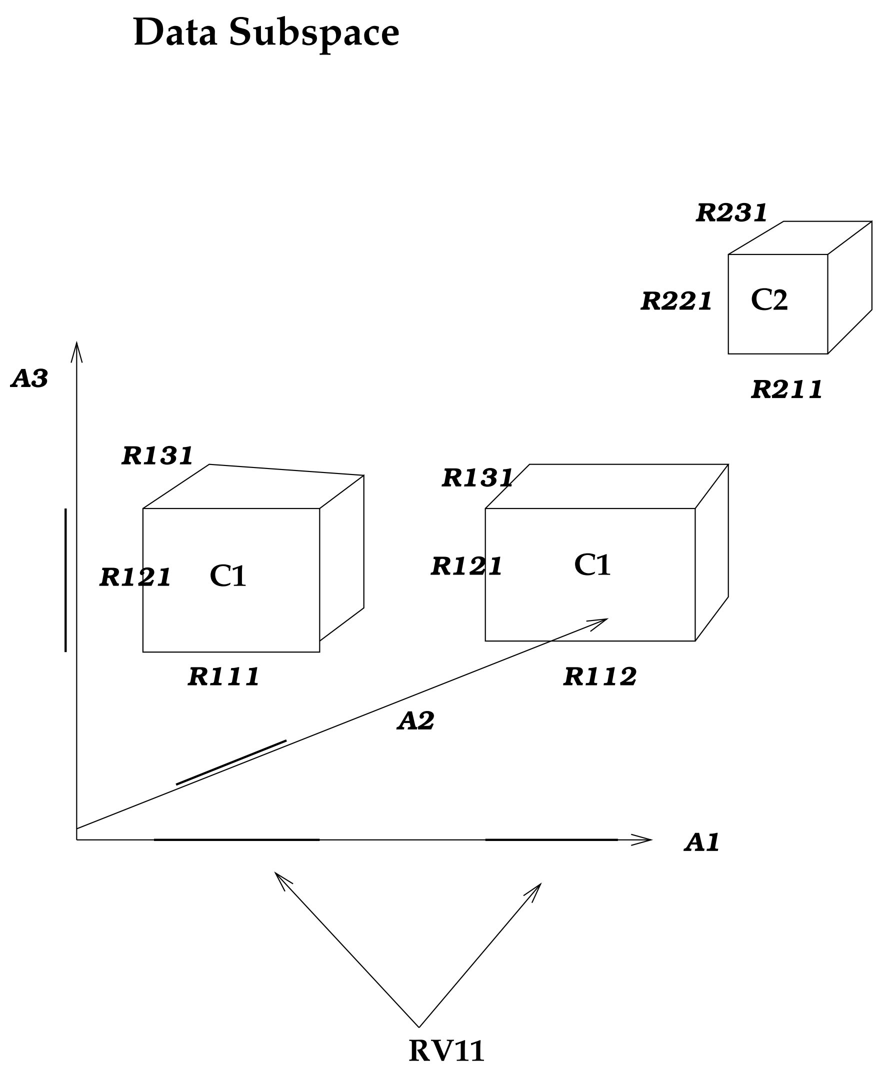


Figure 9: Illustration of a data subspace.

7. A data point P, consisting of values $V_j$, $j = 1, DA$, is said to be 'in' the data subspace if it is in any one of the components. It is in a component if it is in all of that component's data cells. It is in a data cell if it is in any of the data cell's region elements.

8. The intersection of two data subspaces $D^1$ and $D^2$ is calculated as follows: First extend the lists of axis groups of each subspace to be the same. Then

$$D^1 \cap D^2 = \bigcup_i \left( \bigcap_j (\cup_k R^1(i,j,k)) \right) \cap \bigcup_m \left( \bigcap_j (\cup_n R^2(m,j,n)) \right)$$

or

$$D^1 \cap D^2 = \bigcup_i \bigcup_m \left( \bigcap_j \left( \cup_k \cup_n R^1(i,j,k) R^2(m,j,n) \right) \right)$$

The case of a single point can be understood as a special case of this. Consider the value components $V_j$ as closed zero-length ranges $[V_j : V_j]$; then P is a data subspace with one component and $R(i,j,k) = [V_j : V_j]$. The above formula tells us to intersect each component with the corresponding range.

Examples of intersection of data subspaces: First, let's take the point case. Let the data subspace be that of E5 above:

```
Component 1: DETPOS=[1000:1024,800:825], PHA=[2:100], TIME=[:]
Component 2: DETPOS=[1024:1124,800:825], PHA=[:], TIME=[4823.2:4890.1),[5012.4,5100.0)
```

Then let P be the point (DETPOS,PHA,TIME)=((1100,812),200,5050). We have:

```
A(1) = DETPOS
A(2) = PHA
A(3) = TIME
R(1,1,1) = Box 1000:1024, 800:825
R(1,2,1) = [2:100]
R(1,3,1) = [:]
R(2,1,1) = Box 1024:1124, 800:825
R(2,2,1) = [:]
R(2,3,1) = [4823.2,4890.1)
R(2,3,2) = [5012.4,5100.0)
V(1) = (1100,812)
V(2) = 200
V(3) = 5050
```

So first we intersect P with component 1. The intersection is null, since V(1) has no overlap with R(1,1,1) and V(2) has no overlap with R(1,2,1). Next we intersect with component 2. The intersection of V(1) with R(2,1,1) is V(1) itself; similarly for V(2). V(3) is outside R(2,3,1) but inside R(2,3,2) and thus inside their union as required. So the intersection of P with component 2 of the subspace is P itself. Thus, P is inside the subspace.

Now let's take the intersection of two filters. Let the second space be a simple time filter with two intervals, TIME=[4000:4800],[6000:7000]. To do the intersection we add the missing axes:

```
R(1,1,1)=[:,:]
R(1,2,1)=[:]
R(1,3,1)=[4000:4800]
R(1,3,2)=[6000:7000]
```

Then evaluating the intersection equation gives the expected result:

```
A(1) = DETPOS
A(2) = PHA
A(3) = TIME
R(1,1,1) = Box 1000:1024, 800:825
R(1,2,1) = [2:100]
R(1,3,1) = [4000:4900]
R(1,3,2) = [6000:7000]
R(2,1,1) = Box 1024:1124, 800:825
R(2,2,1) = [:]
R(2,3,1) = [4823.2,4800.0]
```

Note that the second element of the TIME region vector in component 2 has disappeared, since it had no overlap with the new filter. The interval type of the first element has changed, it is now a closed interval. If the filter had been [4000:4700], the entire second component would have been removed.

# 6 Header

The ASC Table Header contains metadata analogous to FITS header keywords. We allow ASC header attributes to have all the properties of a Quantity, in contrast to FITS header keywords which do not have the full properties of a FITS table column.

## 6.1 Attribute Data Descriptor

An Attribute Data Descriptor has the same structure as a Table Column Data Descriptor. However, in an initial implementation we will not support array dimensionality greater than 1 or axis group quantities (cf. DSS Data Descriptor).

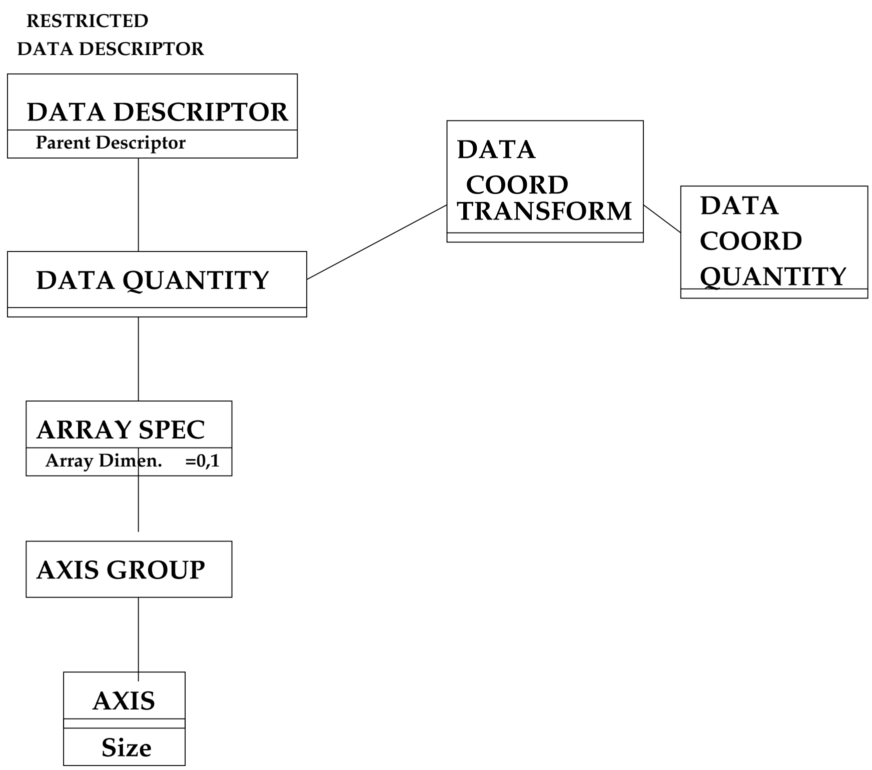


Figure 10: Data Model 6: Restricted Data Descriptor, used in Attribute Data Descriptor and DSS Data Descriptor. Does not support full Array/Image functionality.

Attributes may be related to other 'parent' data descriptors, either other attributes or columns or data subspace axis descriptors. Attributes that are related to columns are called column attributes. Attributes that are related to data subspace axes are called data subspace attributes. All other attributes are table attributes. A generic FITS header keyword is a table attribute; the idea of tying header keywords to particular columns is new. A table attribute which is related to another table attribute may be considered as part of a group (equivalence class) of table attributes; this allows us to group header keywords and refer to them by groups rather than individually.

### 6.2 Attribute Data Cell and Elements

The Attribute data cell and elements are the same as those for Table Data.

## 7 ASC Image

### 7.1 Images and Tables

An ASC Image is an ASC Table with a single Table Column Data Descriptor whose array dimensionality $n > 0$ and with a single Row. Special access routines will be provided for ASC Images. Any single array Data Cell in a table may also be treated as an ASC Image; to instantiate it as such an image, copy it to a new ASC table together with the DSS, the Table Attributes, as well as the Data Descriptor and Column Attributes for its own Column, but discarding the other rows for the column and discarding the other column data descriptors, cells, and column attributes.

I illustrate the structure of an ASC Image in the accompanying diagram; note that from the OO point of view this is just an instance of the ASC Table, not a separate model.

## 8 Case studies and examples

### 8.1 FITS case study: PSPC off axis histogram file

An ASCII dump of a Rosat PSPC FITS file for the off axis histogram for an extracted source is reproduced below; I then interpret it in terms of the data model.

```
XTENSION= 'BINTABLE'           / binary table extension
BITPIX  =                    8 / 8-bit bytes
NAXIS   =                    2 / 2-dimensional binary table
NAXIS1  =                    8 / width of table in bytes
NAXIS2  =                   14 / number of rows in table
PCOUNT  =                    0 / size of special data area
GCOUNT  =                    1 / one data group (required keyword)
TFIELDS =                    2 / number of fields in each row
```

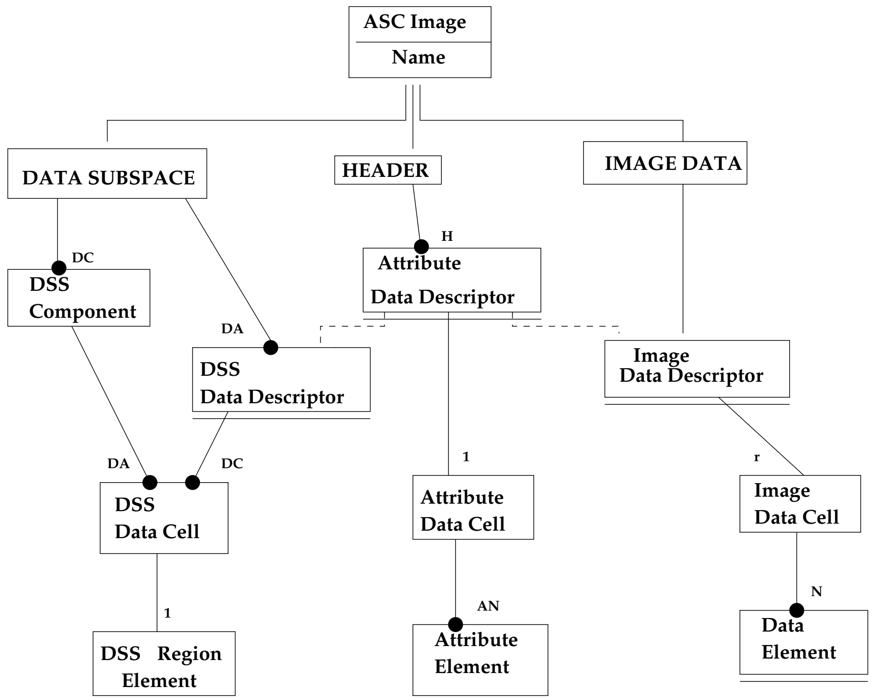


Figure 11: Data Model 7: ASC Image Model, identical to Table Model but without Table Row and with only one Column Descriptor (Image Descriptor).

```
TTYPE1  = 'OFF_AX_RAD'          / Off-axis grid point for histogram bin (arcmin)
TFORM1  = '1E      '            / data format of the field: 4-byte REAL
TUNIT1  = 'arcmin  '            / physical unit of field
TTYPE2  = 'FRAC_TIME'           / Fraction of time spent by source in bin
TFORM2  = '1E      '            / data format of the field: 4-byte REAL
TUNIT2  = 'NONE    '            / physical unit of field
EXTNAME = 'OAH005  '            / Detect extension-asp histogram for given source
CONTENT = 'SOURCE  '            / data content of file
ORIGIN  = 'USRSDC  '            / origin of processed data
DATE    = '13/07/94'            / FITS creation date (DD/MM/YY)
TELESCOP= 'ROSAT   '            / mission name
INSTRUME= 'PSPCC   '            / instrument name
OBS_MODE= 'POINTING'            / obs mode: POINTING,SLEW, OR SCAN
IRAFNAME= 'rp110590n00_oah005.tab' / IRAF file name
MJDREFI =                 48043 / MJD integer  SC clock start
MJDREFF =  8.79745370370074E-01 / MJD fraction SC clock start
ZERODATE= '01/06/90'            / UT date of SC start (DD/MM/YY)
ZEROTIME= '21:06:50'            / UT time of SC start (HH:MM:SS)
```

```
RDF_VERS= '2.9     '           / Rationalized Data Format release version number
RDF_DATE= '13-JUL-1994'        / Rationalized Data Format release date
PROC_SYS= 'SASS7_2_0'          / Processing system
PROCDATE= '2-JUN-1994 11:20:34' / SASS SEQ processing start date
REVISION=                    2 / Revision number of processed data
FILTER  = 'NONE    '           / filter id: NONE OR BORON
OBJECT  = 'XRT/PSPC PSF AR LAC' / name of object
RA_NOM  =         3.320239E+02 / nominal RA (deg)
DEC_NOM =         4.551389E+01 / nominal DEC (deg)
ROLL_NOM=        -1.349511E+02 / nominal ROLL (deg CCW  North)
EQUINOX =         2.000000E+03 / equinox
OBS_ID  = 'CA110590P.N10'      / observation ID
ROR_NUM =               110590 / ROR number
OBSERVER= 'MPE, ROSAT-TEAM'    / PI name
SETUPID = 'NOMINAL '           / Instrument setup
DATE-OBS= '20/06/90'           / UT date of obs start (DD/MM/YY)
TIME-OBS= '11:24:43.000'       / UT time of obs start (HH:MM:SS)
DATE_END= '20/06/90'           / UT date of obs end   (DD/MM/YY)
TIME_END= '13:07:12.000'       / UT time of obs end   (HH:MM:SS)
MJD-OBS =         4.806248E+04 / MJD of seq start
SCSEQBEG=              1606667 / SC seq start(sec)
SCSEQEND=              1612816 / SC seq end  (sec)
NUM_OBIS=                    2 / Number of obs intervals (OBIs)
LIVETIME=         1.884999E+03 / Live time
DTCOR   =         9.602644E-01 / Dead time correction factor
ONTIME  =         1.963000E+03 / On time
MPLSX_ID=                    5 / Source number from merged source list (MPLSX)
EFFAREA =           1.0000E+00 / Effective area scaling factor
QUALITY =                    0 / Quality of data (0 = good data)
RADECSYS= 'FK5     '           / WCS for this file
OFFAX   =         1.478056E+01 / Off-axis angle of source in arcmin
COMMENT
COMMENT   The following keywords are required in order to conform
COMMENT   to the Office of Guest Investigator Programs standard:
COMMENT
AREASCAL=           1.0000E+00 / Area scaling factor
BACKFILE= 'NONE    '           / No background file
BACKSCAL=           1.0000E+00 / Background scaling factor
CORRFILE= 'NONE    '           / No correction file
CORRSCAL=           1.0000E+00 / Correction file scaling factor
RESPFILE= 'NONE    '           / BLDRSP response file name (default)
ANCRFILE= 'NONE    '           / BLDRSP ancillary response file name (default)
```

```
XFLT0001= '        '           / Required keyword
PHAVERSN= '1992A   '           / Version # of OGIP file specification
POISSERR=                    T / Poissonian errors appropriate
SYS_ERR =                  0.0 / No systematic error
CHANTYPE= 'PI      '           / Gain-corrected channels used
DETCHANS=                  256 / Total number of PHA channels available
COMMENT
COMMENT   End required OGIP keywords
COMMENT
COMMENT   This extension contains the off-axis histogram
COMMENT   for the source given in the header.
HISTORY
HISTORY   SASS file used: SPCBF.SEQ
HISTORY
HISTORY   Correspondence with SASS variables:
HISTORY
HISTORY   OFF_AX_RAD = OFF_SPB
HISTORY   FRAC_TIME  = OHS_SBP

OFF_AX_RAD   FRAC_TIME
0            0
5            0
10           2.17989E-04
15           0.73768
20           0.2621
25           0
30           0
35           0
40           0
45           0
50           0
55           0
57.5         0
60           0
```

What does this file contain? There's a lot of stuff all mixed together. We might describe it as follows:

```
Table OAH005( 2 cols, 14 rows)

Colname  OFF_AX_RAD   FRAC_TIME
Datatype Real(4)      Real(4)
Unit  none            none
Elt type   V          V
```

```
Elt dim    1          1
Disp   none           none
Desc   'Off-axis grid point for histogram bin (arcmin)'
       'Fraction of time spent by source in bin'
Component name (same as colname)
Array dim  0          0


Cells: 1 element per cell
Elements: 1 value per element (type V, dimension 1)
Values:
         0            0
         5            0
         10           2.17989E-04
         15           0.73768
         20           0.2621
         25           0
         30           0
         35           0
         40           0
         45           0
         50           0
         55           0
         57.5         0
         60           0


Data Subspace( 4 axes )

 TIME          [1606667:1612816)
   Coordinate: Origin = 0
               Value  = JD 2448044.379745370370074 d
               Delta = 1
               Unit = s
               Comment SC seq start(sec)
               Correction Factor  0.96026   (DTCOR)
 RA/DEC        Region not given (would be nice!)
    2D Coordinate: Origin = not given
               Value  = J2000 (332.0239, +45.51389)

               Delta = not given
               Unit = deg
```

```
The following data subspace axes are not explicitly present in the file:
 OFF_AX_RAD    [0:60]
               Unit = arcmin
               Comment Off-axis grid point for histogram bin
 FRAC_TIME     [0:1]
               Unit = none
               Comment Fraction of time spent by source in bin



The following header cards from the file are not retained
in our 'model' version as header cards per se because
they contain information about the structure of the
file or the attributes of its data axes:

\small

\begin{verbatim}
 Cards from FITS standards, mapped to table structure:
XTENSION= 'BINTABLE'           / binary table extension
BITPIX  =                    8 / 8-bit bytes
NAXIS   =                    2 / 2-dimensional binary table
NAXIS1  =                    8 / width of table in bytes
NAXIS2  =                   14 / number of rows in table
PCOUNT  =                    0 / size of special data area
GCOUNT  =                    1 / one data group (required keyword)
TFIELDS =                    2 / number of fields in each row
TTYPE1  = 'OFF_AX_RAD'         / Off-axis grid point for histogram bin (arcmin)
TFORM1  = '1E      '           / data format of the field: 4-byte REAL
TUNIT1  = 'arcmin  '           / physical unit of field
TTYPE2  = 'FRAC_TIME'          / Fraction of time spent by source in bin
TFORM2  = '1E      '           / data format of the field: 4-byte REAL
TUNIT2  = 'NONE    '           / physical unit of field
EXTNAME = 'OAH005  '           / Detect extension-asp histogram for given source
```

Cards from OGIP rules, mapped to subspace and coordinate info:

```
MJDREFI =                48043 / MJD integer  SC clock start
MJDREFF =  8.79745370370074E-01 / MJD fraction SC clock start
ZERODATE= '01/06/90'           / UT date of SC start (DD/MM/YY)
ZEROTIME= '21:06:50'           / UT time of SC start (HH:MM:SS)
RA_NOM  =         3.320239E+02 / nominal RA (deg)
```

```
DEC_NOM =         4.551389E+01 / nominal DEC (deg)
ROLL_NOM=        -1.349511E+02 / nominal ROLL (deg CCW  North)
EQUINOX =         2.000000E+03 / equinox
DATE-OBS= '20/06/90'           / UT date of obs start (DD/MM/YY)
TIME-OBS= '11:24:43.000'       / UT time of obs start (HH:MM:SS)
DATE_END= '20/06/90'           / UT date of obs end   (DD/MM/YY)
TIME_END= '13:07:12.000'       / UT time of obs end   (HH:MM:SS)
MJD-OBS =         4.806248E+04 / MJD of seq start
SCSEQBEG=              1606667 / SC seq start(sec)
SCSEQEND=              1612816 / SC seq end  (sec)
LIVETIME=         1.884999E+03 / Live time
DTCOR   =         9.602644E-01 / Dead time correction factor
ONTIME  =         1.963000E+03 / On time
```

When writing this file back out, all of the above cards would be generated automatically by the FITS writing layer; there's no need for any of the software beyond the IO layer to ever deal with them.

The remaining header cards come in a number of groups, which we can't deduce from the present structure of the file:

```
Ungrouped header cards
 OFFAX         14.78056
               Unit arcmin
               Comment Nominal off-axis angle of source

Header group PROCESSING
 CONTENT = 'SOURCE  '           / data content of file
 ORIGIN  = 'USRSDC  '           / origin of processed data
 DATE    = '13/07/94'           / FITS creation date (DD/MM/YY)
 IRAFNAME= 'rp110590n00_oah005.tab' / IRAF file name
 RDF_VERS= '2.9     '           / Rationalized Data Format release version number
 RDF_DATE= '13-JUL-1994'        / Rationalized Data Format release date
 PROC_SYS= 'SASS7_2_0'          / Processing system
 PROCDATE= '2-JUN-1994 11:20:34' / SASS SEQ processing start date
 REVISION=                    2 / Revision number of processed data

Header group OBSERVATION_DETAILS
 TELESCOP= 'ROSAT   '           / mission name
 INSTRUME= 'PSPCC   '           / instrument name
 OBS_MODE= 'POINTING'           / obs mode: POINTING,SLEW, OR SCAN
 FILTER  = 'NONE    '           / filter id: NONE OR BORON
 OBJECT  = 'XRT/PSPC PSF AR LAC' / name of object
 OBS_ID  = 'CA110590P.N10'      / observation ID
```

```
 OBSERVER= 'MPE, ROSAT-TEAM'    / PI name
 NUM_OBIS=                    2 / Number of obs intervals (OBIs)
 ROLL_NOM=    -134.95

Header group ROSAT_SPECIFIC
 ROR_NUM =               110590 / ROR number
 SETUPID = 'NOMINAL '           / Instrument setup
 MPLSX_ID=                    5 / Source number from merged source list (MPLSX)
 QUALITY =                    0 / Quality of data (0 = good data)

Header group OGIP_COMPAT  / These keywords may be ignored by our software
 EFFAREA =           1.0000E+00 / Effective area scaling factor
 COMMENT
 COMMENT   The following keywords are required in order to conform
 COMMENT   to the Office of Guest Investigator Programs standard:
 COMMENT
 AREASCAL=           1.0000E+00 / Area scaling factor
 BACKFILE= 'NONE    '           / No background file
 BACKSCAL=           1.0000E+00 / Background scaling factor
 CORRFILE= 'NONE    '           / No correction file
 CORRSCAL=           1.0000E+00 / Correction file scaling factor
 RESPFILE= 'NONE    '           / BLDRSP response file name (default)
 ANCRFILE= 'NONE    '           / BLDRSP ancillary response file name (default)
 XFLT0001= '        '           / Required keyword
 PHAVERSN= '1992A   '           / Version # of OGIP file specification
 POISSERR=                    T / Poissonian errors appropriate
 SYS_ERR =                  0.0 / No systematic error
 CHANTYPE= 'PI      '           / Gain-corrected channels used
 DETCHANS=                  256 / Total number of PHA channels available
 COMMENT
 COMMENT   End required OGIP keywords
 COMMENT

Header group COMMENTS
 COMMENT   This extension contains the off-axis histogram
 COMMENT   for the source given in the header.
 HISTORY
 HISTORY   SASS file used: SPCBF.SEQ
 HISTORY
 HISTORY   Correspondence with SASS variables:
 HISTORY
 HISTORY   OFF_AX_RAD = OFF_SPB
```

```
HISTORY   FRAC_TIME  = OHS_SBP
```

How would we redesign this file to take more advantage of the data model while remaining compatible with software that expects the old format? While I do not expect that we will be writing software to regenerate PSPC standard data products in this way, it's a useful exercise to show what is needed to add the extra structure.

- We add comments to denote Header Groups, grouping the table attributes. This could be used by browsers to organize the user's view of the data. It would be nice for software to be able to use such header groups, but there is a risk that some FITS readers will mangle the order of the header keywords, mixing up the group memberships. I still feel that it's an enhancement worth having, with the warning to users that if they pass the files through other software they may lose that information.

- The other way of making header groups is to explicitly add named cards. This is comparatively inefficient but may be the way to go when it's important that the linkage be robust. This is illustrated with the DAREL keywords for OFFAX and ROLL_NOM.

- The dataset is actually binned data; the OFF_AX_RAD column contains bins which for some perverse reason are uneven in size near the ends. I could have defined a special element type to denote bins where the boundaries are deduced to be half way to the next entry, but this would require the software to handle more than one row at a time. I prefer to accept the overhead of the extra two columns COL1_LO and COL1_HI, turning OFF_AX_RAD into a column of element type T (two sided uncertainty).

- We will store the extraction region in the data subspace header. The information includes the region specification in sky pixel coordinates and the transformation from sky pixel coordinates to RA and Dec, the latter being copied from the original file. This gives us a more logical place to put the info now stored in RA_NOM and DEC_NOM. If we had the region specification in RA and Dec instead of pixels, we would store it in keyword DSC1 instead of DS1.

- The preferred columns are OFF AX RAD and TIME; but we don't need to include PREF1 and PREF2 keywords since these are the only two columns at the data model level and they are in the correct order.

```
XTENSION= 'BINTABLE'           / binary table extension
BITPIX  =                    8 / 8-bit bytes
NAXIS   =                    2 / 2-dimensional binary table
NAXIS1  =                   16 / width of table in bytes
NAXIS2  =                   14 / number of rows in table
PCOUNT  =                    0 / size of special data area
```

```
GCOUNT  =                    1 / one data group (required keyword)
TFIELDS =                    4 / number of fields in each row
TTYPE1  = 'OFF_AX_RAD'         / Off Axis Radius
TFORM1  = '1E      '           / data format of the field: 4-byte REAL
TUNIT1  = 'arcmin  '           / physical unit of field
TTYPE2  = 'COL1_LO '           / Lower Uncertainty
TFORM2  = '1E      '           / 4 byte real
TUNIT2  = 'arcmin  '           /
TTYPE3  = 'COL1_HI '           / Upper Uncertainty
TFORM3  = '1E      '           / 4 byte real
TUNIT3  = 'arcmin  '           /
TTYPE4  = 'FRAC_TIME'          / Fractional Exposure Time
TFORM4  = '1E      '           / data format of the field: 4-byte REAL
TUNIT4  = '        '           / physical unit of field
EXTNAME = 'OAH005  '           / Off Axis Histogram
TDISP1  = 'F8.2    '           / Format to display OFF AX RAD
TDISP4  = 'F8.6    '           / Format to display FRAC TIME
TLMIN1  =                0.0   / Valid range for columns
TLMAX1  =               60.0   /
TLMIN4  =                0.0   /
TLMAX4  =                1.0   /
COMMENT
COMMENT ASC Table Keywords
COMMENT
DCFIELDS=                    2 / Number of logical columns
DCETYP1 = 'T       '           / Two sided uncertainty
DCITYP1 = '[)      '           / Interval type
COMMENT
COMMENT ASC Data Subspace Keywords
COMMENT
DSNAXIS =                    1 / Number of data subspace axes
DSNAM1  = 'SKYPOS  '           / Sky pixel position
DSDIM1  =                    2 / Dimension of DSNAM1
DSTYP1  = 'X       '           / First component of DSNAM1
DSTYP2  = 'Y       '           / Second component of DSNAM1
DSUNIT1 = 'pixel   '           /
DSCNAM1 = 'EQPOS   '           / Coordinate system on DSNAM1
DSCTYP1 = 'RA---TAN'           / Transform for axis 1
DSCTYP2 = 'DEC--TAN'           / Transform for axis 2
DSCUNI1 = 'deg     '           /
DSCRVL1 =             332.0239 / Reference RA value (RA\_NOM)
DSCRVL2 =              45.5138 / Reference Dec value (DEC\_NOM)
```

```
DSCRPX1 =           4096.5000  / Reference X value
DSCRPX2 =           4096.5000  / Reference Y value
DSCDLT1 =             -0.0124  / Deg per pixel
DSCDLR2 =              0.0124  / Deg per pixel
DS1     = 'c 4087.3 4012.3 43.2' / Extraction region in X,Y coords
DSTYP3  = 'TIME    '             / Mission time
DSUNIT3 = 's       '             /
DS2L1   =         1606667.0      / Start time
DS2U1   =         1612816.0      / Stop time
DSITYP3 = '[)      '             / Interval type for TIME
COMMENT
COMMENT Alternative syntax for the above three keywords would be:
COMMENT    DS2 = '[SCSEQBEG:SCSEQEND)'
COMMENT
DSCTYP3 = 'DATE    '             / Calendar date
DSCDLT3 =        1.15741E-05     / Days per second
DSCRVL3 =   48043.879745370370074 / MJD of SC clock start
DSCRPX3 =               0.0        / SC clock start
DSCUNI3 = 'd       '               /
DSTYP4  = 'OFF_AX_RAD'             / Range defaults to TLMIN1/TLMAX1
DSTYP5  = 'FRAC_TIME '             /
COMMENT
COMMENT ASC Table Attributes
COMMENT
COMMENT We only need to use explict DANAMn keywords when we
COMMENT want to add extra information to a keyword.
COMMENT

DANAM1  =  'OFFAX'             / Attribute
OFFAX   =         1.478056E+01 / Off-axis angle of source in arcmin
DAUNI1  =  'arcmin'            / Unit of DANAM1
DAREL1  =  'OFF_AX_RAD'        / Keyword OFFAX is bound to column OFF AX RAD

DANAM2  =  'ROLL_NOM'          / Attribute
ROLL_NOM=        -1.349511E+02 / nominal ROLL (deg CCW  North)
DAUNI2  =  'deg'               /
DAREL2  =  'SKYPOS'            / ROLL_NOM bound to DSS axis SKYPOS

DANAM3  =  'SRC_OFF_AX_RAD'    / Same as OFFAX,
DAVAL3  =         1.478056E+01 / but illustrating a name longer than 8 chars
DAUNI3  =  'arcmin '           /
```

```
DANAM4  =  'ONTIME '           / Denote the fact that the keywords named
DAREL4  =  'TIME   '           / are tied to the TIME information, so if that
DANAM5  =  'DTCOR  '           / becomes invalid so do these.
DAREL5  =  'TIME   '           / Debatable whether we would actually bother
DANAM6  =  'LIVETIME'          / to add these linkages in this case.
DAREL6  =  'TIME   '           /

COMMENT
COMMENT Header Group PROCESSING
COMMENT
CONTENT = 'SOURCE  '           / data content of file
ORIGIN  = 'USRSDC  '           / origin of processed data
DATE    = '13/07/94'           / FITS creation date (DD/MM/YY)
IRAFNAME= 'rp110590n00_oah005.tab' / IRAF file name
RDF_VERS= '2.9     '           / Rationalized Data Format release version number
RDF_DATE= '13-JUL-1994'        / Rationalized Data Format release date
PROC_SYS= 'SASS7_2_0'          / Processing system
PROCDATE= '2-JUN-1994 11:20:34' / SASS SEQ processing start date
REVISION=                    2 / Revision number of processed data
COMMENT
COMMENT Header Group Observation Details
COMMENT
TELESCOP= 'ROSAT   '           / mission name
INSTRUME= 'PSPCC   '           / instrument name
OBS_MODE= 'POINTING'           / obs mode: POINTING,SLEW, OR SCAN
FILTER  = 'NONE    '           / filter id: NONE OR BORON
OBJECT  = 'XRT/PSPC PSF AR LAC' / name of object
OBS_ID  = 'CA110590P.N10'      / observation ID
OBSERVER= 'MPE, ROSAT-TEAM'    / PI name


COMMENT
COMMENT Header Group ROSAT Specific
COMMENT

ROR_NUM =               110590 / ROR number
SETUPID = 'NOMINAL '           / Instrument setup


COMMENT
COMMENT Header Group HEASARC Position Keywords
COMMENT
```

```
RA_NOM  =          3.320239E+02 / nominal RA (deg)
DEC_NOM =          4.551389E+01 / nominal DEC (deg)
EQUINOX =          2.000000E+03 / equinox
RADECSYS= 'FK5     '            / WCS for this file

COMMENT
COMMENT Header Group HEASARC Timing Keywords
COMMENT
MJDREFI =                 48043 / MJD integer  SC clock start
MJDREFF =  8.79745370370074E-01 / MJD fraction SC clock start
ZERODATE= '01/06/90'            / UT date of SC start (DD/MM/YY)
ZEROTIME= '21:06:50'            / UT time of SC start (HH:MM:SS)
DATE-OBS= '20/06/90'            / UT date of obs start (DD/MM/YY)
TIME-OBS= '11:24:43.000'        / UT time of obs start (HH:MM:SS)
DATE_END= '20/06/90'            / UT date of obs end   (DD/MM/YY)
TIME_END= '13:07:12.000'        / UT time of obs end   (HH:MM:SS)
SCSEQBEG=               1606667 / SC seq start(sec)
SCSEQEND=               1612816 / SC seq end  (sec)
MJD-OBS =          4.806248E+04 / MJD of seq start
NUM_OBIS=                     2 / Number of obs intervals (OBIs)
LIVETIME=          1.884999E+03 / Live time
DTCOR   =          9.602644E-01 / Dead time correction factor
ONTIME  =          1.963000E+03 / On time
MPLSX_ID=                     5 / Source number from merged source list (MPLSX)
EFFAREA =            1.0000E+00 / Effective area scaling factor
COMMENT
COMMENT Header Group OGIP_COMPAT
COMMENT
COMMENT   The following keywords are required in order to conform
COMMENT   to the Office of Guest Investigator Programs standard:
COMMENT
AREASCAL=            1.0000E+00 / Area scaling factor
BACKFILE= 'NONE    '            / No background file
BACKSCAL=            1.0000E+00 / Background scaling factor
CORRFILE= 'NONE    '            / No correction file
CORRSCAL=            1.0000E+00 / Correction file scaling factor
RESPFILE= 'NONE    '            / BLDRSP response file name (default)
ANCRFILE= 'NONE    '            / BLDRSP ancillary response file name (default)
XFLT0001= '        '            / Required keyword
PHAVERSN= '1992A   '            / Version # of OGIP file specification
COMMENT Note that the error info given here applies to the counts errors
```

```
COMMENT which are in an entirely different table; so we don't
COMMENT attach them to the data model errors in this file.
POISSERR=                    T / Poissonian errors appropriate
SYS_ERR =                  0.0 / No systematic error
CHANTYPE= 'PI      '           / Gain-corrected channels used
DETCHANS=                  256 / Total number of PHA channels available
COMMENT
COMMENT   End required OGIP keywords
COMMENT
COMMENT Header Ungrouped
COMMENT


QUALITY =                    0 / Quality of data (0 = good data)
COMMENT
COMMENT   This extension contains the off-axis histogram
COMMENT   for the source given in the header.
HISTORY
HISTORY   SASS file used: SPCBF.SEQ
HISTORY
HISTORY   Correspondence with SASS variables:
HISTORY
HISTORY   OFF_AX_RAD = OFF_SPB
HISTORY   FRAC_TIME  = OHS_SBP
```

| OFF_AX_RAD | COL1_LO | COL1_HI | FRAC_TIME |
|---|---|---|---|
| 0 | 0 | 2.5 | 0 |
| 5 | 2.5 | 2.5 | 0 |
| 10 | 2.5 | 2.5 | 2.17989E-04 |
| 15 | 2.5 | 2.5 | 0.73768 |
| 20 | 2.5 | 2.5 | 0.2621 |
| 25 | 2.5 | 2.5 | 0 |
| 30 | 2.5 | 2.5 | 0 |
| 35 | 2.5 | 2.5 | 0 |
| 40 | 2.5 | 2.5 | 0 |
| 45 | 2.5 | 2.5 | 0 |
| 50 | 2.5 | 2.5 | 0 |
| 55 | 2.5 | 1.25 | 0 |
| 57.5 | 1.25 | 1.25 | 0 |
| 60 | 1.25 | 0 | 0 |

## 8.2 Case Study: Barycenter Correction Algorithm

We analysed the Barycenter Correction Algorithm to see how it would be laid out in terms of the data model.

The algorithm uses the following ASC Tables:

- **Event List:** this contains rows which we refer to as photons, and a set of columns which include at least **Pixel Position** and **Time**. The Pixel Position Column Data Descriptor has Data Quantity with default name Pixel Position and component names X and Y; it must be of element dimension 2. We will access it by element type V (Value). It must also have a Data Coordinate Quantity which contains the Equatorial Position (RA and Dec). The Time Data Descriptor may have a Data Coordinate Quantity giving the absolute Date.
- **Orbital Data:** This is a stack containing the names of spacecraft and pointers to their Ephemeris files.
- **Solar System Ephemeris:** This is a stack containing the names of planets and pointers to their Ephemeris files.
- **Ephemeris:** This is a table with the columns Time and 3-Vector-Position. The latter has element dimension 3 and component names X,Y,Z. The ephemeris table has a table attribute Mass, giving the mass of the orbiting body.

The algorithm is:

- Identify the spacecraft in use for this event list: this should be a table attribute of the event list.
- Find the corresponding spacecraft ephemeris from the orbital data stack.
- Open output table with same format as input event list but with extra column named BARY_TIME of dimension 1 and type U. Unit is seconds of mission time; coordinate system is copied from input column whose default name is TIME. Add comment to header describing the fact that BARY_TIME is the time of a different event (arrival of a photon at the barycenter) in the same coordinate system as TIME.
- For each row in the table, get the Pixel Position. Calculate the Equatorial Position using that Data Descriptor's Data Coordinate Transform.
- Calculate the 3-vector direction of the photon (the source vector )from the equatorial position.
- Get time from row of table (represents photon arrival time at spacecraft). (If the ephemerides are in JD rather than mission time, may need to also use this Data Descriptor's Data Coordinate Transform to get JD from time.) Also get time uncertainty if present.
- Interpolate in spacecraft ephemeris at the given time to return the spacecraft ephemeris position and uncertainty (an element of type U and dimension 3).

- For each entry (planet) in the solar system ephemeris stack, interpolate in the corresponding ephemeris and return the mass of the planet and the position (a value element of dimension 3) at the time.
- Calculate the solar system barycenter at the given time by taking the mass weighted mean of the planetary positions. Result is an element of type V and dimension 3.
- Calculate the barycenter to spacecraft vector and its uncertainty. Check that the units of barycenter and spacecraft positions are compatible and apply conversions if necessary.
- Calculate the scalar product of the spacecraft and source vectors and its uncertainty; scale to light travel time to obtain correction. Correction is an element of type U.
- Add this to photon time and combine uncertainty in quadrature. Result is barycenter corrected time (BARY_TIME).
- Copy input row to output, adding new column of BARY_TIME.
- Loop to next photon until complete.