Filtering Syntax in the ASC Data Model
SDS-6.0

Jonathan McDowell, Antonella Fruscione, Aneta Siemiginowska, Bill Joye

February 26, 1998

Contents

1 Introduction

1.1 Summary

This document defines the 'virtual table' or 'virtual datablock' syntax used by the ASC data model. The virtual table syntax allows a single character string to define one ASC datablock in terms of another, related one. Specifically, certain rows and columns of the 'base' datablock are selected to form the 'virtual' datablock. ASC data model I/O operations are performed on the virtual datablock; the calling program only has knowledge of the virtual datablock, with the base datablock visible only to the internals of the data model subroutine library. There are two kinds of virtual datablock: the virtual table and the virtual image (see the ASC Data Model design document for the definition of concepts used in this document). The distinction between a virtual table and virtual image specification is in the calling routine; the same syntax may specify either a table or an image depending on which routine is parsing it (which application the string is being fed to). In some cases an ASC library program may open a virtual table using a base image file, or a virtual image using a base table file.

The first sections of this document give examples of the syntax with some explanation, but are not comprehensive. A full definition of the syntax is given in the final section of the document, which should be treated as the definitive explanation.

1.2 Changes

Jul 22: Changed SELECT command to COLUMNS, since in database contexts the word SELECT implies a row selection and not a column selection.

1.3 Intent

The syntax is largely inspired by the IRAF/PROS filtering, regions, image section, and blocking capability. We have added intensity filtering from XSELECT and relational filtering and column selection from TTOOLS. From MIDAS we plan to add filter stacking capabilities but that isn't in the current version. We have attempted to retain as much as possible of the PROS filtering syntax to keep it familiar to users of that system. We have also attempted to integrate the regions syntax with it and to add new capabilities in a natural way. We've also tried to keep it compact.

The syntax consists of a string with qualifiers surrounded by square parentheses, thus

a[b][c][d][e]

Each group within square parentheses has a different meaning; we could have avoided some of these parentheses and replaced them with spaces, which would have been in keeping with our desire for compactness, but we feel that the risk of getting a valid but unintended string with a single mistype was too high, and the clear 'phrasing' makes the intent much more readable.

1.4 Implementation strategy

For the initial release of the data model, syntax requiring access to more than one dataset or datablock at a time will be ignored (see 'External Tables'). The functional filter capability defined below will not be supported.

2 Examples

Dataset names are case sensitive. All other names are case-insensitive for matching, but case-sensitive for output. (In other words, if you ask to read a quantity evENts, the software will successfully find the table EVENTS; but if you ask for an output table to be named evENts, that is exactly what it will be named.)

2.1 Simple filtering

The simplest case of a virtual datablock is the name of a dataset:

dataset_name

This defines the virtual datablock to be a complete base datablock, the first datablock in the given dataset.

For analogy with FTOOLS, we provide the following syntax to access the nth datablock in the dataset:

dataset_name[n]

where n is a non-negative integer. Example:

```
rh1012.fits[2]
```

accesses the 2nd ASC datablock in the file rh1012.fits. Note that in FTOOLS this syntax would refer to the 3rd Header Data Unit (HDU); in a FITS file consisting only of generic BINTABLEs, this would be the same as the 2nd ASC datablock (since the null primary HDU would be lumped with the first BINTABLE extension), but in more complicated cases the FITS HDU number and the ASC datablock number may not be simply related.

A friendlier syntax for accessing a specific datablock is:

dataset_name[block_name]

However, ASC datablocks whose names consist only of digits cannot be accessed in this way if the corresponding number is less than the number of datablocks in the dataset (the numeric syntax will win over the name syntax in this case; i.e., dataset_name[2] will look for the 2nd block and not for a block named "2".) We hope that no-one will name datablocks using low integers anyway.

We select rows by

dataset_name[name=min:max,min:max,min:max, name=min:max,min:max...]

for example:

```
rh1012.fits[pha=1:20,50:90,time=100:200,500:900]
```

The columns may also be accessed by column number: e.g.

```
rh1012.fits[#1=1:20,50:90,#5=100:200,500:900]
```

2.2 Column selection and renaming

We can select columns from a table datablock by

dataset_name[columns column-list]

for example

```
rh1012.fits[columns detpos,pha,time]
```

We can rename columns in the output table:

```
rh1012.fits[columns pi=pha,time]
```

which takes the old column called pha and renames it pi.

In the select qualifier, there are two special reserved column names: #all and #none, with synonyms * and -. Thus

```
rh1012.fits[events][columns -]
```

makes a copy of the header and data subspace of rh1012.fits[events], but with an empty table/image section. This is useful for making a filter file (see below). (Note: the data subspace records the selection history of the data, e.g. GTI, PHA filter, etc: see the data model design document).

To delete a column, use the ! operator:

```
rh1012.fits[columns !pha]
```

To add a new column onto the table at the end (rightmost position),

```
rh1012.fits[events][columns *,status]
rh1012.fits[events][columns #all,status]
```

adds a column called status. This isn't very useful as it stands, since we don't know what to put in status. It will default to a 4 byte integer column filled with zeroes. By combining this with the renaming syntax, you can copy a column:

```
rh1012.fits[events][columns *,pi=pha]
```

so now there is a pi column whose contents are identical to the pha column. A more useful application will come with the support for interpolation on external tables (see below).

Finally we can specify the desired name of the output table, although this may be overriden by some applications:

```
rh1012.fits[mytable][columns pha,time][newtable]
```

The required ordering of the qualifiers is intended to reflect a logical ordering of the operations: first an input block is chosen, then it is filtered on rows, then on columns, and finally it is sent to a named output block.

2.3 Filtering on an external filter

Often we may want to filter several datasets on the same set of conditions. In particular, we want to select rows matching the same conditions as we selected for another file. The syntax

```
rh1012.fits[events][@source3.pha]
```

uses the data subspace of the file source3.pha as its filter. It intersects that data subspace with the data subspace of rh1012.fits[events]. It doesn't care whether or not source3.pha is just a filter (a data subspace with no table data) or whether there is a lot of data hanging on the end; such data is just ignored.

We also want to be able to store a string filter expression in an ASCII file for use in this way. The best way to support this would be for our eventual ASCII file kernel to the data model to recognizes such expressions as a data subspace definition.

2.4 Filtering on row number

Sometimes we want to filter explicitly on the row number of the original table. To support this we introduce a 'fake column' called #row which always has the value of the row number (prior to any other filtering). For example:

```
rh1012.fits[events][#row=200,300,pha=10:20]
```

would select only rows 200 to 300 of the table, and then further filter on only those rows which have pha from 10 to 20.

2.5 Control over data subspace

Suppose that we have a table hk with columns TIME and STATUS (among others). By default the virtual table

```
rh1012.fits[hk][status=5:10]
```

will have a data subspace which is the intersection of the original [hk] data subspace and the new condition status=5:10. This reflects the requirement 'give me all the rows whose status was from 5 to 10'. But sometimes we want a table with 'give me all the rows for times when the status was from 5 to 10'. The distinction is that if you then filter another table using the data subspace of your result, you want it to be a time filter rather than a status filter. (This is the functionality of the PROS timfilter task, but we want the capability to make filters on other quantities rather than just time). The virtual table

```
rh1012.fits[hk][status(time)=5:10]
```

has a data section identical to the previous example, but a data subspace with a new time filter instead of a status filter. It considers status as a function of time, rather than as an independent variable in its own right.

The virtual table

```
rh1012.fits[hk][status(hktime=time)=5:10]
```

is the same, but renames the time column in the data subspace to be called hktime.

2.6 Filtering on external tables

Another long-term goal of our filtering capability is to support easy interpolation filtering on external tables. For example, suppose that the table called EVENTS in file rh1012.fits contains a column TIME and rows representing photons. Suppose further that the file rh1012_ephem.fits contains a table ORBIT whose columns include ones named EPHTIME and HEIGHT, where EPHTIME contains values on the same timescale as TIME. Then we want to select records from EVENTS whose TIME value is such that the corresponding HEIGHT in the ORBIT table is above 10000 km (e.g. 'give me only photons above the radiation belts'). Currently to do this we would run several tools:

```
asc_copy rh1012_ephem.fits[orbit][height(ephtime)>10000.0] time_filter
asc_rename_column time_filter ephtime time
asc_copy rh1012.fits[@time_filter] result
```

Eventually we would like to do:

```
asc_copy rh1012.fits[events][rh1012_ephem.fits[orbit]height(time=ephtime)>10000.0] result
```

but that won't be supported initially. The same syntax would allow a simpler case if the column in the ORBIT table was named TIME not EPHTIME, and if the ORBIT table was in the same dataset:

```
asc_copy rh1012.fits[events][[orbit]height(time)>10000.0] result
```

We'd also like to do

```
asc_copy rh1012.fits[events][columns pha,time,[orbit]height(time)] result
```

which would add a new column called 'height' to the result[events] table whose values would be interpolated from the height versus time table in [orbit].

3 Virtual images

3.1 Basic syntax

The same simple syntax as before:

```
rh1012.fits
rh1012.fits[2]
rh1012.fits[exposure_map]
```

may also refer to an image. If the base datablock is also an image, the image is opened directly. If the base datablock is a table, the image must be created by binning columns of the table. By default, a 2D image is created by binning the first two columns of the table (or the first column if that column is 2-dimensional, e.g. 'detpos'). If 'preferred axes' are defined in the table, those columns are used instead.

A row-filtered table:

```
rh1012.fits[events][pha=4:8,time=100:200,400:500]
```

may also be used as input to a virtual image; in this case the rows are filtered prior to image binning. If the base datablock is an image, the same syntax implies an image section on the named axes.

3.2 IRAF notation

The old IRAF image section notation is

```
rh1012.fits[100:200,100:400]
```

which is equivalent to

```
rh1012.fits[#1=100:200,#2=100:400]
```

3.3 Binning

If we want to make an image on columns which are not the default columns, we use the syntax

```
rh1012.fits[bin pha,time]
```

This is the replacement for the old PROS key= syntax. The fact that the keyword 'bin' is followed by a space and not = allows us to recognize that this grouping is a binning command and not a filter on a column named 'bin'. We may also want to support the 'key=' syntax as a back compatibility option.

These binning operations on tables use a bin size of 1. If you want to use a different bin size and axis range, the syntax is

```
rh1012.fits[bin pha=4:10:2,time=100:200:10]
```

which makes a 2D image of counts versus pha and time, with the pha axis having 4 pixels running from 4 to 10 in steps of 2. or

```
rh1012.fits[bin energy=5:10:0.02]
```

which makes a 1-D image of counts versus energy (i.e. a spectrum), or

```
rh1012.fits[bin detpos=(100:900:4,200:1100:4)]
```

which makes a 2-D image (in the data model spec, an image with one 2-dimensional axis group) of counts versus detector position. or in general

```
block\_name[bin name=min:max:step,name=min:max:step,...]
```

To filter and bin,

```
rh1012.fits[myevents][pha=2:7,y=1000:1100][bin pha=4:10:2,time=100:200:10]
```

Note that this makes a 2-D pha, time image which is 4 x 10 pixels in size, but the pixels with pha more than 7 will be zero since those pha values were filtered from the table.

To filter on a spatial region,

```
rh1012.fits[myevents][(x,y)=circle(4096,4096,5)&!box(4100 4200 128 128),pha=2:7]
```

Any two columns may be grouped together to make a two dimensional quantity to which region filters apply;

```
rh1012.fits[myevents][(pha,time)=circle(4096,4096,5)&!box(4100 4200 128 128)]
```

although data model-compatible files may also contain predefined two-dimensional columns which may be accessed either by overall name (e.g. 'detpos') or by grouped component names (e.g. '(detx,dety)').

```
rh1012.fits[myevents][detpos=circle(4096,4096,5)&!box(4100 4200 128 128),pha=2:10]
```

To block an image by a given factor,

```
rh1012.fits[bin detpos=4]
```

or

```
rh1012.fits[bin (detx,dety)=4]
```

which is equivalent to the old

```
rh1012.fits[key=detx,dety][bl=4]
```

The bl=n blocking option would now by default be interpreted as a filter on a column called 'bl'. Again, back compatibility support is a possibility.

If we are binning on the default columns, we can omit the names:

```
rh1012.fits[bin 4:10:3,100:200:10]
rh1012.fits[bin 4]
```

Finally we can specify the output image array name. There are two names to specify: the name of the image datablock (for example 'energy_time_image') and the name of the array quantity itself (for example 'counts' or 'flux'). By default these two names are the same if they are not separately specified. We can also specify an intensity filter which must come after the binning

```
rh1012.fits[bin detpos=4][det_image]
rh1012.fits[bin detpos=4][det_image counts]
rh1012.fits[bin detpos=4][det_image counts=10:100]
rh1012.fits[bin detpos=4][det_image counts>50]
```

4 General syntax

4.1 Summary of syntax

Combining all the above specifications, our most general syntax is

dataset_name[block_id][filter-command][columns column-list][output-command]

for a table specification and

dataset_name[block_id][filter-command][bin binning-list][output-command] for an image specification

where

- block_id := block_name *or* block_number
- column-list := column-spec,column-spec...
- column-spec := column_name *or* (column_name,column_name...) *or* new_name = column_name *or* (new_name=column_name, column_name)
- filter-command := filter-component *or* (filter-component)|(filter-component)|...
- filter-component := filter-spec,filter-spec,...
- filter-spec := column-spec=filter *or* function(column-list)=filter
- filter := range-list *or* region-expression
- range-list := range,range,...
- range := min:max *or* min: *or* :max *or* value
- region-expression := region region-op region region-op region ...
- region-op := & *or* &! *or* | *or* |!
- output-command := block_name *or* block_name array_name *or* block_name array_name = filter
- binning-list := binning-spec,binning-spec,...
- binning-spec := column-spec *or* column-spec = binning
- binning := min:max:step *or* min:max *or* step *or* min::step *or* min: *or* :max *or* ::step

5 Detailed specification

5.1 Overall syntax

- The virtual datablock specification consists of a string consisting of a 'base dataset' name and a series of qualifier sections.
- Each qualifier section is a string delimited by square parentheses [].
- The possible qualifiers are:

  - block identifier
  - filter-command
  - column select/bin-command
  - output-command

  Any qualifier may be omitted, but those which appear must appear in this order.

- There may be at most one of each type of qualifier in a specification. If multiple cases of a qualifier are present, all but the first should be ignored. However, a future revision of this document will include 'stack syntax', which will provide an interpretation for multiple qualifiers.

- There are two kinds of virtual datablock specification: a table specification and an image specification. The allowed forms of the select/bin-command and the output-command differ in those cases; however, whether a given string is a table spec or an image spec depends on external context, and cannot in general be determined from the string itself.

- A column-select or bin command qualifier may be recognized by the fact that it begins with the text 'columns' or 'bin' followed by a space followed by further text.

- A block identifier may be recognized because it consists of only a single word.

- An output command may be recognized because it consists of either a single word, or a single word followed by a space and other text, and it cannot be a block identifier because it is not the first qualifier, and it cannot be a select/bin command because the word is not 'select' or 'bin'.

- A filter command can be recognized because it does not match any of the other qualifiers.

5.2 Base dataset

The base dataset identifies the input dataset (file or files).

- The base dataset name is the name of a file on disk. (Note that in the case of the ASC/ETOOLS IRAF/QPOE kernel, this file is a directory). The full filename should be given including any extension. URLs are not currently supported.

- If the base dataset name is '-' (dash), standard in is intended.

- If a virtual datablock specification is stored in a dataset (for instance a file is filtered and the filter command is recorded in the file) the leading base dataset name may be omitted if it is the dataset that the specification is being stored in. That will allow a dataset to have cross references to other tables within itself, without those cross references being broken when you rename the dataset.

- If no base dataset name is given and no dataset is otherwise specified to the software, the meaning is undefined. One possibility is that one should search for the block identifier within all datasets in the current path. For instance, one might refer to [ASPECT] and hope that the software will scan all the datasets in the current directory looking for a table called ASPECT. However, this functionality will not be supported for the time being as there are a lot of potential problems with it.

5.3 Block identifier

The block identifer specifies the input ASC Datablock (table or image) within the selected dataset.

- The block identifier, if present, must be the first qualifier section.

- The block identifier consists of a single word which may contain letters, digits, and the characters . (dot), - (minus), + (plus), / (slash), _ (underbar), # (hash), $ (dollar), ; (semicolon), (and) (round parentheses). Some other characters may also be accepted, but it may not contain whitespace or the characters comma (,), bar (|), square parens ([,]), equals (=), colon (:).

- If the block identifier consists only of digits, and represents an integer n which is less than 10 or less than the number of ASC tables in the dataset, the interpretation is that it refers to the nth ASC table in the dataset, counting from 1. The block identifer [0] is considered to be the same as [1].

- If the block identifier is empty, i.e. [], the first ASC Datablock in the dataset is implied.

- If no block identifier is present, the first ASC Datablock in the dataset is implied. So the following

  ```
  rh1012.fits
  rh1012.fits[]
  rh1012.fits[0]
  rh1012.fits[1]
  ```

  are all identical.

- Otherwise, the block identifier is interpreted as the name of an ASC Datablock within the dataset.

5.4 Filter-command

If the base dataset is a table, the filter-command qualifier selects rows from the table. If the base dataset is an image, the filter-command qualifier selects an image section.

- A filter command is an external filter command, a standard filter command, or a compound filter command.
- A standard filter command consists of only a single filter component.
- A filter component consists of a comma-separated list of filter specifications; the complete filter component is the logical AND of all of the filter specifications.
- A filter specification may be either a range filter, a relational filter, a region filter, or a functional filter.
- Each of these types of filter specification incorporates references to the names of column or axis descriptors or the row number. If the descriptor name consists of the symbol # followed by an integer, and the integer is less than or equal to the number of columns or axes in the ASC datablock, then the name is interpreted as referring to that numbered column (if a table) or axis (if an image).
- If the descriptor name contains the character [, it refers to a quantity in another datablock. We eventually want the ability to filter on a quantity interpolated from another table.
- Otherwise, the name must be one of the header, column or axis names or component names in the datablock.
- The entire filter component may optionally be enclosed in parentheses.
- An external filter command consists of the character @ followed by the name of a datablock (precisely, a virtual block command with only the dataset and block ID's), for example

  ```
  rh1012.fits[events][@tmp12.fits[2]]
  ```

  The data subspace of the specified datablock is opened and used to filter the input datablock.
- An external filter command may also be an ASCII file containing a valid filter specification. For example,

  ```
  rh1012.fits[events][@my.filt]
  ```

  where the file my.filt consists of the single line

```
pha=4:10,detpos=circle(54,52,2)
```

- A compound filter command consists of a set of filter components in round parentheses and separated by the | character, thus:

  $(F1)|(F2)|(F3)$

  This implies the logical OR of each of the restrictions F1, F2, F3. However, some applications may not be able to handle more than one filter component, and so multiple filter components should be avoided where possible.

5.5 Names and renaming in filter commands and select/bin commands

A descriptor name in a range, relational or region filter may be expressed in several ways:

- The name itself, which must be a data descriptor name or data descriptor component name in the input datablock. This is interpreted as generating a filter involving a restriction on that descriptor.
- A name #1, #2, etc. referring to a numbered column or axis.
- A pair of names separated by commas and enclosed in round parentheses, thus for example (detx,dety). This defines a 2-dimensional descriptor from two one-dimensional descriptors or component names.
- The special name #row is allowed for a filter command in a table datablock only. It points to a fake data descriptor whose value is the number of the row in the current input datablock, and so supports the ability to filter on row number.
- text of the form 'newname=descriptorname'. This has the effect of generating a filter on the given descriptor, but in the data subspace of the output table the filter is recorded as being on 'newname'.
- text of the form 'descriptorname(var)'. Example:

  ```
  rh1012.fits[events][height(time)=10:30]
  ```

  This has the intent of considering height as a function of time rather than as an independent variable. The filter generated will be a filter on those values of time for which height will be in the indicated range. The only difference in the output is that the output datablock's data subspace has a time filter rather than a height filter. This can be used in conjunction with the select none command to generate a time filter from a housekeeping file.

```
rh1012hk.fits[hk][voltage(time)=10:30][columns #none]
```

generates an empty table with a data subspace time filter for times where voltage was in the given range.

- We can also use the variable renaming syntax on the independent variable:

  ```
  rh1012hk.fits[hk][voltage(vtime=time)=10:30][columns #none][myfilter]
  ```

  which renames the variable time in [hk] to vtime in [myfilter].

- We can refer to a quantity in a datablock (we'll call it the external datablock) which is not the base datablock, provided we give a way to give that quantity meaning in the base datablock. We can provide this meaning if there is a common column in the two datablocks. We then consider the external datablock as a lookup table of the new quantity considered as a function of the common column. If the external datablock is in the same dataset, we just prefix the quantity name with the datablock ID. If it is in an external dataset, we prefix with the dataset name and datablock ID.

  Example: If the [hk] and [events] tables both contain a column called time, and [hk] is sorted on the time column, and [hk] has a column called voltage, then we can filter the table [events] on voltage by using the hk table to interpolate voltage versus time. Because there may be several columns in common, we must call out the common column explicitly:

  ```
  rh1012hk.fits[events][pha=4:10,[hk]voltage(time)=10:30]
  rh1012hk.fits[events][pha=4:10,my.fits[hk]voltage(time)=10:30]
  ```

  The output data subspace has a filter on time rather than voltage.

- Once again, we may perform variable renaming on the independent variable of the external reference. This may be necessary if the name of the 'common' column is different in the two tables.

  ```
  rh1012hk.fits[events][pha=4:10,[hk]voltage(time=hktime)=10:30]
  ```

5.6 Range filters

- A range filter defines a restriction on a scalar data descriptor (a column in the table or a scalar axis in the image). It consists of the name of the descriptor followed by an equals sign, followed by a range list.

- If the name alone is used, without an equals sign or range list, it must be a quantity of logical data type.

- A range list is a comma-separated list of ranges.
- Each range defines a minimum and maximum value. In the most general form, the minimum and maximum are given separated by a colon. The range is a closed interval on the real line. Several abbrevated forms are supported:
  - **min:max**
    Range from min to max.
  - **min:**
    Range from min to the maximum legal value (TLMAX) of the quantity.
  - **:max** Range from the minimum legal value (TLMIN) to max.
  - **val**
    A single value is equivalent to val:val, a range with only one value.

  In addition the following interval-definition formats are defined, but not yet supported. They are noted here because of their implications for parser design.

  - **[min:max]**, an explicitly closed interval, equivalent to the usual min:max.
  - **(min:max)**, an open interval.
  - **[min:max)** or **(min:max]**, a semi-open interval.

  Since these formats would follow either the equals sign or a comma, they cannot be confused with the opening parenthesis of a new qualifier section or filter component.

5.7 Image sections

A special variant of range filters is the image section syntax, when the names of the quantities are omitted and only one range is permitted for each quantity. In this case, the quantities are assumed to be #1, #2, etc. in order. Thus

```
rh1012.fits[events][4:8,1012:1234,10:20]
```

is equivalent to

```
rh1012.fits[events][#1=4:8,#2=1012:1234,#3=10:20]
```

If Image section syntax is used, it may not be mixed with other kinds of filter command.

5.8 Relational filters

The range filter

```
pha = 1:10,80:300,51:100
```

is read as: 'the list of restrictions on pha is ...'. An alternative filter syntax

```
pha op value
```

may be read as 'true if pha op value'.

The supported operators are

- $>$, greater than
- $<$, less than,
- $>=$, greater than or equal
- $<=$, less than or equal
- $!=$, not equal to

5.9 Region filters

- A region filter consists of a name followed by an equals sign followed by a region expression. The name must refer to a two-dimensional quantity.
- A region expression is a set of region elements joined with the logical region operators & (and), | (or) $\wedge$ (exclusive or) and ! (not) and grouped with round parentheses.
- The region elements define shapes in a two-dimensional plane. Each region element has two forms: the element name followed by round parentheses containing a comma-separated list of parameters, or the element name followed by a space-separated list of parameters terminated by a comma, a region operator, or the closing square parenthesis of the qualifier. Each region element name also has an abbreviated form.
- Allowed region elements are
  - CIRCLE(xcen ycen radius)
  - ANNULUS(xcen ycen rad1 rad2)
  - BOX(xcen ycen width height angle)
  - ELLIPSE(xcen ycen width height angle)
  - PIE(xcen ycen radius angle1 angle2)

5.10 Functional filters

A functional filter defines a restriction based on the value of a function of one or more named quantities. For example,

```
dist(x,y,40,20)<5
```

which is equivalent to the region filter

```
(x,y)=circle(40,20,5)
```

The list of supported functions is:

- sin(x)
- cos(x)
- exp(x)
- ln(x)
- log(x)
- max(x1,x2)
- min(x1,x2)
- eval(expression)

The last of these, EVAL, is a placeholder to support generic algebraic expressions on quantities. For instance, one might eventually want to support a restriction

```
 PHA + PI > 2 * log( ENERGY )
```

Rather than complicate the parser trying to recognize this as one of a comma-separated list of filter expressions, we will require that the EVAL keyword be used to mark off such expressions, as:

```
rh1012.fits[events][time=100:500,eval(pha+pi>2*log(energy)),(x,y)=circle 5 5 3]
```

eval is a reserved name corresponding to a pseudo-quantity whose data type is logical.

5.11 Select command

The select command selects columns for an output table. For an image, it specifies the names and ordering of output axes, and is interpreted as a bin command. A bin command and a select command may not be present in the same virtual datablock specification.

- The select command is recognized by the parser as a qualifier which begins with the word COLUMNS followed by a space followed by other text.
- The select text consists of a comma separated list of column specifications.
- The reserved column specifications #all (synonym *) and #none (synonym -) denote selection of all of the columns of the input table or none of them respectively.
- Omitting the select command is equivalent to [columns *].
- The column specification newname=oldname renames a column from oldname to newname.
- The column name #n, where n is an integer, denotes the nth column in the base datablock.
- The specification !name means delete that column. It is only useful when no other columns are positively selected. Thus

  ```
  rh1012.fits[events][columns !pha,!status]
  rh1012.fits[events][columns *,!pha,!status]
  ```

  each make a table with all the columns of [events] except pha and status, but

  ```
  rh1012.fits[events][columns time,!pha,!status]
  ```

  makes a table with only time, and the !pha,!status commands are redundant.
- An external column name may be given, as in the filter command. Example:

  ```
  rh1012.fits[events][columns time,pha,ecal.fits[ecaldata]energy(pha)]
  ```

  selects columns time and pha from the rh1012.fits[events] table and makes a new column energy, whose values are found by taking the pha values for each row in [events] and evaluating the corresponding value of energy in ecal.fits[ecaldata].

5.12 Binning command

The binning command is only allowed for image specifications. An image specification defines a virtual image (of arbitrary dimension 1, 2, 3, ..., etc) in terms of an input image or table. In the first case, it is a rebinning of the original image; in the second, it bins certain columns of the table to form an image.

The binning command defines the axes of the output image and their definition in terms of the quantities in the input image or table.

- The binning command qualifier starts with the word 'bin' followed by a space followed by the binning list.
- The binning list is a comma-separated list of binning specifications, each of which defines one axis or axis group.
- The simplest kind of binning specification is a name. The name specifies that that axis is next in the ordering. If the base datablock is a table, the corresponding column will be used binned on. The axis will extend from TLMAXn to TLMINn in steps of size 1.0. Example:

  ```
  rh1012.fits[events][bin pha,time]
  ```

  makes a pha versus time image.

  ```
  rh1012.imh[bin #2,#1]
  ```

  transposes the input image, making axis number 2 in the old image be axis number 1 in the new image.

- The axis may be renamed by preceding it with the new name and an equals sign. Example:

  ```
  rh1012.fits[events][bin pha,sctime=time]
  ```

  makes a pha versus time image.

  ```
  rh1012.imh[bin dety=#1,detx=#2]
  rh1012.imh[bin detpos(detx=#1,dety=#2)]
  ```

- The name may be followed by an equals sign and a binning. The binning specifies a start value, a stop value and a step size in the form min:max:step. The defaults are TLMIN, TLMAX and 1.0. Alternate forms supported are:
  - min:max:step

- min:max
- min:
- :max
- step
- ::step
- min::step
- :max:step

The only potential parsing confusion is with the case with no colons, where there is conflict with a column/axis name whose name is the same as the step size. For instance, does

```
rh1012.fits[bin pha=4]
```

mean 'bin pha by 4' or 'bin a column called '4' by 1 and rename it pha'? Another good reason to not give columns numeric names. In this case we will require columns in the binning spec to not have numeric names - if you do have such a column, you'll have to use the #n notation to reference it.

- For multi-dimensional columns or axes, the syntax is to have a comma-separated list of binnings, one for each axis, enclosed in round parentheses.

```
rh1012.fits[bin detpos=(1012:4096:4,2048:4096:4)]
```

As a special helper, replacing this grouping by a single integer is interpreted as a step size (blocking factor) applying to all axes, so these are equivalent:

```
rh1012.fits[bin (detx,dety)=(4,4)]
rh1012.fits[bin (detx,dety)=4]
```

5.13 Output command

The output command allows the user to specify the name of the output datablock.

- For a virtual table specification, the output command must be a simple name, which names the output table datablock.
- If the output command is absent, the output datablock has the same name as the input datablock. (The output dataset name is determined by the application, not by the syntax defined here).

- If the output command is present, another qualifier must also be present to avoid ambiguity with the block identifier. Thus

  ```
  rh1012.fits[events]
  ```

  specifies an input datablock, while

  ```
  rh1012.fits[][events]
  ```

  specifies an output datablock.

- For a virtual image specification, the output datablock name may optionally be followed by a space and the name of the data quantity, or may be enclosed in an extra set of square parens and followed by the data quantity name. Example:

  ```
  rh1012.fits[events][bin pha][spectrum counts]
  rh1012.fits[events][bin pha][[spectrum]counts]
  ```

  bins the [events] table on pha and makes a 1-D array called counts in an image datablock called [spectrum]. The first form is a little less ugly, but the second form is more consistent with the rest of the syntax.

- If the data quantity name is missing, it is given the same name as the output data block. Thus

  ```
  rh1012.fits[events][bin pha]
  ```

  is equivalent to

  ```
  rh1012.fits[events][bin pha][events events]
  ```

  which is probably not what you want, but unless we provide some hints in the input file about how to name binning products, I don't see a generic way around it.

- For a virtual image specification, the name of the data quantity may be provided in the form of a range or relational filter. In an image, the datablock name is used as an alternate data descriptor name for the pixel values in the image. Thus, a filter on the datablock name implies an intensity filter on the image.

  ```
  asccopy rh1012.fits[events][bin pha][spectrum counts>10] result
  ```

  is equivalent to

  ```
  asccopy rh1012.fits[events][bin pha][spectrum counts] tmp
  asccopy tmp[spectrum][counts>10] result
  ```