FITS Table Indexing Proposal

7 April 1997

William Pence (NASA/GSFC)
Mike Denby (Leicester)
Carlo Izzo (MPE)
Don Jennings (ISDC, Geneva)
Clive Page (Leicester)
Ulrich Zimmermann (MPE)

1 Introduction

When dealing with very large FITS tables it may be useful to provide an index on the table contents so that FITS reading software can more quickly locate the rows in the table that contain the desired subset of information. This proposal describes a convention that may be used to define and store an index for any FITS binary or ASCII table. Strictly speaking this implementation deals only with sparse indexes to already sorted datasets; the provision of more general indexes, such as dense ones, or secondary indexes to a separate table, is beyond the current scope.

In the following discussion the FITS table that is being indexed is considered to contain a list of events where each row of the table corresponds to a single event and the columns of the table represent attributes which describe each event. In typical astrophysics applications the columns may give the time, spatial coordinate, and energy associated with each event. This proposal is intended to be very general and supports all of the following features:

- a 1-dimensional index on any single column of a table.
- a 2-dimensional index where the table events are sorted into a 2-D grid of bins based on the values in any 2 columns of the table.
- a n-dimensional index on an arbitrary number of event columns. In practice, there are probably few applications that will benefit from more than a 2-D index.
- the minimum and maximum value of any other table columns for all the events within each index bin may be specified. This is analogous to the feature in IRAF QPOE files which stores the minimum and maximum value of each attribute (e.g., X, Y, TIME, PHA, etc.) in large fixed-size 'buckets' of events. This allows software to quickly check if there are any events within the bin which meet a given search criterion without having to check every individual event. This feature may be espically useful in tables that are otherwise not indexed or sorted by any column except for the virtual 'row number' column which all tables inherently possess. (See example 6, below).
- tables need not be physically sorted, but instead may contain another column in the table which contains a pointer to the next event in that index bin (i.e., a linked list structure). Doubly-linked tables, with both forward and backward links, are also supported.
- the indexed event list does not need to be entirely contained in a single FITS table extension. The events may instead be sorted and subdivided into multiple FITS tables.
- the size of the index may be compressed by deleting any index bins which contain no events. This can make the index less convenient to use, but in some cases the space savings may be a more important consideration.

2 Index Header Keywords

This implementation uses the FITS binary table extension format (`XTENSION= 'BINTABLE'`) to store the table index information. In addition to the mandatory and reserved keywords that apply to all FITS binary table extensions a number of other keywords are defined here and have a reserved meaning within this convention, as described below:

```
EXTNAME = '          '   / TABLE_INDEX, TIME_INDEX or SPATIAL_INDEX (required)
```

The EXTNAME keyword may have one of 3 values under this convention. First, there are 2 common sub-types of indices that are given specific names: `EXTNAME = 'TIME_INDEX'` should be used for a 1-dimension index on a time-ordered column; `EXTNAME = 'SPATIAL_INDEX'` should be used for a 2-dimensional index on 2 spatial position columns (e.g., 'X' and 'Y' columns). Indices that do not fall under one of these 2 subclasses should be given the generic name `'TABLE_INDEX'`. These 3 names are not case-sensitive and may be specified using any mixture of upper and lower case letters.

```
DATE    = '1996-11-19T13:37:34' / date and time of index creation (optional)
```

The DATE keyword records the time that the index was created. This date may be used to help verify that the index is still valid. If, for example, the indexed table has a creation date later than the date of the index, then this may be an indication that the index is obsolete.

```
INDTABLE= 'myfile.fits:BINTABLE:EVENTS:1' / name of indexed table (optional)
```

The naming convention used here conforms to that used in the Jennings et al. FITS file grouping proposal (http://adfwww.gsfc.nasa.gov/other/convert/group.html). Basically, the value of this keyword gives the name of the FITS file followed by the value of the XTENSION, EXTNAME, and EXTVERS keywords in the table that is indexed. In general the table name is considered case sensitive. Other examples of the INDTABLE keyword are:

```
INDTABLE= 'http://mycomp.stateu.edu/user/data/filename.fits:BINTABLE:EVENTS:1'
INDTABLE= '/user/data/filename.fits:BINTABLE:EVENTS:1'  / full path
INDTABLE= 'filename.fits:BINTABLE:EVENTS:1'             / filename only
INDTABLE= ':BINTABLE:EVENTS:1'    / table and index are in the same FITS file
```

If the event list is split into more than one physical FITS table, then 'INDTABLE' becomes a column in the index table rather than a keyword and the individual table name for each bin of the index is than listed in the body of the index table. (See example 5 below). Note that this keyword is considered optional because other conventions may be used to associate a particular table with its index. In particular the file grouping proposal mentioned above provides a general mechanism for constructing hierarchical associations of FITS header-data units.

```
INDCHECK= '       1490'        / checksum of the indexed table data (optional)
```

The value of the INDCHECK keyword may be compared to the computed checksum of the indexed event list data (not including the header keywords) to verify that the event list has not been modified subsequent to creating the index. If the event list is split into more than one physical FITS table, then 'INDCHECK' becomes a column in the index table rather than a keyword. Note that the checksum value is an unsigned integer, but it is represented as a character string (i.e., enclosed in quotes) because the value of the integer may be too large to be stored in the native integer format on some computer systems.

```
IAXIS   = (positive integer) / index dimensionality: 1, 2, ... (required)
```

The IAXIS keyword gives the dimensionality, or number of axes, in the index. In most cases the value will be either 1 (e.g., for an index on the TIME column) or 2 (e.g., for a spatial index on a pair of coordinate columns).

```
IAXISn  = (positive integer) / number of bins along each index axis (required)
```

The value of this indexed keyword contains the length (number of bins) along the corresponding axis of the index. Ordinarily, the value of these keywords corresponds to the number of rows in the index so that the total number of rows in the index (as given by the NAXIS2 keyword) is equal to the product of all the IAXISn keyword values. As discussed below, however, it is possible to compress the size of the index by deleting all the index bins which contain no events in which case NAXIS2 will be less than the product of the IAXISn keywords.

```
ITYPEn  = 'column name'      / column name of index axis n  (required)
```

The ITYPEn keyword gives the name of the column in the indexed table (as given by the TTYPEn keyword) corresponding to this axis of the index. The specified column name is not case sensitive. In the special case where the event list is not sorted and is simply indexed on row number, then INDTABLE will have the reserved value of `'(row#)'` or `'(ROW#)'`. (See example 6).

```
ISTRTn  = (float value) / first inclusive value of first index bin (required)
```

The ISTRTn keyword gives the starting inclusive value for the first index bin along the corresponding axis of the index. It is used in combination with the IDELTn keyword to determine the range of values in each bin of the index.

```
IDELTn  = (float value) / width of bins along this index axis (required)
```

The IDELTn keyword gives the width (or delta) of the bins along the corresponding axis of the index. Negative values of IDELTn are allowed and indicate that the index is sorted in inverse order. The values included in the Nth bin of the index then range from ISTRTn + (N - 1) * IDELTn (inclusive) to ISTRTn + N * IDELTn (exclusive). To be precise, for positive values of IDELTn the range is greater than or equal to the starting value and less than (but not equal to) the last value. In order to reduce the effect of numerical rounding errors, any computations using ISTRTn and IDELTn keywords should be performed using double precision arithmetic. The values of the ISTRTn and IDELTn keywords should also be given with maximum possible numerical precision (typically with up to 15 or 16 decimal places).

```
FORELINK= 'column name'      / column containing forward  links (optional)
BACKLINK= 'column name'      / column containing backward links (optional)
```

If the FITS table is not physically sorted and instead contains columns with forward and/or backward pointers to the next or previous event in the index bin, then the FORELINK (and optionally, BACKLINK) keyword(s) will be present. The specified column name is not case sensitive.

3 Index Table Columns

The index table itself can contain the following columns. The order of the columns in the table is not significant and the names of the columns are case-insensitive. Other columns may be included in the index table but they are not defined by this convention.

`IND_POINTER (required) (integer)`

This column gives the row number (starting with 1 not 0) of the first event in the indexed table that falls within this index bin.

`IND_LENGTH  (optional) (integer)`

This column gives the number of rows/events in the indexed table that are in this index bin. If the IND_LENGTH column is not given then the number of events in each bin can be assumed to be equal to the difference between the consecutive IND_POINTER column values. For example, if the IND_POINTER value for the 2nd index bin is 50 and the IND_POINTER value for the 3rd bin is 70, then there must be 70 - 50 = 20 events in the 2nd index bin. Unless the FORELINK keyword is specified, it is assumed that the events in the indexed table have been sorted so that the events in each index bin all occur in rows IND_POINTER through (IND_POINTER + IND_LENGTH - 1) of the indexed table.

`colname_MIN (optional)`
`colname_MAX (optional)`

These 2 columns give the minimum and maximum value of the specified column in the indexed table for all the events in this index bin. This pair of columns may be repeated for every column in the indexed table. The data type of these columns should be the same as the data type of the original column in the indexed table.

`IND_STARTn (optional) (double-precision floating-point)`

This column should only be present when the size of the index table has been compressed by eliminating all rows that would have a value of IND_LENGTH = 0. The value in this column gives the starting value of the index bin that would otherwise have been implied by the row number and the ISTRTn and IDELTn keyword values. (See example 7, below).

4 Index Table Row Organization

The number of rows in the index table (as defined by the NAXIS2 keyword) will ordinarily be equal to the product of all the IAXISn keyword values. In the case of multi-dimensional indices, the index along the 1st axis varies most rapidly and those along subsequent axes progressively less rapidly. Thus, for a 2-dimensional index the first few rows of the index table correspond to index bins (1,1), (2,1), (3,1) and so on, followed by (1,2), (2,2), (3,2), etc.

There is one special case, however, where the number of rows in the index table may be less than the product of all the IAXISn keywords. This can occur when the index would otherwise contain many rows all with IND_LENGTH = 0. A good example of this is the case where a table contains events observed during 2 1000-second exposures taken a year apart. If one tried to construct an index on the TIME column at 10-second intervals, for example, then the index would contain a huge number of rows containing no events during the one year interval between the 2 exposures. The solution to this problem is to compress the size of the index by deleting all rows that have IND_LENGTH = 0. In order to then determine the time interval covered by each remaining row in the index table, it is necessary to include the IND_STARTn column (as defined above) in the table. This special case can be recognized by the fact that NAXIS2 is less than the product of the IAXISn keywords and by the presence of the IND_STARTn column. Example 7 below illustrates this case.

5 Example Index Tables

In all the following examples, the FITS event table that is indexed contains the following 4 columns with the indicated range of values: TIME (10000 - 50000), X (1 - 512), Y (1 - 512), PHA (1 - 128).

5.1 Example 1

The table is indexed by the TIME column value (1-D index)

```
XTENSION= 'BINTABLE'           / binary table extension
BITPIX  =                    8 / 8-bit bytes
NAXIS   =                    2 / 2-dimensional binary table
NAXIS1  =                    8 / width of table in bytes
NAXIS2  =                   40 / 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  = 'Ind_Pointer'        / label for field   1
TFORM1  = '1J      '           / data format of field: 4-byte INTEGER
TTYPE2  = 'Ind_Length'         / label for field   2
TFORM2  = '1J      '           / data format of field: 4-byte INTEGER
EXTNAME = 'Time_Index'         / this is a time-ordered index of a FITS table
INDTABLE= 'myfile.fits:BINTABLE:EVENTS:1' / name of the indexed table
IAXIS   =                    1 / dimensionality of the index
ITYPE1  = 'TIME    '           / first index column name
ISTRT1  =               10000. / beginning value of first index bin
IDELT1  =                1000. / width of the index bin
IAXIS1  =                  40  / number of bins in this axis of the index
END
```

The table contains 40 rows and 2 columns:

```
IND_POINTER   IND_LENGTH
-----------   ----------
     1           70
    71           80
   151            0
   151            0
   151            0
   151           20
   171           37
     .            .
```

5.2 Example 2

The table is indexed by X and Y (a 2-D spatial type index)

```
XTENSION= 'BINTABLE'           / binary table extension
BITPIX  =                    8 / 8-bit bytes
NAXIS   =                    2 / 2-dimensional binary table
NAXIS1  =                    8 / width of table in bytes
NAXIS2  =                  256 / 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  = 'Ind_Pointer'        / label for field   1
TFORM1  = '1J      '           / data format of field: 4-byte INTEGER
TTYPE2  = 'Ind_Length'         / label for field   2
TFORM2  = '1J      '           / data format of field: 4-byte INTEGER
EXTNAME = 'Spatial_Index'      / this is a spatial index of a FITS table
INDTABLE= `myfile.fits:BINTABLE:EVENTS:1'  / name of the indexed table
IAXIS   =                    2 / dimensionality of the index
ITYPE1  = 'X       '           / first index column name
IAXIS1  =                   16 / number of bins in this axis of the index
ISTRT1  =                   1. / beginning value of first index bin
IDELT1  =                  32. / width of the index bin
ITYPE2  = 'Y       '           / 2nd index column name
IAXIS2  =                   16 / number of bins in this axis of the index
ISTRT2  =                   1. / beginning value of 2nd index bin
IDELT2  =                  32. / width of the index bin
END
```

The table contains 16 x 16 = 256 rows and 2 columns:

```
IND_POINTER   IND_LENGTH
-----------   ----------
     1           320
    321          505
     .            .
```

5.3 Example 3

Same as example 2, except that the index includes additional information about the range of values in the other columns.

```
XTENSION= 'BINTABLE'           / binary table extension
BITPIX  =                    8 / 8-bit bytes
NAXIS   =                    2 / 2-dimensional binary table
NAXIS1  =                   28 / width of table in bytes
NAXIS2  =                  256 / number of rows in table
PCOUNT  =                    0 / size of special data area
GCOUNT  =                    1 / one data group (required keyword)
```

```
TFIELDS =                    6 / number of fields in each row
TTYPE1  = 'Ind_Pointer'        / label for field   1
TFORM1  = '1J      '           / data format of field: 4-byte INTEGER
TTYPE2  = 'Ind_Length'         / label for field   2
TFORM2  = '1J      '           / data format of field: 4-byte INTEGER
TTYPE3  = 'Time_MIN'           / label for field   3
TFORM3  = '1D      '           / data format of field: 8-byte DOUBLE
TTYPE4  = 'Time_MAX'           / label for field   4
TFORM4  = '1D      '           / data format of field: 8-byte DOUBLE
TTYPE5  = 'PHA_MIN'            / label for field   5
TFORM5  = '1I      '           / data format of field: 2-byte INTEGER
TTYPE6  = 'PHA_MAX'            / label for field   6
TFORM6  = '1I      '           / data format of field: 2-byte INTEGER
EXTNAME = 'Spatial_Index'      / this is a spatial index of a FITS table
INDTABLE= 'myfile.fits:BINTABLE:EVENTS:1'  / name of the indexed table
IAXIS   =                    2 / dimensionality of the index: 1, 2, 3, ...
ITYPE1  = 'X       '           / first index column name
IAXIS1  =                   16 / number of bins in this axis of the index
ISTRT1  =                   1. / beginning value of first index bin
IDELT1  =                  32. / width of the index bin
ITYPE2  = 'Y       '           / 2nd index column name
IAXIS2  =                   16 / number of bins in this axis of the index
ISTRT2  =                   1. / beginning value of 2nd index bin
IDELT2  =                  32. / width of the index bin
END
```

The table contains 16 x 16 = 256 rows and 6 columns:

```
IND_POINTER  IND_LENGTH  TIME_MIN  TIME_MAX  PHA_MIN  PHA_MAX
-----------  ----------  --------  --------  -------  -------
     1           320      11035     41700       13      110
   321           505       5010     42150       18      123
     .            .          .         .         .        .
```

5.4 Example 4

Same as example 2 except that the table is not physically sorted and instead contains a column called 'Next_Pointer' that points to the next event in the index bin.

```
XTENSION= 'BINTABLE'           / binary table extension
BITPIX  =                    8 / 8-bit bytes
NAXIS   =                    2 / 2-dimensional binary table
NAXIS1  =                    8 / width of table in bytes
NAXIS2  =                  256 / 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  = 'Ind_Pointer'        / label for field   1
```

```
TFORM1  = '1J      '           / data format of field: 4-byte INTEGER
TTYPE2  = 'Ind_Length'         / label for field   2
TFORM2  = '1J      '           / data format of field: 4-byte INTEGER
EXTNAME = 'Spatial_Index'      / this is a spatial index of a FITS table
INDTABLE= 'myfile.fits:BINTABLE:EVENTS:1'  / name of the indexed table
IAXIS   =                    2 / dimensionality of the index
ITYPE1  = 'X       '           / first index column name
IAXIS1  =                   16 / number of bins in this axis of the index
ISTRT1  =                   1. / beginning value of first index bin
IDELT1  =                  32. / width of the index bin
ITYPE2  = 'Y       '           / 2nd index column name
IAXIS2  =                   16 / number of bins in this axis of the index
ISTRT2  =                   1. / beginning value of 2nd index bin
IDELT2  =                  32. / width of the index bin
FORELINK= 'Next_Pointer'       / name of the column containing forward link
END
```

The table contains 16 x 16 = 256 rows and 2 columns:

```
IND_POINTER   IND_LENGTH
-----------   ----------
   412           320
    65           505
     .            .
```

5.5 Example 5

Same as example 2 except that the event list is stored in several different binary tables. All the events in a particular bin of the index must be contained in a single table, but different bins may be in different FITS tables. In this case all the events in the first and second index bin are in different binary table extensions in 'myfile1.fits' (thus, the extensions have different EXTVERS keyword values), and the third bin of events is contained in 'myfile2.fits'.

```
XTENSION= 'BINTABLE'           / binary table extension
BITPIX  =                    8 / 8-bit bytes
NAXIS   =                    2 / 2-dimensional binary table
NAXIS1  =                    8 / width of table in bytes
NAXIS2  =                  256 / number of rows in table
PCOUNT  =                    0 / size of special data area
GCOUNT  =                    1 / one data group (required keyword)
TFIELDS =                    3 / number of fields in each row
TTYPE1  = 'Ind_Pointer'        / label for field   1
TFORM1  = '1J      '           / data format of field: 4-byte INTEGER
TTYPE2  = 'Ind_Length'         / label for field   2
TFORM2  = '1J      '           / data format of field: 4-byte INTEGER
TTYPE3  = 'IndName '           / label for field   2
TFORM3  = '30A     '           / data format of field: 4-byte INTEGER
EXTNAME = 'Spatial_Index'      / this is a spatial index of a FITS table
IAXIS   =                    2 / dimensionality of the index
```

```
ITYPE1  = 'X       '           / first index column name
IAXIS1  =                   16 / number of bins in this axis of the index
ISTRT1  =                   1. / beginning value of first index bin
IDELT1  =                  32. / width of the index bin
ITYPE2  = 'Y       '           / 2nd index column name
IAXIS2  =                   16 / number of bins in this axis of the index
ISTRT2  =                   1. / beginning value of 2nd index bin
IDELT2  =                  32. / width of the index bin
END
```

The table contains 16 x 16 = 256 rows and 3 columns:

```
IND_POINTER   IND_LENGTH  INDNAME
-----------   ----------  ------------------------------
     1          320       myfile1.fits:BINTABLE:EVENTS:1
     1          505       myfile1.fits:BINTABLE:EVENTS:2
     1          310       myfile2.fits:BINTABLE:EVENTS:1
     .            .             .
```

5.6 Example 6

The table is not sorted on any columns except for the implicit 'row number' column that is in every table. In this case the index just lists the min and max value of each column in bins of 1000 rows each. The IND_POINTER and IND_LENGTH columns are redundant since the same information is provided by the ISTRT1 and IDELT1 keywords, but they are included for clarity.

```
XTENSION= 'BINTABLE'           / binary table extension
BITPIX  =                    8 / 8-bit bytes
NAXIS   =                    2 / 2-dimensional binary table
NAXIS1  =                   44 / width of table in bytes
NAXIS2  =                   50 / number of rows in table
PCOUNT  =                    0 / size of special data area
GCOUNT  =                    1 / one data group (required keyword)
TFIELDS =                   10 / number of fields in each row
TTYPE1  = 'Ind_Pointer'        / label for field   1
TFORM1  = '1J      '           / data format of field: 4-byte INTEGER
TTYPE2  = 'Ind_Length'         / label for field   2
TFORM2  = '1J      '           / data format of field: 4-byte INTEGER
TTYPE3  = 'Time_MIN'           / label for field   3
TFORM3  = '1D      '           / data format of field: 8-byte DOUBLE
TTYPE4  = 'Time_MAX'           / label for field   4
TFORM4  = '1D      '           / data format of field: 8-byte DOUBLE
TTYPE5  = 'X_MIN'              / label for field   5
TFORM5  = '1J      '           / data format of field: 2-byte INTEGER
TTYPE6  = 'X_MAX'              / label for field   6
TFORM6  = '1J      '           / data format of field: 2-byte INTEGER
TTYPE7  = 'Y_MIN'              / label for field   7
TFORM7  = '1J      '           / data format of field: 2-byte INTEGER
TTYPE8  = 'Y_MAX'              / label for field   8
```

```
TFORM8  = '1J      '           / data format of field: 2-byte INTEGER
TTYPE9  = 'PHA_MIN'            / label for field   9
TFORM9  = '1I      '           / data format of field: 2-byte INTEGER
TTYPE10 = 'PHA_MAX'            / label for field  10
TFORM10 = '1I      '           / data format of field: 2-byte INTEGER
EXTNAME = 'TABLE_INDEX'        / this extension is an index of a FITS table
INDTABLE= `myfile.fits:BINTABLE:EVENTS:1' / name of the table that is indexed
IAXIS   =                1     / dimensionality of the index
ITYPE1  = '(row#)  '           / reserved virtual column name
IAXIS1  =               50     / number of bins in this axis of the index
ISTRT1  =                1.    / beginning value of first index bin
IDELT1  =             1000.    / width of the index bin
END
```

The table contains 50 rows and 10 columns:

```
IND_POINTER IND_LENGTH TIME_MIN TIME_MAX X_MIN X_MAX Y_MIN Y_MAX PHA_MIN PHA_MAX
----------- ---------- -------- -------- ----- ----- ----- ----- ------- ----
    1          1000      11035.    41700.   17   490    23    502    13     11
 1001          1000       5010.    42150.   26   468     5    511    18    123
    .             .          .         .     .     .     .      .     .      .
```

5.7 Example 7

The same as example 1 except that the size of the index has been compressed by deleting all rows with IND_LENGTH = 0. The fact that the table has been compressed is evident from the fact that NAXIS2 is less than IAXIS1. This case requires that the IND_START1 column be added to give the starting time of each index bin.

```
XTENSION= 'BINTABLE'             / binary table extension
BITPIX  =                    8 / 8-bit bytes
NAXIS   =                    2 / 2-dimensional binary table
NAXIS1  =                   16 / width of table in bytes
NAXIS2  =                   20 / number of rows in table
PCOUNT  =                    0 / size of special data area
GCOUNT  =                    1 / one data group (required keyword)
TFIELDS =                    3 / number of fields in each row
TTYPE1  = 'IND_START1'           / label for field   1
TFORM1  = '1D      '             / data format of field: 8-byte DOUBLE
TTYPE2  = 'Ind_Pointer'          / label for field   2
TFORM2  = '1J      '             / data format of field: 4-byte INTEGER
TTYPE3  = 'Ind_Length'           / label for field   3
TFORM3  = '1J      '             / data format of field: 4-byte INTEGER
EXTNAME = 'Time_Index'           / this is a time-ordered index of a FITS table
INDTABLE= `myfile.fits:BINTABLE:EVENTS:1' / name of the indexed table
IAXIS   =                    1 / dimensionality of the index
ITYPE1  = 'TIME    '             / first index column name
ISTRT1  =               10000. / beginning value of first index bin
IDELT1  =                1000. / width of the index bin
```

```
IAXIS1  =                   40  / number of bins in this axis of the index
END
```

The table contains 20 rows and 3 columns:

```
 IND_START1    IND_POINTER   IND_LENGTH
 ----------    -----------   ----------
    10000.          1           70
    11000.         71           80
    15000.        151           20
    16000.        171           37
       .            .            .
```