Tiled-Table Convention for Compressing FITS Binary Tables

William Pence, NASA/GSFC
Rob Seaman, NOAO
Richard L. White, STScI

Version 2.0
November 21, 2013

1 Overview

This document describes a convention for compressing FITS binary tables that is modeled after the widely used FITS tiled-image compression method (White et al. 2009). The uncompressed table may be subdivided into tiles, each containing the same number of rows, then each column of data within each tile is extracted, compressed, and stored as a variable-length array of bytes in the output compressed table. Most of the header keywords from the uncompressed table, with only a few limited exceptions, are copied verbatim to the header of the compressed table. These header keywords remain uncompressed for efficient access. The compressed table is itself a valid FITS binary table that contains the same number and order of columns as in the uncompressed table, and contains one row for each tile of rows in the uncompressed table. All the currently supported compression algorithms (Rice and 2 variants of Gzip) are lossless, so no information is lost when the table is compressed.

This convention currently only supports FITS binary tables and cannot be used to compress FITS ASCII tables.

2 Compression Overview

The procedure for compressing a FITS binary table consists of the following sequence of steps:

A. **Divide Table into Tiles (Optional)**

In order to limit the amount of data that must be managed at one time, large FITS tables may be optionally divided into tiles, each containing the same number of rows (except for the last tile which may contain fewer rows). Each tile of the table is compressed in turn and is stored in a single row in the output compressed table. There is no fixed upper limit on the allowed tile size, but for practical purposes, it is recommended that it not exceed 100 MB so as to not impose too great of a memory resource burden on software that compresses or uncompresses the table.

B. **Decompose each Tile into the Component Columns**

FITS binary tables are physically stored in row-by-row sequential order, such that the data values for the first row in each column are followed by the values in the second row, and so on. Because adjacent columns in binary tables can contain very non-homogeneous types of data, it can be challenging to efficiently compress the native stream of bytes in the FITS tables. For this reason, the table is first decomposed into its component columns, and then each column of data is compressed separately. This also allows one to choose the most efficient compression algorithm for each column.

C. **Compress Each Column of Data**

Each column of data is compressed with a suitable compression algorithm. If the table is divided into tiles, then the same compression algorithm must be applied to a given column in every tile. In the case of variable-length array columns, (where the data are stored in the table heap), each individual variable length vector is compressed separately.

D. **Store the Compressed Bytes**

The compressed stream of bytes for each column is written into the corresponding column in the output table. The compressed table has exactly the same number and order of columns as the input table, however the data type of the columns in the output table will all have a variable-length byte data type, with TFORMn = '1QB', which is appropriate for storing the compressed stream of bytes. Each row in the compressed table corresponds to a tile of rows in the uncompressed table.

In the case of variable-length array columns, the array of descriptors that point to each compressed variable-length array, as well as the array of descriptors from the input uncompressed table, are also compressed and written into the corresponding column in the compressed table. See section 6 for more details.

3 Compression Directive Keywords

The following optional 'compression directive' keywords, if present in the header of the table that is to be compressed, provide guidance to the compression software on how the table should be compressed. The compression software will attempt to obey these directives, but if that is not possible, the software may disregard them and use an appropriate alternative.

- `FZTILELN` The value field of this keyword shall contain an integer that specifies the requested number of table rows in each tile which are to be compressed as a group.

- `FZALGOR` The value field of this keyword shall contain a character string giving the mnemonic name of the algorithm that is requested to be used by default to compress every column in the table. The current allowed values are `GZIP_1`, `GZIP_2`, and `RICE_1`. The corresponding algorithms are described in Section 5.

- `FZALGn`. The value field of these keywords shall contain a character string giving the mnemonic name of the algorithm that is requested to be used to compress column `n` of the table. The current allowed values are the same as for the `FZALGOR` keyword. The `FZALGn` keyword takes precedence over the `FZALGOR` keyword in determining which algorithm to use for a particular column if both keywords are present. If the column cannot be compressed with the requested algorithm (e.g., if it has an inappropriate data type), then a default compression algorithm will be used instead.

4 Keywords in the Compressed Table

With only a few exceptions, all the keywords from the uncompressed table are copied verbatim, in order, into the header of the compressed table. The header keywords remain uncompressed for ease of access. Note in particular that the values of the reserved column descriptor keywords `TTYPEn`, `TUNITn`, `TSCALn`, `TZEROn`, `TNULLn`, `TDISPn`, and `TDIMn`, as well as all the column-specific WCS keywords defined in the FITS standard, have the same values in both the original and in the compressed table, with the understanding that these keywords apply to the uncompressed data values.

The only keywords that are not copied verbatim from the uncompressed table header to the compressed table header are the mandatory `NAXIS1`, `NAXIS2`, `PCOUNT`, and `TFORMn` keywords, and the optional `CHECKSUM`, `DATASUM`, and `THEAP` keywords. These keywords must necessarily describe the contents of the compressed table itself. The original values of these keywords in the uncompressed table are stored in a new set of reserved keywords in the compressed table header. The complete set of keywords that have a reserved meaning within the header of a tile-compressed binary table are listed below:

- `ZTABLE` (required keyword). The value field of this keyword shall contain the logical value T. This indicates that the FITS binary table extension contains a tile-compressed binary table.
- `ZNAXIS1` (required keyword). The value field of this keyword shall contain an integer that gives the value of the `NAXIS1` keyword in the original uncompressed FITS table header. This represents the width in bytes of each row in the uncompressed table.
- `ZNAXIS2` (required keyword). The value field of this keyword shall contain an integer that gives the value of the `NAXIS2` keyword in the original uncompressed FITS table header. This represents the number of rows in the uncompressed table.
- `ZPCOUNT` (required keyword). The value field of this keyword shall contain an integer that gives the value of the `PCOUNT` keyword in the original uncompressed FITS table header.
- `ZFORMn` (required indexed keywords). These required array keywords supply the character string value of the corresponding `TFORMn` keyword that defines the data type of the column in the original uncompressed FITS table.
- `ZTHEAP` (optional keyword). The value field of this keyword shall contain an integer that gives the value of the `THEAP` keyword if present in the original uncompressed FITS table header. In practice, this keyword is rarely used.
- `ZTILELEN` (required keyword). The value of this keyword shall contain an integer representing the number of rows of data from the original binary table that are contained in each tile of the compressed table. The number of rows in the last tile may be less than in the previous tiles. Note that if the entire table is compressed as a single tile, then the compressed table will only contains a single row, and the `ZTILELEN` and `ZNAXIS2` keywords will have the same value.
- `ZCTYPn` (required indexed keywords). The value field of these keywords shall contain a character string giving the mnemonic name of the algorithm that was used to compress column `n` of the table. The current allowed values are `GZIP_1`, `GZIP_2`, and `RICE_1`, and the corresponding algorithms are described in Section 5.
- `ZHECKSUM` (optional keyword). The value field of this keyword shall contain a character string that gives the value of the `CHECKSUM` keyword in the original uncompressed FITS table header.
- `ZDATASUM` (optional keyword). The value field of this keyword shall contain an integer that gives the value of the `DATASUM` keyword in the original uncompressed FITS table header.

5 Supported Compression Algorithms

This section describes the currently supported compression algorithms. Other compression algorithms may be added in the future.

5.1 GZIP_1

This lossless compression algorithm is designated by the keyword `ZCTYPn = 'GZIP_1'`. Gzip is the compression algorithm used in the widely distributed GNU free software utility of the same name. It was created by Jean-loup Gailly and Mark Adler. It is based on the DEFLATE algorithm, which is a combination of LZ77 and Huffman coding. Further information about this compression technique is readily available on the Web. The "gzip -1" option is generally used which significantly improves the compression speed with only a small loss of compression efficiency.

It is important to note that any numerical data values must be arranged in big-endian byte order (the FITS standard) before the array of bytes is compressed.

5.2 GZIP_2

This lossless compression algorithm is designated by the keyword `ZCTYPn = 'GZIP_2'`. This algorithm is a variation of the `GZIP_1` algorithm in which the bytes in the arrays of numeric data columns are preprocessed by shuffling them so that they are arranged in order of decreasing significance before being compressed. For example, a 5-element array of 2-byte (16-bit) integer values, with an original big-endian byte order of

```
A1 A2 B1 B2 C1 C2 D1 D2 E1 E2,
```

will have the following byte order after shuffling the bytes:

```
A1 B1 C1 D1 E1 A2 B2 C2 D2 E2.
```

where A1, B1, C1, and D1 are the most significant bytes from each of the integer values. Byte shuffling can only be performed for numeric binary table columns that have `TFORMn` data type codes of `I, J, K, E, D, C,` or `M`. The bytes in columns that have a `L, X,` or `A` type code are never shuffled.

This byte-shuffling technique has been shown to be especially beneficial when compressing floating-point values because the bytes containing the exponent and the most significant bits of the mantissa are often similar for all the floating point values in the array. Thus these repetitive byte values generally compress very well when grouped together in this way. HDF Group has used this byte-shuffling technique when compressing HDF5 data files (HDF 2000).

5.3 RICE_1

This lossless compression algorithm is designated by the keyword `ZCTYPn = 'RICE_1'` and may only be applied to integer data type columns that have `TTYPEn` data type code values of 'B', 'I', or 'J'. The Rice algorithm (Rice, 1993) is very simple and fast. It requires only enough memory to hold a single block of 32 integers at a time and is able to adapt very quickly to changes in the input array statistics.

6 Compressing Variable-Length Array Columns

Compression of binary tables that contain variable-length array (VLA) columns (with a `P` or `Q` data type code) requires special consideration because the data values in these columns are not stored directly in the table, but instead are stored in what is called the 'data heap' which follows the main table. The VLA column in the main data table itself only contains a 'descriptor', which is composed of 2 integers that give the size and location of the actual array in the heap. When compressing a variable length array column, one must first process each individual VLA in turn by reading it from the uncompressed table, compressing it, then writing the compressed bytes to the heap in the compressed table. The descriptors that point to these compressed VLAs must be stored in a temporary array of descriptors that has been allocated for this purpose. Once all the individual VLAs in the column have been processed, that temporary array of descriptors is then itself compressed with GZIP_1, and then finally written into the heap of the compressed table.

There is one other complexity that must be addressed when dealing with VLA columns: one needs to know the original descriptor values to be able to write the uncompressed VLAs back into the same location in the heap as in the original uncompressed table. For this reason, we concatenate the array of descriptors from the uncompressed table onto the end of the temporary array of descriptors (to the compressed VLAs in the compressed table) before the 2 combined arrays of descriptors are compressed and written into the heap in the compressed table.

When uncompressing a VLA column, 2 stages of uncompression must be performed: First, the combined array of descriptors must be uncompressed, then these descriptors are used one by one to read the compressed VLA from the compressed table, uncompress it, and then write it back into the correct location in the uncompressed table. Note also that the descriptors to the compressed VLAs are always 64-bit Q-type descriptors, but the descriptors from the original uncompressed table may be either Q-type or P-type.

The following example illustrates how this works in practice: suppose one compresses a 100 row table containing a column of 2-byte integer variable length arrays (with `TFORMn = '1PI'`). When compressing this column, each of the 100 individual VLAs are read from the uncompressed table, compressed with the appropriate algorithm, and then written to the corresponding `TFORMn = '1QB'` column in the compressed table. After all the VLAs have been processed, the array of 100 P-type descriptors from the uncompressed table are concatenated onto the end of the temporary array of 100 'Q-type descriptors from the compressed table, and this combined array is compressed with the GZIP_1 algorithm and written into the compressed table.

References

HDF 2000, "Performance Evaluation Report: gzip, bzip2 compression with and without shuffling," `http://www.hdfgroup.org/HDF5/doc_resource/H5Shuffle_Perf.pdf`

Rice, R. F., Yeh, P.-S., and Miller, W. H. 1993, in Proc. of the 9th AIAA Computing in Aerospace Conf., AIAA-93-4541-CP, American Institute of Aeronautics and Astronautics

White, R. L., Greenfield, P., Pence, W., Tody, D., and Seaman, R. 2009, "Tiled Image Compression Convention", `http://fits.gsfc.nasa.gov/registry/tilecompression.html`