Specifications for Storing Compressed Images in FITS Binary Tables

Richard L. White, STScI
Perry Greenfield, STScI
William Pence, NASA/GSFC
Doug Tody, NOAO

October 21, 1999

1 General Description

This document describes a convention for compressing n-dimensional images and storing the resulting byte stream in a variable-length column in a FITS binary table. The general file structure outlined here is independent of the specific data compression algorithm that is used. The implementation details for several commonly used compression algorithms are described in the appendixes of this document.

The general principle used in this convention is to first divide the n-dimensional image into a rectangular grid of subimages or 'tiles'. Each tile is then compressed as a continuous block of data, and the resulting compressed byte stream is stored in a row of a variable length column in a FITS binary table. By dividing the image into tiles it is generally possible to extract and uncompress subsections of the image without having to uncompress the whole image. The default tiling pattern treats each row of a 2-dimensional image (or higher dimensional cube) as a tile, such that each tile contains `NAXIS1` pixels. Any other rectangular tiling pattern may be defined using the `ZTILEn` keywords that are described below. In the case of relatively small images it may be sufficient to compress the entire image as a single tile, resulting in an output binary table with 1 row. In the case of 3-dimensional data cubes, it may be advantageous to treat each plane of the cube as a separate tile if application software typically needs to access the cube on a plane by plane basis.

2 Keywords

The following keywords are defined by this convention for use in the header of the FITS binary table extension to describe the structure of the compressed image.

- `ZIMAGE` (required keyword) This keyword must have the logical value T. It indicates that the FITS binary table extension contains a compressed image, and that logically this extension should be interpreted as an image and not as a table.

- `ZCMPTYPE` (required keyword) The value field of this keyword shall contain a character string giving the name of the algorithm that must be used to decompress the image. Currently, values of `GZIP_1`, `RICE_1`, `PLIO_1`, and `HCOMPRESS_1` are reserved, and the corresponding algorithms are described in the appendixes of this document.

- `ZBITPIX` (required keyword) The value field of this keyword shall contain an integer that gives the value of the `BITPIX` keyword in the uncompressed FITS image.

- `ZNAXIS` (required keyword) The value field of this keyword shall contain an integer that gives the value of the `NAXIS` keyword in the uncompressed FITS image.

- `ZNAXISn` (required keywords) The value field of these keywords shall contain a positive integer that gives the value of the `NAXISn` keywords in the uncompressed FITS image.
- `ZTILEn` (optional keywords) The value of these indexed keywords (where `n` ranges from 1 to `ZNAXIS`) shall contain a positive integer representing the number of pixels along axis `n` of the compression tiles. All the pixels within each tile are compressed as a contiguous data array and stored in a row of a variable-length vector column in the binary table. The size of each image dimension (given by `ZNAXISn`) is not required to be an integer multiple of ZTILEn, and if it is not, then the last tile along that dimension of the image will contain fewer image pixels than the other tiles. If the `ZTILEn` keywords are not present then the default 'row by row' tiling will be assumed such that `ZTILE1 = ZNAXIS1`, and the value of all the other `ZTILEn` keywords equals 1.

  The compressed image tiles are stored in the binary table in the same order that the first pixel in each tile appears in the FITS image; the tile containing the first pixel in the image appears in the first row of the table, and the tile containing the last pixel in the image appears in the last row of the binary table.
- `ZNAMEn and ZVALn` (optional keywords) These pairs of optional array keywords (where n is an integer index number starting with 1) supply the name and value, respectively, of any algorithm-specific parameters that are needed to compress or uncompress the image. The value of `ZVALn` may have any valid FITS datatype. The order of the compression parameters may be significant, and may be defined as part of the description of the specific decompression algorithm (see the appendixes for examples).
- `Other Keywords` The binary table header may contain any additional keywords to provide information about the image. In general, all the keywords in the header of the FITS image will be copied verbatim into the header of the compressed binary table and these keywords will have the same meaning in the binary table as they did in the image. The mandatory `BITPIX`, `NAXIS`, and `NAXISn` keywords are not copied and are instead replaced by the `ZBITPIX`, `ZNAXIS`, and `ZNAXISn` keywords as described above.

3 Columns

The following columns in the FITS binary table are defined by this convention. The order of the columns in the table is not significant. The column names (given by the `TTYPEn` keyword) are shown here in upper case letters, but the case is not significant.

- `COMPRESSED_DATA` (required column) Each row of this variable-length column contains the byte stream that was generated as a result of compressing the corresponding image tile. The datatype of the column (as given by the `TFORMn` keyword) will generally be either `'1PB'`, `'1PI'`, or `'1PJ'`, depending on whether the compression algorithm generates an output stream of 8-bit bytes, 16-bit integers, or 32-bit integers, respectively. If it is not possible to efficiently compress a particular image tile, then the `COMPRESSED_DATA` vector in the corresponding row will have a length of zero, and the uncompressed tile pixels will be written instead to the `UNCOMPRESSED_DATA` column described below.
- `UNCOMPRESSED_DATA` (optional column) This variable length column will contain the uncompressed pixels for any tiles that cannot be compressed. The datatype of this column will usually correspond to the datatype of the original image as shown in the following table:

Datatype	BITPIX	TFORMn
byte	8	'1PB'
short int	16	'1PI'
long int	32	'1PJ'
float	-32	'1PE'
double	-64	'1PD'

If all the tiles in an image are compressed, then the UNCOMPRESSED_DATA column is not required.

- ZSCALE and ZZERO (optional columns) These columns give the linear scale factor and zero point offset which may be needed to transform the raw uncompressed values back to the original image pixel values (or at least a close approximation to the original values) using the following formula:

  $$\text{image\_pixel\_value} = \text{uncompressed\_value} * \texttt{ZSCALE + ZZERO}$$

  ZSCALE and ZZERO generally have double precision values and have default values of 1.0 and 0.0, respectively. If the same values of ZSCALE and ZZERO apply to every tile in the image, then they may be given as header keywords rather than as table columns.

  ZSCALE and ZZERO are typically used to scale floating point images (with BITPIX = -32 or -64) into integers before compression, since most compression algorithms are not very efficient with floating point data. See appendix A for a description of one particularly effective scaling algorithm.

  These 2 parameters should not be confused with the reserved BSCALE and BZERO keywords which may be present in integer FITS images (which have BITPIX = 8, 16, or 32). Any such integer images should normally be compressed without any further scaling, and the BSCALE and BZERO keywords should be copied verbatim into the header of the binary table containing the compressed image.

- ZBLANK (optional column) In cases where floating point images are converted to integers before being compressed, this column gives the the integer value that is used to represent undefined pixels (if any) in the image. These pixels would have an IEEE NaN (Not a Number) value in the uncompressed floating point FITS image. If every tile uses the same null value, then ZBLANK may be given as a keyword instead of as a table column. If there are no undefined pixels in the image then ZBLANK is not required. If the uncompressed image has an integer datatype (ZBITPIX > 0) then the reserved BLANK keyword which already serves this purpose should be used instead of ZBLANK.

- Other Columns Any number of other columns may be present in the table to supply other parameters that relate to each image tile.

A Appendex A: Quantization algorithm

[description of the noise estimation and quantization algorithm goes here]. This algorithm is specifically used to quantize floating point images prior to compressing them with the Rice algorithm (see below), however, this same quantization algorithm could be used equally well with other integer compression algorithms.

B Appendex B: Rice algorithm

[description of the Rice decoding algorithm goes here.]

C Appendex C: IRAF PLIO algorithm

The IRAF PLIO (Pixel List I/O) algorithm was developed to store image masks in a compressed form. The performance of this encoding is very good for typical masks consisting of isolated high or low values or extended regions at the same level. The worst case performance occurs when successive pixels have different values. Even in this case the encoding will only require one word (16 bits) per mask pixel, provided either the delta intensity change between pixels is usually less than 12 bits, or the mask represents a zero floored step function of constant height. The worst case cannot exceed npix*2 words provided the mask depth is 24 bits or less.

A good compromise between storage efficiency and efficiency of runtime access, while keeping things simple, is achieved if we maintain the compressed line lists as variable length arrays of type short integer (16 bits per list element), regardless of the mask depth. A line list consists of a series of simple instructions which are executed in sequence to reconstruct a line of the mask. Each 16 bit instruction consists of the sign bit (not used at present), a three bit opcode, and twelve bits of data, i.e.:

```
+--+-----------+----------------------------+
|16|15       13|12                         1|
+--+-----------+----------------------------+
|  |  opcode   |            data            |
+--+----------------------------------------+
```

The significance of the data depends upon the instruction. The instructions currently implemented are summarized in the table below.

```
Instruction     Opcode          Description

   ZN            00        Output N zeros
   HN            04        Output N high values
   PN            05        Output N-1 zeros plus one high value
   SH            01        Set high value, absolute
   IH,DH         02,03     Increment or decrement high value
   IS,DS         06,07     Like IH-DH, plus output one high value
```

In order to reconstruct a mask line, the application executing these instructions is required to keep track of two values, the current high value and the current position in the output line. The detailed operation of each instruction is as follows:

ZN Zero the next N (=data) output pixels.

HN Set the next N output pixels to the current high value.

PN Zero the next N-1 output pixels, and set pixel N to the current high value.

SH Set the high value (absolute rather than incremental), taking the high 15 bits from the next word in the instruction stream, and the low 12 bits from the current data value.

IH,DH Increment (IH) or decrement (DH) the current high value by the data value. The current position is not affected.

IS,DS Increment (IS) or decrement (DS) the current high value by the data value, and step, i.e., output one high value.

The high value is assumed to be set to 1 at the beginning of a line, hence the IH,DH and IS,DS instructions are not normally needed for boolean masks. If the length of a line segment of constant value or the difference between two successive high values exceeds 4096 (12 bits), then multiple instructions are required to describe the segment or intensity change.

D Appendex D: HCompress algorithm

[description of the HCompress decoding algorithm goes here.]