Introduction to FITS

Jonathan McDowell

Jun 7, 1994

Who uses FITS?

FITS stands for Flexible Image Transport System. FITS was first used as a tape format in 1979 to transfer data between US and Dutch radio observatories with very different computers. It has become the astronomy-wide standard for transporting data, and is used by many different software packages and archives:

- AIPS: Radio data (IMAGE, 'Random Groups' structure)
- IRAF: export images and spectra (IMAGE).
- Various specialized optical astronomy instruments, e.g. CCD cameras, etc. (IMAGE)
- IUE final archive (IMAGE)
- HST data (IMAGE). FITS is one of the standard export formats available from the HST archive.
- Einstein and ROSAT (IMAGE, BINTABLE). The entire Einstein archive has been converted to FITS. ROSAT used FITS from the start for distributing its data. Rev 1 introduced a new choice of keywords (RDF) to support multimission applications more easily.
- All high energy missions archived or planned at Goddard: XTE, GRO, BBXRT, SAS-2, Vela, etc. (OGIP; IMAGE, BINTABLE)
- AXAF: the ASC will provide all data to Goddard in FITS format for archiving.

What is a FITS File?

- The point of FITS files is that parts of the file are in ASCII (so they're easy to interpret) and describe in a standard way the contents of the rest of the file. This makes FITS files **self-documenting**. The definition of the original FITS format is in the astronomical literature so FITS files will be decodable as long as astronomy libraries exist.
- FITS was designed for portability so the *storage* aspects of a FITS file are heavily specified. For instance, real numbers must be stored in the particular bit representation specified by the FITS standard. FITS is required to be back compatible so that old FITS files will never stop being valid FITS.
- In contrast, the layout of the data is NOT known in advance, but largely specified by information in the header. This is in contrast to most older binary data formats, e.g. Einstein XPR files where you know which byte is going to contain what. This means that there are an infinite number of specific file formats, all considered to be examples of the one 'FITS format'.

- Modern FITS files can contain lots of FITS ‘structures’. Older FITS files contained only one structure, and only supported one *kind* of structure, the IMAGE. For back compatibility, every FITS file’s first structure is an IMAGE, although the IMAGE is often zero by zero pixels. Older FITS readers may not notice the extra structures.

Details of FITS file structure

- FITS files are divided into **FITS Structures**, also called Header Data Units (HDUs).
- FITS Structures consist of ASCII **header** sections and binary (or ASCII) **data** sections.
- Both the header and data sections are made up of an integer number of FITS logical **records** which are 2880 bytes long.
- Headers consist of one or more ASCII header records.
- Header records consist of 80-byte ASCII subrecords called **cards**.
- The data section consists of zero or more data records

- Header units contain information allowing you to calculate the number of records in the corresponding data unit. This allows software to skip over the data sections without reading them. This means that in cases where the number of output data records is not known in advance, the output file must be rewound at the end to write the size information to the header - a problem for pipe I/O.
- In contrast, the header contains no information about its own length. Header units are terminated by an END card. There is also in general no information about the existence of any further HDUs. To navigate a FITS file, the following algorithm is therefore necessary:
    1. Read header records, parsing into cards until 'END' card is found.
    2. Calculate number of data records in the HDU
    3. Read or skip this number of data records
    4. Iterate, reading another header unit, unless end-of-file is reached.
- Note that the sizes of a header card and a record are commensurate, i.e. there are an integral number (36) of cards in a record. In contrast, no attempt is made to make the layout of the data in the data sections commensurate with the record size, so individual data values may be split across records.

- There are three types of structure (plus some rare or home grown ones): **IMAGE**, **TABLE**, and **BINTABLE**. The first data unit in a file must be an IMAGE. Any data unit may be null (of zero size). IMAGEs store n-dimensional matrices of data, the most common case is a picture (2-D) or a spectrum (1-D). TABLE stores tabular data in ASCII form, while BINTABLE stores tabular data in binary form.
- An IMAGE data unit is an n-dimensional array of pixels, Each pixel in an IMAGE has the same data type; allowed data types are 1 byte unsigned integers, 2 and 4 byte signed integers and 4 and 8 byte IEEE reals.
- A TABLE data unit is a series of equal length ASCII strings ('rows'). Each row is divided into unequal length substrings ('columns'); the layout is the same for each row in the table.
- A BINTABLE data unit is a series of equal length binary rows, divided into columns like the TABLE.

Examples

I now give some ASCII dumps of sample FITS files. The data sections for these files are of course in a binary format in the actual FITS files. The files are not good examples of FITS style since they contain very little header information and no comments. A better example is given in the appendix.

Here is the smallest possible valid FITS file:

```
SIMPLE  =                    T  /  FITS STANDARD
BITPIX  =                    8  /  Binary Data
NAXIS   =                    0  /  No image data array present
END
```

consisting of a header with only the absolutely mandatory hedaer cards and no data and no extensions. A somewhat more interesting file is

```
SIMPLE  =                    T  /  FITS STANDARD
BITPIX  =                  -32  /  Real Data bits per pixel
NAXIS   =                    2  /
NAXIS1  =                    2  /
NAXIS2  =                    2  /
END

1.0 0.0
0.0 1.0
```

which is a very small FITS IMAGE without any WCS coordinate information.

A minimalist FITS BINTABLE would be

```
SIMPLE  =                    T  /  FITS STANDARD
BITPIX  =                    8  /  Binary Data
NAXIS   =                    0  /
EXTEND  =                    T  /
END

XTENSION= 'BINTABLE'            /
BITPIX  =                    8  / Required for table
NAXIS   =                    2  / Required for table
NAXIS1  =                   18  / Bytes (pixels) per row
NAXIS2  =                    2  / Rows
PCOUNT  =                    0  / Required
GCOUNT  =                    1  / Required
TFIELDS =                    3  / Number of columns
EXTNAME = 'Example'             / Name of extension
TFORM1  = '1J'                  / 4 byte integer
TTYPE1  = 'CATNUM'              / Name of column
TFORM2  = '1E'                  / 4 byte real
TTYPE2  = 'Z'                   / Name of column
TFORM3  = '10A'                 / 10 bytes string
TTYPE3  = 'NAME'                / Name of column
END

CATNUM    Z          NAME

  273     0.158      PG1226+023
   10    -0.006      Tycho SNR
```

Header Cards

Types of header card

There are 8 types of header card, corresponding to 5 data types and three special cases COMMENT, HISTORY and END. Each card contains an 8 byte keyword, a value, and (except for the special cases) a comment.

```
--------------------------------------------------------------------
XTENSION= 'BINTABLE'           / Character card
SIMPLE  =                    T / Logical card
NAXIS   =                    2 / Integer card
EQUINOX =               2000.0 / Real card
POLZN   =                  4.3                -2.2 / Complex card
COMMENT This is a comment card
HISTORY History card for informational purposes
END
--------------------------------------------------------------------
```

(The standard also allows for integer complex values but we recommend these not be distinguished from real-valued complex values.) Some keywords are 'indexed keywords' consisting of a base name and an integer suffix - for example, there is a family of indexed keywords TTYPEn, such as TTYPE1, TTYPE13, etc.

Sources of Conventions

There are five main groups of conventions which control the header cards to be used.

- The FITS standard and associated documents (eg BINTABLE standard) from the NASA standards office NOST. These apply to all FITS files. They change on a sensibly long (5-10 year) timescale).
- The WCS (World Coordinate System) draft standard from NRAO. These apply to FITS files using the WCS coordinate conventions, including many files used in IRAF and AIPS.
- The OFWG Recommendations from the NASA-GSFC OGIP FITS Working Group. These apply to high energy astrophysics data files. These are new (1993) and still changing rapidly.
- The RDF format from the NASA-GSFC and SAO ROSAT teams. These apply to ROSAT Rev 1 files.
- The ASC FITS Group Recommendations, which apply to FITS files generated by the ASC. These are currently TBD.

Is This File FITS? ... etc

- If the first 80 characters start off ”SIMPLE = T ” it thinks it’s FITS.
- It really is FITS if:
  - It consists of valid header and data sections
  - You can access it as a 2880 byte direct access file
  - the MANDATORY KEYWORDS are all there
  - The number of bytes in the data section is as predicted in the header
  - The data is stored in the right IEEE bit pattern.
- All other “standards” come down to particular conventions on what header keywords have special meanings, and what specific structure designs are recognized.
- Example: the TSI extension in ROSAT Rev1 FITS files. What makes the extension FITS is that it is a BINTABLE FITS structure with header keywords that define the columns present in the data. What makes it a TSI extension is that the BINTABLE has the title TSI and contains the columns TIME, LOGICAL, FAILED, etc. What makes it RDF is the specific choice of header keywords used to describe certain quantities: ROLL_NOM for the nominal roll angle, for instance.

Reading a FITS file

If you want to dump the contents of a FITS file, use the FTOOLS task FDUMP or my own program FITSREAD.

```
/proj/jcm/bin/fitsread test.fits test.lis dump   ! Dumps the file
/proj/jcm/bin/fitsread test.fits - scan          ! Summarizes the file
iraf> fdump test.fits
```

If you want to write a program to read in values from a FITS file and manipulate them, use Bill Pence's FITSIO library or my own JCMFITS library, or just open the file as a direct access, 2880-byte unformatted file and interpret the data yourself.

Writing a FITS file is pretty easy. I append an example using the JCMFITS library, writing a FITS file with two binary table extensions.

Writing a FITS file

Suppose we want to write the simple binary table example I gave earlier. The following program writes a slightly more verbose version with an IMAGE extension as well..

```
        program fits_test
! Compile this program using:
!  f77 -u -xl -o fits_test fits_test.o -L/proj/jcm/JCM/lib -lnjcmfits -lnjcmutils
!
        integer fptr  ! Pointer to FITS file
        integer status ! I/O status
! Executable statements start here
        call jcmlib_init
! Create new file
        call open_fits_w( 'table.fit', fptr, status )
        call null_header( fptr )
        call table( fptr )
        call image( fptr )
! Close the file
        call close_fits( fptr )
        end

        subroutine null_header( fptr )
        integer fptr    ! (i) FITS file pointer
! Write the null primary header

! Write the primary header mandatory keys
        call fits_bttop( fptr )
! Write a comment card
        call fits_wcmt( fptr, 'Test Binary Table file')
! Close out the primary header
        call fits_ehead( fptr )

        end


        subroutine table( fptr )
        integer fptr    ! (i) FITS file pointer
        integer row
```

```
! Declare table structures
       integer naxis1, naxis2, tfields
       parameter (tfields=3)            ! Number of columns
       parameter (naxis1 = 18)          ! Number of bytes per row
       parameter (naxis2 = 2)           ! Number of rows
       character*8 tform(tfields)       ! Type of columns
       character*8 ttype(tfields)       ! Names of columns
       character*8 tunit(tfields)       ! Units of columns
       character*32 cform(tfields)      ! Comment on type
       character cunit(tfields)         ! Comment on units
       character*32 ctype(tfields)      ! Comment on names
       character*8 tdisp(tfields)       ! Display formats
! Declare table data arrays
       integer namelen
       parameter (namelen=10)
       integer catnum(naxis2)
       real z(naxis2)
       character*(namelen) name(naxis2)
! Here we define the structure of the table
       data tform / '1J', '1E', '10A' /
       data ttype / 'CATNUM', 'Z', 'NAME' /
       data tunit / tfields*' '/
       data cform / 'Long integer','Single precision real','String'/
       data cunit / tfields*' '/
       data ctype / 'Catalog designation', 'Redshift', 'Name'/
       data tdisp / 'I6','F7.3','A10'/
! Now define the data values
       data catnum / 273, 10 /
       data z  / 0.158, -0.006 /
       data name / 'PG1226+023', 'Tycho SNR' /

! Write the table extension
!
! Write table header description
       call fits_bthead( fptr, 'Example', 'Name of extension',
    &   naxis1, tfields, tform, ttype, tunit, cform, ctype, cunit,
    &   tdisp,  naxis2 )
! Write another keyword we just invented
       call fits_wicard( fptr, 'VERSION', 2, 'Version number of file')
! Close out the table header
```

```
       call fits_ehead( fptr )
! Start writing the data
       do row = 1, naxis2
        call fits_puti( fptr, catnum(row) )
        call fits_putr( fptr, z(row) )
        call fits_putc( fptr, name(row), namelen )
       enddo
! Fill out the logical record with nulls and write buffer
       call fits_edata( fptr )
       end
!
       subroutine image( fptr )
       integer fptr    ! (i) FITS file pointer
! Write an example 2x2 image
       integer bitpix
       parameter (bitpix = -32 )  ! Real data
       integer naxis
       parameter (naxis =2 )
       integer naxisn(naxis)
       real image_array(2,2)
       integer i,j
       data naxisn / 2, 2 /    ! 2 x 2 image
       data image_array / 1.5, 2.5, 3.5, 4.5 /
! Write mandatory keys
       call fits_wpkeys( fptr, 'IMAGE', bitpix, naxis, naxisn,
    &  'Example2', 'Example of IMAGE extension')
! Write a WCS, the hard way
       call fits_wrcard( fptr, 'EQUINOX', 2000.0D0, 'J2000')
       call fits_wccard( fptr, 'RADECSYS', 'FK5', 'J2000')
       call fits_wccard( fptr, 'CTYPE1', 'RA---TAN','Projection')
       call fits_wccard( fptr, 'CTYPE2', 'DEC--TAN','Projection')
       call fits_wrcard( fptr, 'CRPIX1', 1.0, 'Reference pixel')
       call fits_wrcard( fptr, 'CRPIX2', 1.0, 'Reference pixel')
       call fits_wrcard( fptr, 'CRVAL1', 95.12, 'RA (deg)')
       call fits_wrcard( fptr, 'CRVAL2', -30.12, 'Dec (deg)')
       call fits_wrcard( fptr, 'CDELT1', -0.001, 'Deg per pixel')
       call fits_wrcard( fptr, 'CDELT2',  0.001, 'Deg per pixel')
       call fits_wccard( fptr, 'CUNIT1', 'deg', 'Unit of RA')
       call fits_wccard( fptr, 'CUNIT2', 'deg', 'Unit of Dec')
! Close the header
```

```
        call fits_ehead( fptr )
! Write the data
        do j = 1, naxisn(2)
         do i = 1, naxisn(1)
          call fits_putr( fptr, image_array(i,j) )
         enddo
        enddo
        call fits_edata( fptr )
        end
```

WORLD COORDINATE SYSTEM (WCS)

The idea of WCS is to have a convention describing in the header the coordinate system used in the data. The description defines that coordinate system in terms of other coordinate systems already known to the user (because their definitions are published). Examples: image pixels to sky RA and Dec, spacecraft mission time to UTC Julian Date.

The model for WCS is that the transformation between the coords used in the data (*pixel coords*) and the global coords (*world coords*) is **locally linear**. This means we can define an intermediate coordinate system which is just a linear transform of the pixel coords to scale and align them with the global system.

- For 1D cases, we just need to **scale** the coordinates and **register** them on the absolute system. We use the keyword xDELTn (e.g. CDELT2, TDELT4) to give the the value of the scale. We use the keyword CRPIXn or TCRPXn to select a **reference pixel** and the keyword CRVALn or TCRVLn to give the value of the world coordinates at that pixel.
- For 2D cases, we also need to **align** the coordinate systems. This can be done using a set of rotation matrix keywords.
- In cases where the linearity of the transformation is only local, we then need to define the non-linear transformation between the locally linear system and the global world coordinate system. This is done using CTYPEn and TCTYPn keywords to report

the type of transformation used. The most common one is the tangent plane mapping

```
CTYPE1='RA---TAN', CTYPE2='DEC--TAN'
```

REFERENCES

FITS Standard:
nssdca.gsfc.nasa.gov::anon_dir/fits/fits_standard.ps, users_guide.ps
WCS Draft:
fits.cv.nrao.edu::fits/documents/wcs/wcs.ps.all.Z

Original FITS papers:
Greisen, Wells, and Harten, 1980, SPIE 264, 298.
Wells, Greisen and Harten, 1981, Astron. Astrophys. Suppl., 44, 363.
Greisen and Harten, 1981. Astron Astrophys. Suppl., 44, 371.

Appendix: A FITS summary sheet

There are some simplifications in this description, and the reader is referred to the FITS Standard for the truth.

FITS Structures	
IMAGE	FITS Image header and array.
TABLE	FITS ASCII table
BINTABLE	FITS binary table

The first structure is special: it is called the Primary Array, it must be of type IMAGE, and the first card is SIMPLE instead of XTENSION. For TABLE and BINTABLE, the value of BITPIX must be 8. The number of bits in the data array following the header is always

$$NBITS = |BITPIX| \times GCOUNT \times (PCOUNT + \prod_{m=1}^{NAXIS} NAXISm)$$

FITS header cards are divided into several fields:

Bytes	Field
1-8	Keyword name
9-10	Value indicator
11-30	Value field
31-50	Imaginary part value field
11-80	Comment or string value field

Keywords are 8 characters long, contain the characters 0-9, A-Z, _, and −. Blanks (ASCII hex 20) are permitted only at the end of the keyword. Note that lower case letters are not permitted. The value indicator field is an equals sign followed by a blank, if it is present.

FITS Header card types

Type	'=' present?	Value
INTEGER	Y	Columns 11-30, right justified
REAL	Y	Columns 11-30, right justified, Fortran F or E format
LOGICAL	Y	Column 30, T or F
CHARACTER	Y	Column 11: quote (hex 27)
		Column 12-($n-1$): string value
		Column n: quote (hex 27)
		Require $20 \leq n \leq 80$
COMPLEX	Y	Column 11-30 (real part), 31-50 (Imag part)
COMMENT	N	Text in column 9-80
HISTORY	N	Text in column 9-80
END	N	Column 4-80 must be blank

Header Card	Type	Special Values	Meaning
			Mandatory cards
SIMPLE	Logical	T	FITS File (Primary array only)
XTENSION	Character	‘IMAGE’	Structure is an IMAGE
		‘BINTABLE’	Strucure is a BINTABLE
		‘TABLE’	Structure is an ASCII table
BITPIX	Integer	8	Byte array
		16	Short integer array
		32	Long integer array
		-32	Real array
		-64	Double precision array
NAXIS	Integer		Number of axes (2 for tables)
NAXIS1	Integer		Number of elements for axis 1
EXTEND	Logical	T	Primary data array may not be the only structure
PCOUNT	Integer	0	Required for tables
GCOUNT	Integer	1	Required for tables
TFIELDS	Integer		Number of table columns (tables only)
EXTNAME	Character		Name of this structure (not allowed for primary array)
END	END		Marks last header card. Data does not begin until next 2880 byte logical record.

Header Card	Type	Special Values	Meaning
		Special repeatable keywords	
COMMENT	Comment		Arbitrary ASCII text
HISTORY	History		Text describing how data was processed
		Keywords reserved by FITS standard	
DATE	Character	‘12/31/94’	UT Date structure was created
ORIGIN	Character	‘SAO’	Organization creating file
DATE-OBS	Character	‘12/31/94’	UT Date observation was made
TELESCOP	Character	‘ROSAT’	Telescope or satellite used
INSTRUME	Character	‘IPC-1’	Detector used
OBSERVER	Character	‘E. Hubble’	Name of observer or PI
OBJECT	Character	‘Barnard”s Star’	Name of object or field observed
EQUINOX	Real	2000.0	Equinox of positions (used by WCS)
AUTHOR	Character	‘W. Shakespeare’	Who compiled the data in this file
REFERENC	Character	‘J.I.R. 14, 121 (1994)’	Where this data is published
		Keywords used to define ASCII tables	
TBCOLn	Integer	1	Byte position of start of column n
TFORMn	Character	‘F8.4’	Format to read column n with
TTYPEn	Character	‘XVAL’	Name of column n
		Keywords used to define binary tables	
TTYPEn	Character	‘XVAL’	Name of column n
TFORMn	Character	‘3E’	Binary content of column n (see below)
TDISPn	Character	‘F8.4’	Format to print column n with
		Keywords reserved but deprecated by FITS standard	
BLOCKED			
EPOCH			

Binary table column types

Each column in a binary table is defined by a TFORMn keyword whose value is of the form rT, represening a vector of type T and dimension r. The dimension r is an integer (usually 1, except when the type is A (alphabetic)) and the type T is a character, defining both the number of bytes taken up by each element of the vector and the data type of the vector. Possible values are listed below:

Common BINTABLE TFORMn values		
TFORM	Bytes	Meaning
1I	2	Signed Integer
1J	4	Signed Integer
1E	4	Real
1D	8	Real
1A	1	ASCII character
1L	1	'T' or 'F'
Other BINTABLE TFORMn values		
8X	8	Bits
1C	8	Complex
1M	16	Complex
1B	1	Unsigned integer
1P	8	Reserved for variable array convention

Scaling of data values and coordinates

The IMAGE arrays contain pixel values and pixel numbers. Both may require rescaling to physical values. The TABLE data may also require rescaling. The table below lists the keywords in use in the high energy astrophysics community (specifically GSFC and SAO) to represent these rescalings. We use a linear rescaling:

$$y = y_0 + \Delta(x - x_0)$$

where y and y_0 have the units YUNIT and the name of Y is YNAME. The linear rescaling then is reprojected using a projection of type PROJ. Null values are represented for x of type real by an IEEE NaN value, and for x of type integer by a selected 'dud' integer value x_{NaN} (e.g. -99 or something). The header may also contain the maximum and minimum valid values of y, y_{max} and y_{min} to aid in displaying the data. Note that these are the extremes ALLOWED, and the particular dataset may not reach these extremes.

Quanitity	IMAGE data[1]	IMAGE axis n	TABLE data, col. n [3]	
PROJ	-	CTYPEn	-	*TCTYPn*
YNAME	-	-	TTYPEn	TTYPEn
YUNIT	BUNIT	*CUNITn*	TUNITn	*TCUNIn*
x_0	-	CRPIXn	-	*TCRPXn*
y_0	BZERO	CRVALn	TZEROn	*TCRVLn*
Δ	BSCALE	CDELTn	TSCALn	*TCDLTn*
y_{min}	DATAMIN	-	-	*TLMINn*
y_{max}	DATAMAX	-	-	*TLMAXn*
x_{NaN}	BLANK	-	TNULLn	-

Note: 1) $x_0 = 0$ is required for IMAGE data.
2) Data in italics represents keywords not in the FITS standard or WCS proposals.
3) The first set of conventions for TABLEs considers them as data, the second considers them as pixel lists. The overlap is unfortunate as there is no fundamental distinction.