Chapter 14

FITS Tapes

14.1 Overview

The principal route for getting data and images into and out of AIPS is by FITS (Flexible Image Transport System) format files. FITS is an internationally adopted medium of exchange of astronomical data and allows easy interchange of data between observatories and image processing systems. FITS also has the advantages that it is a self-defining format and that the actual bit pattern on the file is independent of the machine on which the file was written. The purpose of this chapter is to describe the general features of FITS and the details of the AIPS implementation. This chapter is not intended to be a rigorous description of the FITS standards. See the chapter on devices for information on read and writing tapes in AIPS. In addition to tapes, data can be written into FITS format files on other media; in these cases the files are considered to be byte streams organized into 2880 byte logical records.

The fundamental definition of the FITS system is given in Wells, Greisen, and Harten (1981), with an extension described in Greisen and Harten (1981). A proposed further extension is given in Harten, Grosbol, Tritton, Greisen and Wells, (1984). FITS has been adopted as the recommended medium of exchange of astronomical data by the IAU, the Working Group on Astronomical Software (WGAS) of the AAS, and comparable working groups in Europe. AIPS now also supports the proposal of these working groups for the writing of blocked FITS tapes.

Because of the great flexibility of the FITS system, many of its features have been adopted for the internal data storage format in AIPS. See the chapter on the catalog header for more details on the AIPS internal storage format.

There are three main portions of a FITS file (1) the main header, (2) the main data, and (3) any number of records containing auxiliary information. In addition, an extension of the original definition of the FITS structure allows storage of ungridded visibility data. Each of these is discussed in detail in the following sections.

14.2 Philosophy

FITS is a philosophy as much as a data format. The underlying philosophy is to provide a standardized, simple, and flexible means to transport data between computers or image processing systems. FITS is standardized in the sense that any FITS reader should be able to read any FITS image, at least to the degree that the array read is of the correct dimensions and the pixel values have at least the correct relative scaling. In addition, any FITS reader should be able to cope with any FITS format tape and, at least, skip over portions, or ignore keywords, that it doesn't understand.

The requirement of simplicity means that the implementation of FITS reading and writing should be fairly straightforward on any computer used for astronomical image processing. Simple also implies that the structure of the file should be self-defining and, to a large degree, self-documenting.

The main advantage of FITS is its flexibility. Due to the self-defining nature of the files, a large range of data transport needs are fulfilled. The introduction of new keywords gives the ability to add new pieces of information as needed, and the use of generalized extension files allows almost unlimited flexibility in the type of information to be stored. Thus, FITS can grow with the needs of the Astronomical community.

The great flexibility of FITS is a potential weakness as well as a strength. There is a great temptation to proliferate keywords and new extension file types. This should be done with great caution. Since FITS is a worldwide medium of data exchange, there needs to be coordination of keywords and extension files to prevent duplication and inconsistencies in usage.

The most fundamental philosophical ideal of FITS is that no change in the system should render old tapes illegal or unreadable. This philosophy is reflected in the AIPS implementation of FITS in that all obsolete implementations (e.g., old CLEAN component or antenna extension files) are trapped and processed in the most accurate manner possible.

14.3 Image Files

The most common form of astronomical information is the image and historically the first FITS tape files were for multi-dimensional images. The following sections describe FITS image files.

14.3.1 Overall Structure

The structure of a FITS image file consists of one or more records containing ASCII header information followed by one or more binary data records. (These may be followed by other records which are discussed in another section.)

All "logical" records on FITS files are 2880 8-bit bytes long, with one or more records per tape block. Blocking factors from 1 through 10 are now allowed by international agreement and are supported since the 15APR87 release of AIPS. The number of bits in a FITS record is an even multiple of words and bytes on any computer ever sold commercially. The definition of FITS allows standard ANSI labeled tapes, but the AIPS implementation only writes unlabeled tapes. Labeled tapes may be read by AIPS, but verbs like AVFILE require the user to take the label files into account.

Each FITS header record contains 36 80-byte "card images" written in 7-bit ASCII (sign bit set to zero). These header records contain all the information necessary to read, and hopefully, label the image. In addition, other information including the processing history may be given.

Following the header records come the data records. These records contain the pixel values in one of several binary formats.

14.3.2 Header Records

Each "card image" in the header is in the form,

```
keyword = value                / comment
```

Keywords should be no more than 8 characters long and the keyword = value should be readable by Fortran 77 list-directed I/O. To accommodate more primitive systems, a fixed format is mandatory for the required keywords and suggested for the optional keywords. This fixed format is as follows:

1. Keyword name beginning in column 1.
2. "=" in column 9
3. T or F (logical true or false) in column 30.
4. Real part (integer or floating) right justified, ending in column 30.
5. Imaginary part (integer or floating) right justified, ending in column 50.
6. Character string with a beginning " ' " in column 11 and an ending " ' " in or after column 20

The first keyword in a header must be SIMPLE and have a value of T (true), if the file conforms to FITS standards, and an F (false), if it doesn't. (The ASCII string "SIMPLE = T" occupying the first 30 bytes of a file of 2880-byte records is the "signature" of FITS.) The keywords and values must convey the size of the image and the number of bits per pixel value. Optionally, the coordinate system, scaling and other information may be given. In the AIPS implementation, a considerable amount of information is given.

Keywords

The following keywords (data type) are *required* for ALL FITS files (for all time) in the order given.

1. SIMPLE (logical) says if the file conforms to FITS standards.
2. BITPIX (integer) is the number of bits used to represent the pixel value; 8 => 8 bit unsigned integers, 16 => 16 bit, twos complement signed integers, 32 => 32 bit, twos complement signed integers, -32 => IEEE 32 bit floating point values, -64 => IEEE 64 bit floating point values.
3. NAXIS (integer) is the number of axes in the array.
4. NAXIS1 (integer) is the number of pixels on the fastest varying axis.
5. up to NAXIS999 (integer) is the number of pixels on the 999 th fastest varying axis.
6. END — the last keyword *must* be END. The last header record should be blank filled past the END keyword.

AIPS routines can accept up to 7-dimensional images. If a tape might contain more than one logical record per tape block, then there is an additional required keyword. Since it can do no harm, it should always be included; it has no meaning for non-tape FITS files. It is

1. BLOCKED (logical) states whether a tape may be blocked with more than one logical record per tape block.

The following optional keywords were suggested by Wells *et. al.* (1981). Their order (between the required keywords and the END keyword) is arbitrary; in general, all of these keywords appear in an AIPS FITS header.

1. BSCALE (floating) is the scale factor used to convert tape pixel values to true values (true = [tape ∗ BSCALE] + BZERO).
2. BZERO (floating) is the offset applied to true pixel values (see BSCALE).
3. BUNIT (character) gives the brightness units.
4. BLANK (integer) is the tape pixel value assigned to undefined pixels.
5. OBJECT (character) is the image name.
6. DATE (character) is the date the file was written ('dd/mm/yy')
7. DATE-OBS (character) is the date of data acquisition ('dd/mm/yy').
8. ORIGIN (character) is the tape writing institution.
9. INSTRUME (character) is the data acquisition instrument.
10. TELESCOP (character) is the data acquisition telescope.
11. OBSERVER (character) is the observer name / identification.
12. blank in col 1-8 (none) means columns 9 - 80 are a comment.
13. COMMENT (none) means columns 9 - 80 are a comment.

14. HISTORY (none) means columns 9 - 80 are a comment.
15. CRVALn (floating) is the value of physical coordinate on axis n at the reference pixel.
16. CRPIXn (floating) is the array location of reference pixel along axis n. CRPIX may be a fractional pixel and/or be outside of the limits of the array.
17. CDELTn (floating) is the increment in physical coordinate along axis n as the array index increases by 1.
18. CTYPEn (character) is the type of physical coordinate on axis n.
19. CROTAn (floating) is the rotation angle of actual axis n from stated coordinate type.
20. DATAMAX (floating) is the maximum data value in file (after scaling).
21. DATAMIN (floating) is the minimum data value in file.
22. EPOCH (floating) is the epoch of coordinate system (years).

Of these keywords, all are well defined except the rotation; see the chapter on the catalog header for more details on the current AIPS rotation conventions. AIPS routines can currently read up to 32768 header records each consisting of 36 card images.

History

In the AIPS implementation, the "HISTORY" cards contain the entries of the history file associated with the image. As they appear on the tape, these history entries are in the form:

```
HISTORY  tsknam keyword1=value1, keyword2=value2 ...  / comment
```

where "tsknam" is the name of the task (or AIPS) making the entry and the keywords are the AIPS adverbs used. Thus, these history records may be used to carry AIPS-specific values which don't have official keywords. This feature is used, for example, to determine the default file name, class, etc. when reading a file which was written on an AIPS system.

AIPS Non-standard Image File Keywords

There are a number of keywords used by AIPS which are not standard.

1. TABLES (integer) is the number of tables following the file. (now obsolete)
2. DATE-MAP (character) is the date the map was made. ('dd/mm/yy')
3. OBSRA (floating) is the Right ascension of the antenna and delay tracking position used for the observations.
4. OBSDEC (floating) is the declination of the antenna and delay tracking position used for the observations.
5. VELREF (floating) is the reference velocity.
6. ALTRVAL (floating) is the value of the alternate (frequency/velocity) axis at the alternate reference pixel (ALTPIX).
7. ALTRPIX (floating) is the alternate (frequency/velocity) reference pixel.
8. RESTFREQ (floating) is the rest frequency of the spectral line being observed.
9. XSHIFT (floating) is the offset of the phase center from the tangent point of the Right ascension after any rotation.

10. YSHIFT (floating) is the offset of the phase center from the tangent point of the declination after any rotation.

A number of keywords which are specific to AIPS are hidden on HISTORY cards. These keywords are recognized if the first symbol in columns 10 - 17 is one of the following: 'AIPS', 'VLACV', or 'RANCID'.

1. IMNAME (character) the name of the file in an AIPS (or RANCID) system used to generate the FITS tape.
2. IMCLASS (character) the class of the AIPS file.
3. IMSEQ (integer) the sequence number of the AIPS file.
4. USERNO (integer) the AIPS user number.
5. PRODUCT (integer) the type of CLEAN image. 1=>normal clean, 2=>components, 3=>residual, 4=>points.
6. NITER (integer) the number of CLEAN components used for the image.
7. BMAJ (floating) the major axis (FWHP) of the restoring beam. (degrees)
8. BMIN (floating) the minor axis (FWHP) of the restoring beam.
9. BPA (floating) the position angle (from north through east) of the major axis of the restoring beam.

AIPS also recognizes, but does not write, the following non-standard keywords:

1. OPHRAE11 (floating) an obscure number related to the Right ascension of the center on an image made on the VLA pipeline PDP11.
2. OPHDCE11 (floating) an obscure number related to the declination of the center on an image made on the VLA pipeline PDP11.
3. MAPNAM11 (character) the name of the file on the VLA pipeline PDP11.

Any keywords which are not recognized by AIPS are written into the history file.

Coordinate Systems

The coordinate type and the system used for each type is given by the CTYPEn values. The character strings used for these values are identical to the strings used in the AIPS catalog header record (CAT4(K4CTP+n-1)). The coordinate type is encoded into the first 4 characters of the coordinate type string (e.g., 'RA–' indicating Right ascension) and the system used is encoded into characters 5 - 8 (e.g., '-SIN' indicating a sine projection onto the sky). The coordinate systems and their symbolic names are described in detail in the chapter on the catalog header and AIPS Memo Numbers 27 and 46. The coordinate system used to describe the polarization of an image needs careful attention.

The AIPS convention for projected geometries is to specify the tangent point of the projection as the reference pixel, even though this need not correspond to an integer pixel and need not even be contained in the array given. The tangent point is the position on the sky where the plane on which the image is projected is tangent to the celestial sphere. For images derived from synthesis arrays, this is the position for which u, v, and w were computed. The reference pixel for a synthesis array beam image is the phase reference of the image; this should be the position of the peak of the beam (pixel value = 1.0).

The use of one rotation angle per axis cannot be used to define a general rotation of the axis system. Since the AIPS catalog header uses the same convention, the same problems occur internally to AIPS. See the chapter on the AIPS catalog header for a brief discussion of the conventions used in AIPS. The same conventions are used when reading and writing FITS tapes.

Example Image Header

The following is an example of an image header written by AIPS (with most of the HISTORY entries removed).

```
000000000111111111122222222223333333333444444444455555555556666666666
123456789012345678901234567890123456789012345678901234567890123456789
SIMPLE  =                    T /
BITPIX  =                   16 /
NAXIS   =                    4 /
NAXIS1  =                 2048 /
NAXIS2  =                 1024 /
NAXIS3  =                    1 /
NAXIS4  =                    1 /
EXTEND  =                    T / Tables following main image
BLOCKED =                    T / Tape may be blocked
OBJECT  = '3C405   '           / Source name
TELESCOP= '        '           /
INSTRUME= '        '           /
OBSERVER= 'PERL    '           /
DATE-OBS= '27/10/82'           /Observation start date dd/mm/yy
DATE-MAP= '14/07/83'           /Date of last processing dd/mm/yy
BSCALE  =    7.04625720812E-05 /Real = Tape * BSCALE + BZERO
BZERO   =    2.18688869476E+00 /
BUNIT   = 'JY/BEAM '           /Units of flux
EPOCH   =      1.950000000E+03 /Epoch of RA, Dec
DATAMAX =      4.495524406E+00 /Max pixel value
DATAMIN =     -1.217470840E-01 /Min pixel value
CTYPE1  = 'RA---SIN'           /
CRVAL1  =    2.99435165226E+02 /
CDELT1  =     -4.166666986E-05 /
CRPIX1  =      1.024000000E+03 /
CROTA1  =      0.000000000E+00 /
CTYPE2  = 'DEC--SIN'           /
CRVAL2  =    4.05961940065E+01 /
CDELT2  =      4.166666986E-05 /
CRPIX2  =      5.130000000E+03 /
CROTA2  =      0.000000000E+00 /
CTYPE3  = 'FREQ    '           /
CRVAL3  =    4.86635000000E+09 /
CDELT3  =      1.250000000E+07 /
CRPIX3  =      1.000000000E+00 /
CROTA3  =      0.000000000E+00 /
CTYPE4  = 'STOKES  '           /
CRVAL4  =    1.00000000000E+00 /
CDELT4  =      1.000000000E+00 /
CRPIX4  =      1.000000000E+00 /
CROTA4  =      0.000000000E+00 /
HISTORY UVLOD /DATA BASE CREATED BY USER 76 AT 14-JUL-1983 10:17:08
HISTORY UVLOD OUTNAME='CYGA        ' OUTCLASS='XY    '
HISTORY UVLOD OUTSEQ=    1   OUTDISK=  3
        ...
ORIGIN  = 'AIPSNRAO VLA VAX3       '    /
DATE    = '19/08/83'                    / TAPE WRITTEN ON DD/MM/YY
HISTORY AIPS   IMNAME='CYGA         ' IMCLASS='IMAP  ' IMSEQ=   1   /
```

```
HISTORY AIPS   USERNO=   76            /
END
```

Units

The units for pixel values and coordinate systems should be SI units where appropriate (e.g., velocities in meters/sec); angles in degrees; pixel values in Jy, Jy/beam, magnitudes, or magnitudes/pixel.

14.3.3 Data Records

The data array starts at the beginning of the record following the last header record. The data occurs in the order defined by the header - in increasing pixel number, with axis 1 the fastest varying and the last axis defined the slowest varying. Data is packed into the 2880 byte records with no gaps; that is, the first pixel of any given axis does not necessarily appear in the first word of a new record.

The bits in each word are in order of decreasing significance with the sign bit first. This convention means the PDP-11 and VAX machines will have to reverse the order of the bytes in 16- and 32-bit words before writing, or after reading, the tape. There are a number of AIPS utility routines for converting FITS tape data to the local convention; these are briefly described in the following list. Complete details of the call sequences etc. are given at the end of the chapter on the Z routines.

1. ZCLC8 converts local characters to standard 8-bit ASCII.
2. ZC8CL extracts 8-bit standard characters from a buffer and stores them in the local character form.
3. ZI16IL extracts 16-bit twos complement integers from a buffer and puts them in a local integer array.
4. ZI32IL extracts 32-bit twos complement integers from a buffer and puts them in a local array of integers.
5. ZI8IL converts 8-bit unsigned binary numbers to local integers.
6. ZILI16 converts a buffer of local integers to a buffer of standard 16-bit, twos complement integers.
7. ZRLR32 converts local single precision floating point values to IEEE 32 bit values.
8. ZRLR64 converts local double precision floating point values to IEEE 64 bit values.
9. ZR32RL converts IEEE 32 bit floating point values to local single precision.
10. ZR64RL converts IEEE 64 bit floating point values to local double precision.
11. ZR8P4 converts between IBM format integers and double precision .

14.4 Random Group (UV data) Files

The extension of the original FITS standards described by Greisen and Harten (1981) allows uv data to be written in FITS files. These files are called “Random group” FITS files. This extension is to allow multiple “images,” i.e., rectangular data arrays, each of which is arbitrarily located on some “axes”. Thus, each data array is preceded by a number of “random” parameters which describe its location on axes on which it is not regularly gridded, e.g., u, v, w, time, and baseline. The definition of what constitutes an “axis” is extremely vague. Currently, AIPS FITS routines can accept up to 7 actual axes in the regular portion of a group and up to 20 random parameter words; the AIPS catalog header has space for 14 random parameter labels. The structure of a group is shown in the following.

```
| r1, r2, r3, ... rk | p11, p12, ... pmn |

          where r1 ... rk are random parameters 1 through k
                p11 ... pmn are the pixel values in the order
                     defined for image arrays.  Two dimensions
                     are used only for demonstration.
```

FITS image files are actually a subset of this more general structure, but, for historical reasons, the random group FITS is treated as a special case of the image file. This has unfortunate consequences as will shortly become obvious. Most of the features of random group files are identical to image files and the discussion in the following section will concern the differences between image and random group FITS files.

AIPS uv data files will generally have a number of associated extension tables. The FITS format for these tables is described later in this chapter; the details of the tables themselves are given in the chapter on calibration and editing.

14.4.1 Header Record

For obscure historical reasons, random group FITS files are declared to have zero pixels on the first axis; the first real axis is labeled axis 2 and so on. This will allow FITS image readers that don't know about random group files to do something reasonable, i.e., skip over the file. Thus a random group FITS file has one more axis described in the header than actually occurs in the data.

In addition to playing games with the axis numbers, random group FITS headers have the following required keywords (in any order):

1. GROUPS (logical) is true (T) if the data file is a random group FITS file.
2. PCOUNT (integer) is the number of random parameters preceding each data array.
3. GCOUNT (floating) is the number of groups in the file.

The random parameters may be labeled and scaled in a fashion similar to image axes and pixels. In addition, multiple-word precision in some of the random parameters is allowed by giving multiple random parameters the same label. If several random parameters have the same name (PTYPE), their values should be summed after scaling. Labeling and scaling use the following optional keywords (arbitrary order):

1. PTYPEn (character) is the label for the n-th random parameter. If several random parameters have the same value of PTYPEn they should be summed after scaling.
2. PSCALn (floating) gives the scale factor for random parameter n. True_value = tape_value * PSCALn + PZEROn
3. PZEROn (floating) gives the scaling offset for random parameter n.

A number of keywords, which are specific to AIPS, are hidden on HISTORY cards. These keywords are recognized if the first symbol in columns 10 - 17 is one of the following: 'AIPS', 'VLACV', or 'RANCID'.

1. SORT ORDER (character) the order of the groups.
2. WTSCAL (floating) an additional scaling factor for visibility weights.

14.4.2 Data Records

The binary data records are stored, beginning in the first record following the last header record, in much the same way that image files are stored; the beginning of a group does not necessarily correspond to the beginning of a record. The same pixel data types are allowed as for image files (note: the data type must be the same for all values, both random parameters and the "data" array).

Units of Random Parameters

The FITS conventions do not include a way of specifying the units of random parameters; however, SI units should be used where possible. The conventions used by AIPS for the random parameter types (PTYPEn) are given in the following:

1. 'UU', 'VV', 'WW': These are the spatial frequency coordinates in seconds of light travel time.
2. 'BASELINE': This is the baseline code as antenna1*256 + antenna2 + 0.01 * (array -1). Antenna numbers specify entries in the Antenna table ('AIPS AN') following the data. Each array has an antenna table; the version (EXTVER) number of the table is the array number.
3. 'DATE': The time tags for the data are kept in the form of Julian date in days.
4. 'SOURCE': The source identification number specifies an entry in the source ('AIPS SU') table, which must follow the data if this random parameter is present.
5. 'FQID': The frequency/bandwidth identification number specifies an entry in the frequency ('AIPS FQ') table, which must follow the data if this random parameter is present.

Units of the Regular Axis Coordinates

The units of the regular axis coordinates are defined by convention; the conventions used by AIPS for the regular axis types (CTYPEn) are the following:

1. 'COMPLEX': the complex axis consists of the brightness, baseline value subtracted, and (optional) weight. Magic value blanking is supported.
2. 'STOKES': this axis is used to describe which Stokes' parameters are given; the conventions are the same as used internally in AIPS. These conventions are discussed in the chapter on disk I/O.
3. 'FREQ': the frequency axis coordinates are in Hz.
4. 'IF': The IF axis is a construct which allows irregularly spaced groups of frequency channels. The IF number specifies an entry in the ('AIPS FQ') table which must follow the data if this axis is present. This table gives the offsets from the reference frequency specified by the FREQ axis.
5. 'RA' and 'DEC': the celestial coordinates are given in degrees.

Weights and Flagging

Uv FITS files written by AIPS have as their first (real, i.e., second in the header) axis the 'COMPLEX' axis which is dimensioned 3. The values along this axis (coordinate values 1, 2, and 3) are real part (in Jy), imaginary part, and (optional) weight. A non-positive weight indicates that the the visibility has been flagged. The scaling desired for the weight may be different than that for the real and imaginary parts, so an additional scaling factor is stored in the header as a HISTORY entry as follows:

```
HISTORY AIPS WTSCAL = 2.76756756757E+01
                                  / CMPLX WTS=WTSCAL*(TAPE*BSCALE+BZERO)
```

The use of WTSCAL allows the reader to recover the same values for the weights as the AIPS file which was used to generate the FITS file. If WTSCAL is ignored (or absent), the relative, but not absolute, scaling of the weights is preserved.

In addition to the form described above, AIPS will accept other forms of weighting/flagging data.

1. *Magic value blanking.* In this case, the COMPLEX axis is dimensioned 2 (real and imaginary) and the header keyword BLANK is used to indicate undefined data values. Thus, if either the real or imaginary parts are 'blanked', the data is assumed to be flagged (invalid).
2. *Random parameter flagging.* Data written on the VLA pipeline/ISIS(?) is in this format. The weights and flags are passed as random parameters but no one seems to understand the flagging bits.

Antennas and Subarrays

If data from different arrays (or different configurations of an array) are combined, the physical identity of a given antenna number may not be constant in a given data base. In order to identify the physical antennas involved in a given visibility record, AIPS uses a subarray number. The (subarray number - 1) ∗ 0.01 is added to the baseline number to identify the subarray.

There is an antenna file, or list, for each subarray. The information about the antennas (e.g., locations, etc.) is given in the antenna files. Currently, AIPS writes these files as extension table files (described later) with the file version number corresponding to the subarray number.

AIPS will also recognize antenna locations given in the HISTORY cards. An example (from Greisen and Harten 1981) of this follows:

```
COMMENT  ANTENNA LOCATIONS IN NANOSECONDS:
HISTORY VLACV ANT  N= 2 X=  5470.525 Y=-14443.276 Z= -8061.210 ST='AW4'
HISTORY VLACV ANT  N= 4 X=  1667.280 Y= -4396.334 Z= -2452.399 ST='CW8'
HISTORY VLACV ANT  N= 5 X=    37.719 Y=   135.627 Z=   -50.585 ST='DE2'
HISTORY VLACV ANT  N= 6 X=  3353.710 Y= -8816.123 Z= -4910.700 ST='BW6'
HISTORY VLACV ANT  N= 7 X=   118.761 Y=   445.786 Z=  -170.397 ST='DE4'
HISTORY VLACV ANT  N= 9 X= 10924.708 Y=-28961.684 Z=-16194.042 ST='AW6'

COMMENT  FORMULA FOR BASELINES BETWEEN ANTENNA I AND J (I<J):
COMMENT  BASELINE(IJ) = LOCATION(I) - LOCATION(J)

COMMENT  FORMULA FOR UU, VV, WW :
COMMENT  UU = BX * SIN(HA) + BY * COS(HA)
COMMENT  VV = BZ * COS(DEC) + SIN(DEC) * (BY * SIN(HA) - BX * COS(HA))
COMMENT  WW = BZ * SIN(DEC) + COS(DEC) * (BX * COS(HA) - BY * SIN(HA))
            WHERE UU AND VV ARE THEN ROTATED TO THE EPOCH
```

The above example also defines the antenna geometry and u, v, and w terms used for VLA data (-SIN projection).

Coordinates

The coordinate systems used to write FITS uv data tapes are very similar to the AIPS internal systems; the major difference being the use of 'DATE' (giving the Julian date) for time tagging the data rather than 'TIME1' (giving the time in days from the beginning of the experiment). Another difference is the use of seconds for u, v, and w in FITS, but wavelengths at the reference frequency inside AIPS. See the uv data section of the disk I/O chapter for more details of the AIPS internal uv data coordinate systems.

Sort Order

The ordering of visibility records is variable and may be changed by programs such as AIPS task UVSRT. The sort order is given as a two character code in the FITS header as in the following example:

```
HISTORY AIPS  SORT ORDER = 'XY'
```

Data sorted in AIPS has a two-key sort order with the first key varying the slower. The two keys are coded as characters given by the following table:

```
B => baseline number
T => time order
U => u spatial frequency coordinate
V => v spatial frequency coordinate
W => w spatial frequency coordinate
```

```
R => baseline length
P => baseline position angle
X => descending ABS(u)
Y => descending ABS(v)
Z => ascending ABS(u)
M => ascending ABS(v)
* => not sorted
```

14.4.3 Typical VLA Record Structure

The following is a uv FITS header for multi-source, continuum VLA data for data written in scaled 16 bit integers which demonstrates the use of multiple precision random parameters. Most of the HISTORY records are removed from this example and it has been edited for clarity. The header indicates that the data in this example is followed by extension tables.

```
00000000011111111112222222222333333333344444444445555555555666666666
12345678901234567890123456789012345678901234567890123456789012345678 9
SIMPLE  =                    T /
BITPIX  =                   16 /
NAXIS   =                    7 /
NAXIS1  =                    0 /No standard image just group
NAXIS2  =                    3 /
NAXIS3  =                    4 /
NAXIS4  =                    1 /
NAXIS5  =                    2 /
NAXIS6  =                    1 /
NAXIS7  =                    1 /
EXTEND  =                    T /Tables following main image
BLOCKED =                    T /Tape may be blocked
OBJECT  =  'MULTI   '          /Source name
TELESCOP=  'VLA     '          /
INSTRUME=  'VLA     '          /
OBSERVER=  'VC35    '          /
DATE-OBS=  '02/12/84'          /Observation start date dd/mm/yy
DATE-MAP=  '09/11/86'          /Date of last processing dd/mm/yy
BSCALE  =    4.98494155534E-05 /Real = tape * BSCALE + BZERO
BZERO   =    0.00000000000E+00 /
BUNIT   =  'JY      '          /Units of flux
EPOCH   =      1.950000000E+03 /Epoch of RA, Dec
BLANK   =               -32768 /Tape value of blank pixel
CTYPE2  =  'COMPLEX '          /
CRVAL2  =    1.00000000000E+00 /
CDELT2  =      1.000000000E+00 /
CRPIX2  =      1.000000000E+00 /
CROTA2  =      0.000000000E+00 /
CTYPE3  =  'STOKES  '          /
CRVAL3  =   -1.00000000000E+00 /
CDELT3  =     -1.000000000E+00 /
CRPIX3  =      1.000000000E+00 /
CROTA3  =      0.000000000E+00 /
CTYPE4  =  'FREQ    '          /
CRVAL4  =    1.66499989984E+09 /
CDELT4  =      5.000000000E+07 /
```

```
CRPIX4  =      1.000000000E+00 /
CROTA4  =      0.000000000E+00 /
CTYPE5  =  'IF      '          /
CRVAL5  =    1.00000000000E+00 /
CDELT5  =      1.000000000E+00 /
CRPIX5  =      1.000000000E+00 /
CROTA5  =      0.000000000E+00 /
CTYPE6  =  'RA      '          /
CRVAL6  =    0.00000000000E+00 /
CDELT6  =      1.000000000E+00 /
CRPIX6  =      1.000000000E+00 /
CROTA6  =      0.000000000E+00 /
CTYPE7  =  'DEC     '          /
CRVAL7  =    0.00000000000E+00 /
CDELT7  =      1.000000000E+00 /
CRPIX7  =      1.000000000E+00 /
CROTA7  =      0.000000000E+00 /
GROUPS  =                    T /
GCOUNT  =               14655. /
PCOUNT  =                    7 /
PTYPE1  =  'UU      '          /  u in seconds
PSCAL1  =    3.38531271081E-09 /
PZERO1  =    0.00000000000E+00 /
PTYPE2  =  'VV      '          /  v in seconds
PSCAL2  =    3.22965771440E-09 /
PZERO2  =    0.00000000000E+00 /
PTYPE3  =  'WW      '          /  w in seconds
PSCAL3  =    3.66412235782E-10 /
PZERO3  =    0.00000000000E+00 /
PTYPE4  =  'DATE    '          /  Date + time as Julian date (days)
PSCAL4  =    2.50000000000E-01 /
PZERO4  =    2.44603650000E+06 /
PTYPE5  =  'DATE    '          /
PSCAL5  =    1.52587890600E-05 /
PZERO5  =    0.00000000000E+00 /
PTYPE6  =  'BASELINE'          /  Ant1*256 + Ant2 + (subarray-1)*0.01
PSCAL6  =    1.00000000000E+00 /
PZERO6  =    0.00000000000E+00 /
PTYPE7  =  'SOURCE  '          /  Source ID number.
PSCAL7  =    1.22137404580E-04 /
PZERO7  =    0.00000000000E+00 /
         / Where BASELINE = 256*ANT1 + ANT2 + (ARRAY#-1)/100
HISTORY FILLR / IMAGE CREATED BY USER  103 AT 09-NOV-1986   15:49:35
HISTORY FILLR OUTNAME='MULTI      '   OUTCLASS='FILLR '
HISTORY FILLR OUTSEQ=    4   OUTDISK=  2
        ...
ORIGIN  = 'AIPSNRAO CVAX            15APR87'    /
DATE    = '13/11/86'                    / File written on dd/MM/YY
HISTORY AIPS   IMNAME='MULTI       ' IMCLASS='FILLR ' IMSEQ=   4     /
HISTORY AIPS   USERNO=  103            /
HISTORY AIPS   SORT ORDER = 'TB'
              / Where T means time (IAT)
              / Where B means baseline num
HISTORY AIPS WTSCAL =  1.83759533423E+00 / CMPLX WTS=WTSCAL*
```

```
                                      (TAPE*BSCALE+BZERO)
END
```

14.4.4 Single Dish Data

Observations made with filled aperature instruments are frequently made at essentially random positions on the sky, possibly using a number of offset feeds or detectors. This type of data may be convienently described using the random groups (UV FITS) format. The FITS form of this data is the same as visibility data except that the number and meaning of the random parameters are different. The celestial coordinates may be either Right Ascension and declination or projected coordinates about a specified tangent point.

A logical record consists of all data recorded from a given beam on the sky at a given time. A dummy AN table is optional.

Single Dish Random Parameters

The single dish random parameter types (PTYPEn) are described in the following:

1. 'RA' and 'DEC': These random parameters are the Right Ascension and Declination of the observation in degrees. If the coordinates have been projected onto the tangent plane then the RA and Declination types become 'RA—xxx' and 'DEC–xxx' where -xxx is the projection code. See the chapter on AIPS catalog headers and/or AIPS memoes 27 and 46 for details of the projection codes. These random parameters are required but the order is arbitrary.

2. 'DATE': The time tags for the data are kept in the form of Julian date in days. This random parameter is required but the order is optional.

3. 'BEAM': This random parameter gives the beam number + 256. This random parameter is optional. The beam offset makes the data look more like uv data and more of the the AIPS uv data tasks will work for this data.

4. 'SCAN': This random parameter gives the scan number. This random parameter is optional.

5. 'SAMPLE': This random parameter gives the sample number in the scan. This random parameter is optional.

Single Dish Regular Axis Coordinates

The units of the regular axis coordinates are defined by convention; the conventions used by AIPS for the regular axis types (CTYPEn) are the following:

1. 'COMPLEX': the complex axis consists of the real, imaginary and (optional) weight. Magic value blanking is supported. The imaginary part may be used to carry any baseline values which have been subtracted. This axis is required.

2. 'STOKES': this axis is used to describe which Stokes' parameters are given; the conventions are the same as used internally in AIPS. These conventions are discussed in the chapter on disk I/O. This axis is required.

3. 'FREQ': the frequency axis coordinates are in Hz. This axis is required.

4. 'IF': The IF axis is a construct which allows irregularly spaced groups of frequency channels. The IF number specifies an entry in the ('AIPS FQ') table which gives the offsets from the reference frequency specified by the FREQ axis. This axis is optional but if it is present, then an "FQ" table must also be present.

5. 'RA' and 'DEC': the celestial coordinates are given in degrees. The values associated with these axes are irrelevant (although they should be present) for unprojected data. For data with projected coordinates the coordinate values of these axes should be the tangent point, i.e. the position on the sky at which the plane onto which the coordinates are projected is tangent to the celestial sphere and these axes should become 'RA—ccc' and 'DEC–ccc' where ccc is the projection code. These axes are required.

Weights and flagging are handled the same as for visibility data. Sort order is the same as for visibility data except that the sort codes for sorting by u and v become:

```
U => ordered by RA
V => ordered by Declination
X => descending ABS (RA)
Y => descending ABS (Declination)
Z => ascending ABS (RA)
M => ascending ABS (Declination)
```

Example Single Dish Data File Header

The following is a FITS header for a single dish data file containing 16 frequency channels and a single IF and using unprojected coordinated. The data in this file is written in 16 bit scaled integers. The first two lines are not part of the FITS file.

```
000000000111111111122222222223333333333444444444455555555556666666666
1234567890123456789012345678901234567890123456789012345678901234567890123456789
SIMPLE  =                    T /
BITPIX  =                   16 /
NAXIS   =                    6 /
NAXIS1  =                    0 /No standard image just group
NAXIS2  =                    3 /
NAXIS3  =                    1 /
NAXIS4  =                   16 /
NAXIS5  =                    1 /
NAXIS6  =                    1 /
EXTEND  =                    T /Tables following main image
BLOCKED =                    T /Tape may be blocked
OBJECT  =  'ALL SKY '          /Source name
TELESCOP=  '300 FT  '          /
INSTRUME=  '6 CM    '          /
OBSERVER=  'PHANTOM '          /
DATE-OBS=  '02/12/86'          /Observation start date dd/mm/yy
DATE-MAP=  '09/11/87'          /Date of last processing dd/mm/yy
BSCALE  =    4.98494155534E-05 /Real = tape * BSCALE + BZERO
BZERO   =    0.00000000000E+00 /
BUNIT   =  'JY      '          /Units of flux
EPOCH   =      1.950000000E+03 /Epoch of RA, Dec
BLANK   =               -32768 /Tape value of blank pixel
CTYPE2  =  'COMPLEX '          /
CRVAL2  =    1.00000000000E+00 /
CDELT2  =      1.000000000E+00 /
CRPIX2  =      1.000000000E+00 /
CROTA2  =      0.000000000E+00 /
CTYPE3  =  'STOKES  '          /
CRVAL3  =   -1.00000000000E+00 /
```

```
CDELT3  =     -1.000000000E+00 /
CRPIX3  =      1.000000000E+00 /
CROTA3  =      0.000000000E+00 /
CTYPE4  =  'FREQ    '          /
CRVAL4  =    1.66499989984E+09 /
CDELT4  =      5.000000000E+07 /
CRPIX4  =      1.000000000E+00 /
CROTA4  =      0.000000000E+00 /
CTYPE5  =  'RA      '          /
CRVAL5  =    0.00000000000E+00 /
CDELT5  =      1.000000000E+00 /
CRPIX5  =      1.000000000E+00 /
CROTA5  =      0.000000000E+00 /
CTYPE6  =  'DEC     '          /
CRVAL6  =    0.00000000000E+00 /
CDELT6  =      1.000000000E+00 /
CRPIX6  =      1.000000000E+00 /
CROTA6  =      0.000000000E+00 /
GROUPS  =                    T /
GCOUNT  =               14655. /
PCOUNT  =                    5 /
PTYPE1  =  'RA      '          /  RA in degrees
PSCAL1  =    1.09890000000E-02 /
PZERO1  =    0.00000000000E+00 /
PTYPE2  =  'DEC     '          /  Declination in degrees
PSCAL1  =    1.09890000000E-02 /
PZERO2  =    0.00000000000E+00 /
PTYPE3  =  'BEAM    '          /  Beam number
PSCAL3  =    3.66412235782E-10 /
PZERO3  =    0.00000000000E+00 /
PTYPE4  =  'DATE    '          /  Date + time as Julian date (days)
PSCAL4  =    2.50000000000E-01 /
PZERO4  =    2.44603650000E+06 /
PTYPE5  =  'DATE    '          /
PSCAL5  =    1.52587890600E-05 /
PZERO5  =    0.00000000000E+00 /
ORIGIN  = 'AIPSNRAO CVAX            15APR87'    /
DATE    = '13/11/87 '                     / File written on dd/mm/yy
HISTORY AIPS WTSCAL =  1.83759533423E+00 / CMPLX WTS=WTSCAL*
                                           (TAPE*BSCALE+BZERO)
END
```

14.5 Extension Files

There is frequently auxiliary information associated with an image or data set which needs to be saved in the same tape file. Examples of this in AIPS are the Antenna files and CLEAN component files. There is currently a draft proposal to the IAU (Harten *et. al.* 1984) defining a standard format for the invention of extension files to be written after the main data records (if any) and defining a "Tables" type extension file. The tables extension files will be able to carry information which can be expressed in the form of a table. AIPS also makes use of a 3-D Table extension which is similar to tables but allows arrays as table entries. The following section will describe the proposed standards which are being incorporated into AIPS.

14.5.1 Standard Extension

The standard, generalized extension file is not a true tape file in the sense that it is separated by tape EOF marks, but is a number of records inside a FITS file which contains information of relevance to the file. Each standard extension "file" will have a header which is very similar to the main FITS header. This header consists of one or more 2880 8-bit byte "logical" records each containing 36 80-byte "card images" in the form:

```
keyword = value    / comment
```

The extension file header begins in the first record following the last record of main data (if any) or following the last record of the previous extension file. The format of the generalized extension "file" header is such that a given FITS reader can decide if it wants (or understands) a given extension file type and can skip over the extension file if the reader decides it doesn't.

Most of the standards concerning data types and bit orders for the main FITS data records also apply to extension files. One difference is that 8-bit pixel values can be used to indicate ASCII code.

The use of the generalized extension "files" requires the use of a single additional keyword in the main header:

1. EXTEND (logical) if true (T) indicates that there *may* be extension files following the data records and, if there are, that they conform to the generalized extension file header standards.

The required keywords in an extension file header record are, in order:

1. XTENSION (character) indicates the type of extension file, this must be the first keyword in the header.
2. BITPIX (integer) gives the number of bits per "pixel" value. The types defined for the main data records plus 8-bit ASCII are allowed.
3. NAXIS (integer) gives the number of "axes"; a value of zero is allowed which indicates that no data records follow the header.
4. NAXIS1 (integer) is the number of "pixels" along the first axis (if any).
5. NAXISn (integer) is the number of "pixels" along the nth axis.
6. PCOUNT (integer) is the number of "random" parameters before each group. This is similar to the definition of random group main data records. The value may be zero.
7. GCOUNT (integer) is the number of groups of data defined as for the random group main data records. If an image-like file (e.g., a table file) is being written this will be 1.
8. END is always the last keyword in a header. The remainder of the record following the END keyword is blank filled.

There are three optional standard keywords for extension file header records. The order, between the required keywords and the END keyword, is arbitrary.

1. EXTNAME (character) can be used to give a name to the extension file to distinguish it from other similar files. The name may have a hierarchal structure giving its relation to other files (e.g., "map1.cleancomp")
2. EXTVER (integer) is a version number which can be used with EXTNAME to identify a file.
3. EXTLEVEL (integer) specifies the level of the extension file in a hierarchal structure. The default value for EXTLEVEL should be 1. AIPS has not implemented a heirarchical structure for tables.

The number of bits in an extension file (excluding the header) should be given by the formula:

```
NBITS = BITPIX * GCOUNT * (PCOUNT + NAXIS1 * NAXIS2 * ... * NAXISn)
```

The number of data records following the header record are then given by:

```
NRECORDS = INT ((NBITS + 23039) / 23040)
```

It is important that the above formulas accurately predict the number of data records in an extension "file" so that readers can skip over these "files". The data begins in the first record following the last record of the header.

Extreme caution must be exercised when inventing new types of extension files. In particular, duplication of types, or several types with the same function, must be avoided. This means that when a new extension file type is invented, it should be as general as possible so that it may be used for other similar problems.

14.5.2 Tables Extension

A very common type of extension file is one containing data that can be expressed in the form of a table. That is, a number of entries which are all identical in form. A general, self-defining table extension file type is proposed by Harten *et. al.* (1984). The following sections describe the proposed format.

The table extension file uses ASCII records to carry the tabular information. Each table entry will contain a fixed number of entries (although the number can vary between different extension files). For each entry is given (1) a label (optional), (2) the beginning column, (3) an undefined value (optional), (4) a Fortran format to decode the entry, (5) scaling and offset information (optional), and (6) the units (optional).

Tables Header Record

The keywords for tables extension file headers are given in the following:

1. XTENSION (character) is required to be the first keyword and has a value 'TABLE ' for table extension files.
2. BITPIX (integer) is a required keyword which must have a value of 8 indicating printable ASCII characters.
3. NAXIS (integer) is a required keyword which must have a value of 2 for tables extension files.
4. NAXIS1 (integer) is a required keyword which gives the number of characters in a table entry (i.e., "row").
5. NAXIS2 (integer) is a required keyword which gives the number of entries in the table (i.e., number of rows). A value of 1 is allowed.
6. PCOUNT (integer) is a required keyword which must have the value of 0 for tables extension files.
7. GCOUNT (integer) is a required keyword which must have the value of 1 for tables extension files.
8. TFIELDS (integer) is a required keyword which must follow the GCOUNT keyword. TFIELDS gives the number of fields in each table entry.
9. EXTNAME (character) is the name of the table.
10. EXTVER (integer) is the version number of the table.
11. EXTLEVEL (integer) is the hierarchal level number of the table, 1 is recommended. (optional)
12. TBCOLnnn (integer) the byte number of the first character in the nnn'th field.
13. TFORMnnn (character) the Fortran format of field nnn (I,A,E,D)

14. TTYPEnnnn (character) the label for field nnn. (optional, order arbitrary)
15. TUNITnnn (character) the physical units of field nnn. (optional, order arbitrary)
16. TSCALnnn (floating) the scale factor for field nnn. True_value = tape_value * TSCAL + TZERO. Note: TSCALnnn and TZEROnnn are not relevant to A-format fields. Default value is 1.0 (optional, order arbitrary)
17. TZEROnnn (floating) the offset for field nnn. (See TSCALnnn.) Default value is 0.0 (optional, order arbitrary)
18. TNULLnnn (character) the (tape) value of an undefined value. Note: an exact left-justified match to the field width as specified by TFORMnnn is required. (optional, order arbitrary)
19. AUTHOR (character) the name of the author or creator of the table. (optional, order arbitrary)
20. REFERENC (character) the reference for the table. (optional, order arbitrary)
21. END must always be the last keyword and the remainder of the record must be blank filled.

The TFORMnnn keywords should specify the width of the field and are of the form Iww, Aww, Eww.dd, or Dww.dd (integers, characters, single precision and double precision). If -0 is ever to be distinguished from +0 (e.g., degrees of declination), the sign field should be declared to be a separate character field.

Table Data Records

The table file data records begin with the next record following the last header record and each contains 2880 ASCII characters in the order defined by the header. Table entries do not necessarily begin at the beginning of a new record. The last record should be blank filled past the end of the valid data.

Example Table Header and Data

The first two lines of numbers are only present to show card columns and are not part of the extension file.

```
         1         2         3         4         5         6         7
1234567890123456789012345678901234567890123456789012345678901234567890
XTENSION= 'TABLE   '           / Extension type
BITPIX  =                    8 / Printable ASCII codes
NAXIS   =                    2 / Table is a matrix
NAXIS1  =                   60 / Width of table in characters
NAXIS2  =                  449 / Number of entries in table
PCOUNT  =                    0 / Random parameter count
GCOUNT  =                    1 / Group count
TFIELDS =                    3 / Number of fields in each row
EXTNAME = 'AIPS CC '           / AIPS clean components
EXTVER  =                    1 / Version number of table
TBCOL1  =                    1 / Starting char. pos. of field n
TFORM1  = 'E15.6   '           / Fortran format of field n
TTYPE1  = 'FLUX    '           / Type (heading) of field n
TUNIT1  = 'JY      '           / Physical units of field n
TSCAL1  =                  1.0 / Scale factor for field n
TZERO1  =                  0.0 / Zero point for field n
TBCOL2  =                   17 / Starting char. pos. of field n
TFORM2  = 'E15.6   '           / Fortran format of field n
TTYPE2  = 'DELTAX  '           / Type (heading) of field n
TUNIT2  = 'DEGREES '           / Physical units of field n
TSCAL2  =                  1.0 / Scale factor for field n
TZERO2  =                  0.0 / Zero point for field n
```

```
TBCOL3  =                    33 / Starting char. pos. of field n
TFORM3  = 'E15.6   '            / Fortran format of field n
TTYPE3  = 'DELTAY  '            / Type (heading) of field n
TUNIT3  = 'DEGREES '            / Physical units of field n
TSCAL3  =                   1.0 / Scale factor for field n
TZERO3  =                   0.0 / Zero point for field n
END
```

The rest of the header block is blank filled. The data cards start on the next block boundary.

```
   0.183387E+00   -0.138889E-03    0.694444E-04
   0.146710E+00   -0.138889E-03    0.694444E-04
   0.117368E+00   -0.138889E-03    0.694444E-04
   0.938941E-01   -0.138889E-03    0.694444E-04
   0.183387E+00   -0.138889E-03    0.694444E-04

                       .
                       .
                       .
```

14.5.3 3- D Tables Extension

There are several types of extension files in AIPS which do not fit, in a natural way, into the FITS Tables format (e.g., gain and calibration tables). These files tend to be large, and conversion to and from ASCII can be expensive. A "3-D" extension to the FITS tables format, in which an entry can be a 1-dimensional array, gives the needed flexibility. Use of binary data, including IEEE floating point formats, allows efficient implementation of readers and writers.

Each row in a 3-D table entry contains a fixed number of entries (although the number can vary between different extension files). The header is a standard FITS extension header; for each table entry is given (1) the size and data type of the entry, (2) a label (optional), (3) the units (optional), and (4) an undefined value (optional). An entry may be omitted from the table, but still defined in the header, by using a zero repeat count in the TFORMn entry.

3-D Tables Header Record

The keywords for 3-D tables extension file headers are given in the following:

1. XTENSION (character) is required to be the first keyword and has a value '3DTABLE' or 'A3DTABLE' for table extension files.
2. BITPIX (integer) is a required keyword which must have a value of 8.
3. NAXIS (integer) is a required keyword which must have a value of 2 for 3-D Tables extension files.
4. NAXIS1 (integer) is a required keyword which gives the number of 8-bit bytes in a table row.
5. NAXIS2 (integer) is a required keyword which gives the number of rows in the table. A value of 1 is allowed.
6. PCOUNT (integer) is a required keyword which must have the value of 0 for tables extension files.
7. GCOUNT (integer) is a required keyword which must have the value of 1 for tables extension files.
8. TFIELDS (integer) is a required keyword which must follow the GCOUNT keyword. TFIELDS gives the number of fields in each table entry.

9. EXTNAME (character) is the name of the table.
10. EXTVER (integer) is the version number of the table.
11. EXTLEVEL (integer) is the hierarchal level number of the table, 1 is recommended. (optional)
12. TFORMnnn (character) is the size and data type of field nnn. If repeat count is absent, it is assumed to be 1. A value of zero is allowed. (required, order arbitrary)
13. TTYPEnnnn (character) the label for field nnn. (optional, order arbitrary)
14. TUNITnnn (character) the physical units of field nnn. (optional, order arbitrary)
15. TNULLnnn (integer) the value of an undefined value. These are only for integer data types; for floating data types, the IEEE nan (not a number) values are used.
16. AUTHOR (character) the name of the author or creator of the table. (optional, order arbitrary)
17. REFERENC (character) the reference for the table. (optional, order arbitrary)
18. END must always be the last keyword and the remainder of the record must be blank filled.

If -0 is ever to be distinguished from +0 (e.g., degrees of declination), the sign field should be declared to be a separate character field.

Table Data Records

The 3-D table file binary data logical records begin with the next record following the last header record and each contains 2880 8-bit bytes in the order defined by the header. Table entries do not necessarily begin at the beginning of a new record. The last record should be zero filled past the end of the valid data.

The data types are defined in the following list (r is the repeat count):

1. *rL.* A logical value consists of an 8-bit ASCII "T" indicating true and "F" indicating false.
2. *rX.* A bit array will start in the most significant bit of the byte and the following bits in the order of decreasing significance in the byte. Bit signifigance is in the same order as for integers. A bit array entry consists of an integral number of bytes with trailing bits zero.
3. *rI.* A 16-bit integer is a two's complement integer with the bits in decreasing order of significance.
4. *rJ.* A 32-bit integer is a two's complement integer with the bits in decreasing order of significance.
5. *rA.* Character strings are 8-bit ASCII characters in their natural order.
6. *rE.* Single precision floating point values are in IEEE 32-bit precision format with the bits in the following order:

```
               1         2         3
     01234567890123456789012345678901
     seeeeeeeemmmmmmmmmmmmmmmmmmmmmmm

sign = -1 ** s, exponent = eee..., mantissa = 1.mmmmm... .
The value is given by:

     value = sign * 2 **(exponent-127) * mantissa
```

Note: these values have a "hidden" bit and must always be normalized. The IEEE nan (not a number) values are used to indicate an invalid number; a value with sign and all exponent bits set is recognized as a nan. The IEEE special values (-0., +/- Infinity) are not recognized. A multiplication by a factor of 4.0 converts between VAX F and IEEE 32-bit formats.

7. *rD.* Double precision floating point values are in IEEE 64-bit precision format with the bits in the following order:

```
              1         2         3
    01234567890123456789012345678901
    seeeeeeeeeeemmmmmmmmmmmmmmmmmmmm

    3       4         5         6
    23456789012345678901234567890123
    mmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm
```

```
sign = -1 ** s, exponent = eee..., mantissa = 1.mmmmm... . The value
is given by:

     value = sign * 2 **(exponent-1023) * mantissa
```

Note: these values have a "hidden" bit and must always be normalized. The IEEE nan (not a number) values are used to indicate an invalid number; a value with sign and all exponent bits set is recognized as a nan. The IEEE special values (-0., +/- Infinity) are not recognized. A multiplication by a factor of 4.0 converts between VAX G and IEEE 64-bit formats.

Example 3- D Table Header

The first two lines of numbers are only present to show card columns and are not part of the extension file.

```
         1         2         3         4         5         6
1234567890123456789012345678901234567890123456789012345678901234
XTENSION= 'A3DTABLE'           / Extension type
BITPIX  =                    8 / Binary data
NAXIS   =                    2 / Table is a matrix
NAXIS1  =                   12 / Width of table in bytes
NAXIS2  =                  449 / Number of entries in table
PCOUNT  =                    0 / Random parameter count
GCOUNT  =                    1 / Group count
TFIELDS =                    3 / Number of fields in each row
EXTNAME = 'AIPS CC '           / AIPS CLEAN components
EXTVER  =                    1 / Version number of table
TFORM1  = '1E      '           / Count and data type of field n
TTYPE1  = 'FLUX    '           / Type (heading) of field n
TUNIT1  = 'JY      '           / Physical units of field n
TFORM2  = '1E      '           / Count and data type of field n
TTYPE2  = 'DELTAX  '           / Type (heading) of field n
TUNIT2  = 'DEGREES '           / Physical units of field n
TFORM3  = '1E      '           / Count and data type of field n
TTYPE3  = 'DELTAY  '           / Type (heading) of field n
TUNIT3  = 'DEGREES '           / Physical units of field n
END
```

The rest of the header block is blank filled. The binary data starts on the next logical block boundary. The last record of table data is zero filled past the end of the valid data.

14.5.4 Older AIPS Tables

Prior to the (presumed) establishment of the standard tables extension files, AIPS had its own tables file format and a large number of tapes have been written with these tables. These old tables were encoded in ASCII and could have any number of columns in the table. However, all values in the table had to be of the same data type and written with the same format. AIPS FITS readers will continue to recognize and deal with these obsolete tables indefinitely. The following sections describe these tables.

General Form of Header

The presence of the old format AIPS tables is indicated in the main header by the presence of the integer keyword TABLES which gives the number of tables following the data records. Each table has a header record in a manner similar to the now standard extension file header, but with different keywords. The header contains the following keywords:

1. TABNAME (character) gives the name of the file.
2. TABVER (integer) gives the version number of the file.
3. TABCOUNT (integer) gives the number of entries in the table.
4. TABWIDTH (integer) gives the number of values per table entry.
5. TABCARDS (integer) gives the number of values per card image.
6. TTYPEn (character) gives a label for the n'th column.
7. NUMTYPE (character) gives the data type used for internal storage (I*2, R*4, R*8)
8. FORMAT (character) gives the format for the table elements.
9. END is the last keyword.

Data Records

The data records consist of values encoded in ASCII in 36 80-byte card images per record in a free field format. The values are encoded TABCARDS values per 80-byte card image.

CC Files

The details of the old AIPS CLEAN component (CC) table file are illustrated in the following example of a header. Component positions are given in degrees from the tangent point (reference pixel) of the image in the projected and rotated plane (i.e., not true RA and dec). Component flux densities are in Janskys. CLEAN components are stored, 2 per card image, written as 6E13.5.

```
TABNAME = 'AIPS  CC'                   / AIPS CLEAN COMPONENTS
TABVER  =                    1         / VERSION NUMBER
TABCOUNT=                  100         / # LOGICAL RECORDS IN TABLE
TABWIDTH=                    3         / # VALUES PER LOGICAL RECORD
TABCARDS=                    6         / # VALUES PER CARD IMAGE
TTYPE1  = 'DELTAX  '                   / COLUMN 1 LABEL
TTYPE2  = 'DELTAY  '                   / COLUMN 2 LABEL
TTYPE3  = 'FLUX(JY)'                   / COLUMN 3 LABEL
NUMTYPE = 'R*4     '                   / OUR INTERNAL STORAGE SIZE
FORMAT  = 'E13.5   '                   / FORMAT ACTUALLY USED HERE
END
```

AN Files

The details of the old AIPS antenna table file are illustrated in the following example of a header. Antenna positions are given in seconds (light travel time)

```
TABNAME = 'AIPS  AN'                    /ANTENNA IDS, LOCATIONS
TABVER  =                    1          /VERSION NUMBER
TABCOUNT=                   28          / # LOGICAL RECORDS IN TABLE
TABWIDTH=                    5          / # VALUES PER LOGICAL RECORD
TABCARDS=                    5          / # VALUES PER CARD IMAGE
TTYPE1  = 'AN NO.  '                    / COLUMN 1 LABEL
TTYPE2  = 'STATION '                    / COLUMN 2 LABEL
TTYPE3  = 'LX      '                    / COLUMN 3 LABEL
TTYPE4  = 'LY      '                    / COLUMN 4 LABEL
TTYPE5  = 'LZ      '                    / COLUMN 5 LABEL
END
```

14.6 AIPS FITS INCLUDEs

There are several AIPS INCLUDEs which contain tables of KEYWORD names, data types, and pointers to the AIPS catalog header. Each of the sets consists of a declaration and EQUIVALENCE include (Dnnn.inc) and a DATA include (Vnnn.inc). These includes can be used directly by routines such as FPARSE. The basic components of these includes is shown below:

1. AWORD (character*8) - this array contains the recognized keywords. This array can be sent to GETCRD as the list of keywords.
2. NCT (integer) - this gives the number of required keyword names in CWORD, which is equivalenced at the beginning of AWORD.
3. NKT (integer) - this gives the number of optional keywords names in KWORD, which is equivalenced into AWORD after CWORD.
4. ATYPE (integer) - this array gives the data types corresponding to keywords in AWORD. 1=>logical variable, 2=>numerical value, and 3=>string.
5. APOINT (integer) - this array contains pointers in the common in the include DHDR.INC to the AIPS catalog header in the form 1000*nbytes + 100*offset + position of pointer in common. Here nbytes gives the number of bytes used in the AIPS catalog header (2 means an integer which is really usually four 8-bit bytes) and the offset is the character offset past the position indicated by the header pointer. The text of these includes is in the following sections.

14.6.1 DFUV.INC

```
C                                                           Include DFUV.
C                                       FITS parsing, esp. with uv data
      INTEGER   ATYPE(151), APOINT(151), CTYPE(11), KTYPE(140),
     *   CPOINT(11), POINT(140), NKT, NCT
      CHARACTER AWORD(151)*8, CWORD(11)*8, KWORD(140)*8, K1(74)*8,
     *   K2(66)*8
      EQUIVALENCE (AWORD(1), CWORD(1)),      (AWORD(86), K2(1)),
     *   (AWORD(12), KWORD(1), K1(1))
      EQUIVALENCE (APOINT(1), CPOINT(1)),    (APOINT(12), POINT(1))
      EQUIVALENCE (ATYPE(1), CTYPE(1)),      (ATYPE(12), KTYPE(1))
C                                                           End DFUV.
```

14.6.2 DFIT.INC

```
C                                                          Include DFIT.
      INTEGER   ATYPE(83), APOINT(83), CTYPE(10), KTYPE(73), CPOINT(10),
     *   POINT(73), NKT, NCT
      CHARACTER AWORD(83)*8, CWORD(10)*8, KWORD(73)*8
      EQUIVALENCE (AWORD(1), CWORD(1)),      (AWORD(11), KWORD(1))
      EQUIVALENCE (APOINT(1), CPOINT(1)),    (APOINT(11), POINT(1))
      EQUIVALENCE (ATYPE(1), CTYPE(1)),      (ATYPE(11), KTYPE(1))
C                                                          End DFIT.
```

14.6.3 VFUV.INC

```
C                                                          Include VFUV.
C                                      FITS parsing, esp. with uv data
      DATA NCT/11/,     NKT/140/
      DATA CWORD/'SIMPLE  ','BITPIX  ','NAXIS   ','NAXIS1  ',
     *         'NAXIS2  ','NAXIS3  ','NAXIS4  ','NAXIS5  ',
     *         'NAXIS6  ','NAXIS7  ','NAXIS8  '/
      DATA K1 /'OBJECT  ','TELESCOP','INSTRUME','OBSERVER',
     *         'DATE-OBS','DATE-MAP','BSCALE  ','BZERO   ',
     *         'BUNIT   ','CTYPE1  ','CTYPE2  ','CTYPE3  ',
     *         'CTYPE4  ','CTYPE5  ','CTYPE6  ','CTYPE7  ',
     *         'CTYPE8  ','CRVAL1  ','CRVAL2  ','CRVAL3  ',
     *         'CRVAL4  ','CRVAL5  ','CRVAL6  ','CRVAL7  ',
     *         'CRVAL8  ','CDELT1  ','CDELT2  ','CDELT3  ',
     *         'CDELT4  ','CDELT5  ','CDELT6  ','CDELT7  ',
     *         'CDELT8  ','CRPIX1  ','CRPIX2  ','CRPIX3  ',
     *         'CRPIX4  ','CRPIX5  ','CRPIX6  ','CRPIX7  ',
     *         'CRPIX8  ','CROTA1  ','CROTA2  ','CROTA3  ',
     *         'CROTA4  ','CROTA5  ','CROTA6  ','CROTA7  ',
     *         'CROTA8  ','EPOCH   ','DATAMAX ','DATAMIN ',
     *         'BLANK   ','IMNAME  ','IMCLASS ','IMSEQ   ',
     *         'USERNO  ','PRODUCT ','NITER   ','BMAJ    ',
     *         'BMIN    ','BPA     ','VELREF  ','ALTRVAL ',
     *         'ALTRPIX ','OBSRA   ','OBSDEC  ','RESTFREQ',
     *         'XSHIFT  ','YSHIFT  ','DATE    ','ORIGIN  ',
     *         'ISCALE  ','IZERO   '/
      DATA K2 /'GROUPS  ','GCOUNT  ','PCOUNT  ','PTYPE1  ',
     *         'PTYPE2  ','PTYPE3  ','PTYPE4  ','PTYPE5  ',
     *         'PTYPE6  ','PTYPE7  ','PTYPE8  ','PTYPE9  ',
     *         'PTYPE10 ','PTYPE11 ','PTYPE12 ','PTYPE13 ',
     *         'PTYPE14 ','PTYPE15 ','PTYPE16 ','PTYPE17 ',
     *         'PTYPE18 ','PTYPE19 ','PTYPE20 ','PSCAL1  ',
     *         'PSCAL2  ','PSCAL3  ','PSCAL4  ','PSCAL5  ',
     *         'PSCAL6  ','PSCAL7  ','PSCAL8  ','PSCAL9  ',
     *         'PSCAL10 ','PSCAL11 ','PSCAL12 ','PSCAL13 ',
     *         'PSCAL14 ','PSCAL15 ','PSCAL16 ','PSCAL17 ',
     *         'PSCAL18 ','PSCAL19 ','PSCAL20 ','PZERO1  ',
     *         'PZERO2  ','PZERO3  ','PZERO4  ','PZERO5  ',
     *         'PZERO6  ','PZERO7  ','PZERO8  ','PZERO9  ',
     *         'PZERO10 ','PZERO11 ','PZERO12 ','PZERO13 ',
     *         'PZERO14 ','PZERO15 ','PZERO16 ','PZERO17 ',
     *         'PZERO18 ','PZERO19 ','PZERO20 ','TABLES  ',
```

```
     *        'SORTORDR','WTSCAL  '/
C                                  1=Logical variable
C                                  2=Number
C                                  3=String
      DATA CTYPE /1,2,2,2, 2,2,2,2, 2,2,2/
      DATA KTYPE /3,3,3,3, 3,3,2,2, 3,3,3,3, 3,3,3,3, 3,2,2,2,
     *            2,2,2,2, 2,2,2,2, 2,2,2,2, 2,2,2,2, 2,2,2,2,
     *            2,2,2,2, 2,2,2,2, 2,2,2,2, 2,3,3,2, 2,2,2,2,
     *            2,2,2,2, 2,2,2,2, 2,2,3,3, 2,2,    1,2,2,
     *            20*3, 20*2, 20*2, 2,3,2/
C                                  nbytes=2 => 4-byte integer
C                                  1000*nbytes + 100*offset +
C                                  position of pointer in common
C                      "KIBPX"KIDIM KINAX =>
      DATA CPOINT /   0, 2000, 2040, 2041, 2141, 2241, 2341, 2441,
     *             2541, 2641, 2741/
C                 KHOBJ KHTEL KHINS KHOBS KHDOB KHDMP"KDBSC"KDBZE"
      DATA POINT / 8001, 8002, 8003, 8004, 8005, 8006, 8000, 8000,
C                 KHBUN KHCTP =>
     *             8007, 8009, 8109, 8209, 8309, 8409, 8509, 8609,
C                       KDCRV = >
     *             8709, 8029, 8129, 8229, 8329, 8429, 8529, 8629,
C                       KRCIC =>
     *             8729, 4010, 4110, 4210, 4310, 4410, 4510, 4610,
C                       KRCRP =>
     *             4710, 4011, 4111, 4211, 4311, 4411, 4511, 4611,
C                       KRCRT =>
     *             4711, 4012, 4112, 4212, 4312, 4412, 4512, 4612,
C                       KREPO KRDMX KRDMN KRBLK KHIMN KHIMC
     *             4712, 4013, 4014, 4015, 4016,12017, 6218,
C                 KIIMS KIIMU KITYP KINIT KRBMJ KRBMN KRBPA KIALT
     *             2042, 2043, 2044, 2035, 4020, 4021, 4022, 2045,
C                 KDARV KRARP KDORA KDODE KDRST KRXSH KRYSH
     *             8033, 4023, 8030, 8031, 8032, 4024, 4025,    0,
C                                    KIGCN KIPCN
     *                0, 8000, 8000, 1001, 2034, 2039,
     *             20*8008, 20*8000, 20*8000, 2000, 2044, 4000/
C                                                       End VFUV.
```

14.6.4 VFIT.INC

```
C                                                       Include VFIT
      DATA NCT/10/,     NKT/73/
      DATA CWORD /'SIMPLE  ','BITPIX  ','NAXIS   ','NAXIS1',
     *            'NAXIS2  ','NAXIS3  ','NAXIS4  ','NAXIS5  ',
     *            'NAXIS6  ','NAXIS7  '/
      DATA KWORD /'OBJECT  ','TELESCOP','INSTRUME','OBSERVER',
     *            'DATE-OBS','DATE-MAP','BSCALE  ','BZERO   ',
     *            'BUNIT   ','CTYPE1  ','CTYPE2  ','CTYPE3  ',
     *            'CTYPE4  ','CTYPE5  ','CTYPE6  ','CTYPE7  ',
     *            'CRVAL1  ','CRVAL2  ','CRVAL3  ','CRVAL4  ',
     *            'CRVAL5  ','CRVAL6  ','CRVAL7  ','CDELT1  ',
     *            'CDELT2  ','CDELT3  ','CDELT4  ','CDELT5  ',
```

```
     *           'CDELT6  ','CDELT7  ','CRPIX1  ','CRPIX2  ',
     *           'CRPIX3  ','CRPIX4  ','CRPIX5  ','CRPIX6  ',
     *           'CRPIX7  ','CROTA1  ','CROTA2  ','CROTA3  ',
     *           'CROTA4  ','CROTA5  ','CROTA6  ','CROTA7  ',
     *           'EPOCH   ','DATAMAX ','DATAMIN ','BLANK   ',
     *           'IMNAME  ','IMCLASS ','IMSEQ   ','USERNO  ',
     *           'PRODUCT ','NITER   ','BMAJ    ','BMIN    ',
     *           'BPA     ','VELREF  ','ALTRVAL ','ALTRPIX ',
     *           'OBSRA   ','OBSDEC  ','RESTFREQ','XSHIFT  ',
     *           'YSHIFT  ','DATE    ','ORIGIN  ','TABLES  ',
     *           'OPHRAE11','OPHDCE11','MAPNAM11','ISCALE  ',
     *           'IZERO   '/
C                                  1=Logical variable
C                                  2=Number
C                                  3=String
      DATA CTYPE /1,2,2,2, 2,2,2,2, 2,2/
      DATA KTYPE /3,3,3,3, 3,3,2,2, 3,3,3,3, 3,3,3,3, 2,2,2,2,
     *           2,2,2,2, 2,2,2,2, 2,2,2,2, 2,2,2,2, 2,2,2,2,
     *           2,2,2,2, 2,2,2,2, 3,3,2,2, 2,2,2,2, 2,2,2,2,
     *           2,2,2,2, 2,3,3,2, 2,2,3,2, 2/
C                                  1000*nbytes + 100*offset +
C                                  position of pointer in common
C                  "KIBPX"KIDIM KINAX
      DATA CPOINT /   0, 2000, 2040, 2041, 2141, 2241, 2341, 2441,
     *            2541, 2641/
C               KHOBJ KHTEL KHINS KHOBS KHDOB KHDMP"KDBSC"KDBZE"
      DATA POINT / 8001, 8002, 8003, 8004, 8005, 8006, 8000, 8000,
C               KHBUN KHCTP =>
     *            8007, 8009, 8109, 8209, 8309, 8409, 8509, 8609,
C               KDCRV =>                                 KRCIC =>
     *            8029, 8129, 8229, 8329, 8429, 8529, 8629, 4010,
C                                                  KRCRP =>
     *            4110, 4210, 4310, 4410, 4510, 4610, 4011, 4111,
C                                            KRCRT = >
     *            4211, 4311, 4411, 4511, 4611, 4012, 4112, 4212,
C                                    KREPO KRDMX KRDMN KRBLK
     *            4312, 4412, 4512, 4612, 4013, 4014, 4015, 4016,
C               KHIMN KHIMC KIIMS KIIMU KITYP  KINIT KRBMJ KRBMN
     *           12017, 6218, 2042, 2043, 2044, 2035, 4020, 4021,
C               KRBPA KIALT KDARV KRARP KDORA KDODE KDRST KRXSH
     *            4022, 2045, 8033, 4023, 8030, 8031, 8032, 4024,
C               KRYSH
     *            4025,    0,    0, 2000, 4100, 4200,12017, 8000,
     *            8000/
C                                                       End VFIT.
```

14.7 AIPS FITS Parsing Routines

There are several AIPS utility routines which are useful for parsing (reading) FITS header records. These routines are briefly described in the following; details of the call sequences etc. will be given later.

1. EXTREQ parses a FITS file looking for required extension file keywords.

2. FPARSE parses a FITS header card, interpretes the card image, and puts the data value into the correct location in the AIPS catalog header. This routine is for standard uv FITS headers, but with the substitution of the INCLUDEs DFIT.INC and VFIT.INC for DFUV.INC and VFUV.INC, would work for images as well. The routine will work for FITS image tapes written on the VLA pipeline.
3. GETCRD obtains a given card image from a header block of FITS data and looks for keywords in a supplied table.
4. GETSYM finds the next symbol in a buffer. A symbol is defined to begin with a letter and have up to 8 alpha-numeric characters.
5. GETLOG obtains the value of a logical variable from a buffer.
6. GETNUM converts an ASCII numeric field into a double precision value.
7. GETSTR obtains a character string from a buffer.
8. GETKEY returns keyword values from a buffer.
9. GTWCRD reads input buffer and finds symbol value from a list of possible symbols; accepts wild cards.
10. IDWCRD searches a character string for a symbol value from a list of possible symbols using wild cards. Like GTWCRD except it works from a character string rather than a FITS buffer.
11. R3DTAB reads 3-D table data from tape and translates it to an AIPS table file.
12. RWTAB translates ASCII table data to an AIPS table file.
13. TABAXI reads a card image array containing a FITS table header and returns the values for required keywords. Like EXTREQ except it works from a character string rather than a FITS buffer.
14. TABHDK parses a character card image processing FITS table header keywords. Like TABHDR except it works from a character string rather than a FITS buffer.
15. TABHDR reads a FITS table header and returns the values for recognized symbols.

Following are the details of the call sequences and functions of the AIPS FITS parsing utility routines.

14.7.1 EXTREQ

This routine will parse a block from a FITS file and look for the required cards of a FITS extension header block, namely XTENSION, BITPIX, NAXIS, NAXISn, PCOUNT, GCOUNT.

```
EXTREQ (FDVEC, TBIND, TAPBUF, FITBLK, ICARD, EXTEN, EOF, IERR)
Inputs:
   FDVEC     I(50)   File descriptor for TAPIO input stream
Input/Output:
   TBIND     I       Pointer in FITBLK
   TAPBUF    I(*)    TAPIO i/o buffer
   FITBLK    C*2880  a block of FITS header data.
Outputs:
   ICARD     I       The number of the last card parsed.
   EXTEN     L       T means extension record, F means no.
   EOF       L       T means end of file on 1st record.
   IERR      I       0=ok, 1=messed up. An error message will
                             be printed.
```

14.7.2 FPARSE

FPARSE (parse FITS card) will unpack and interpret a card image from a block of FITS data and put that data into the internal AIPS header. Corrects for dummy 1st axis in Groups extension.

```
FPARSE (ICARD, FITBLK, PSCAL, POFF, PTYPES, TABLES, END, GROUP,
  *   BITPIX, BSC, BZE, IERR)
Inputs:
   ICARD   I         The card number (1-36) in block to interpret.
   FITBLK  C*2880    A block of FITS header data.
In/out:
   PSCAL   D(20)     Random parameter scalings
   POFF    D(20)     Random parameter offsets
   PTYPES  C*8(20)   Random parameter types
   TABLES  I         # Tables extension
   END     L         True if end card found, else false.
   GROUP   I         Set to 0 or 1 as NAXIS1 not= or = 0.
                     Checked if GROUPS keyword found later.
   BITPIX  I         Number bits/pixel on the tape
   BSC     D(2)      Scaling factor: (1) tape, (2) history
   BZE     D(2)      Scaling offset: (1) tape, (2) history
Output:
   IERR    I         error code 0=ok. 1=error.
COMMON /MAPHDR/
COMMON /FITINF/
```

14.7.3 GETCRD

GETCRD (get card) will obtain a given card image from a header block of FITS data, look for a recognizable key word from a supplied table and return information to the calling routine. Looks for keywords after 'HISTORY AIPS', 'HISTORY VLACV', 'HISTORY RANCID', or 'HISTORY VLA'

```
GETCRD (ICARD, NOSYM, STRSYM, SYMTAB, FITBLK, NPNT, KL, SYMBOL,
  *   TABNO, ISHIST, END, IERR)
Inputs:
   ICARD   I            the card image (1-36) in FITS data block.
   NOSYM   I            the number of entries in key word table.
   STRSYM  I            Start search with symbol # STRSYM
   SYMTAB  C(NOSYM)*8   keywords
   FITBLK  C*2880       the block of FITS header cards.
In/out:
   NPNT    I            The position to start scan in array KL.
                        Returns the last position scanned plus one.
   KL      C*80         On input, the card image if NPNT > 1,
                        else returns the card image.
Outputs:
   SYMBOL  C*8          the symbol found on the card.
   TABNO   I            SYMBOL matches SYMTAB(TABNO).
   ISHIST  L            True if history card else false.
   END     L            True if end card found, else false.
   IERR    I            0=match found, 1=no match on otherwise
                        valid keyword, 2=card ends or other trouble
```

14.7.4 GETKEY

Given a character string of the form: “SYMBOL = some_value”, determine the type of some_value, its value, and put all of this stuff in KEYVAL.

```
GETKEY (SYMBOL, KL, NPNT, NUMKEY, KEYWRD, KEYVAL, KEYCHR, KEYTYP, IERR)
Inputs:
   SYMBOL   C*8      Characters from KL. "SYMBOL = ..."
   KL       C*(*)    Input line of the form "SYMBOL = some_value"
In/Out:
   NPNT     I        The character position after the "="
   NUMKEY   I        The number of keywords already in KEYVAL etc.
   KEYWRD   C(*)*8   Keywords
   KEYVAL   D(*)     List of arbitrary keyword numeric values:
                        KEYVAL(n) => Value in D.
   KEYCHR   C(*)*8   List of arbitrary keyword character values:
                        KEYCHR(n) => string if KEYTYP(n)=3.
   KEYTYP   I(*)     Type code: 1=>Double, 2=>single, 3=>char.
                        4=>integer, 5=>logical
   IERR     I        Error code. 0=ok.
```

14.7.5 GETLOG

Obtains the value of a logical variable from buffer.

```
GETLOG (KB, LIMIT, KBP, IL)
Inputs:
   KB       C*80  Card image
   LIMIT    I     Number of characters in KB
   KBP      I     Pointer position at start
Outputs:
   KBP      I     Pointer position of next field
   IL       I     Value of logical field
                  0--> .false.
                  1--> .true.
                  2--> invalid
```

14.7.6 GETNUM

Converts ASCII numeric field into double precision number.

```
GETNUM (KB, KBPLIM, KBP, X)
Inputs:
   KB      C*80    character buffer
   KBPLIM  I       max # characters in buffer
   KBP     I       start of numeric field
Outputs:
   KBP     I       start of next field (incl blanks)
   X       D       numerical value: sets to magic indef = DBLANK
                   from DDCH.INC when overflow exponent or when
                   there are no numeric characters found
Common:
   DERR.INC        Sets ERRNUM to 27 on failures
```

14.7.7 GETSTR

Obtains a character string from a buffer.

```
GETSTR (KB, KBPLIM, NMAX, KBP, ISTR, NCHAR)
Inputs:
   KB      C*80     character buffer
   KBPLIM  I        size of buffer
   NMAX    I        max string length in characters
   KBP     I        start position in KB
Outputs:
   KBP     I        start position in KB next field
   ISTR    C*(*)    string, blank filled
   NCHAR   I        # characters (0 => no string found)
```

14.7.8 GETSYM

Scrutinizes a card image to look for the next symbol. A symbol begins with a letter and contains up to eight alpha-numeric characters (A-Z,0-9,_). This routine is used for interpreting a FITS file and for interpreting the HI files.

```
GETSYM (LBUFF, NPNT, SYM, IERR)
Inputs:
   LBUFF  C*80  Card image
   NPNT   I     Pointer to first character
Output:
   NPNT   I     Pointer value after getting symbol
   SYM    C*8   Symbol, padded with blanks
   IERR   I     Return code
                   0--> Found legal symbol followed by '='
                   1--> Ran off the end of the card
                   2--> Symbol had >8 characters
                   3--> Found legal symbol with no '='
                        or SYM is HISTORY or COMMENT
                   4--> Found a '/' symbol
                   5--> Symbol contains an illegar char
```

14.7.9 GTWCRD

GTWCRD (get wild card) will obtain a given card image from a header block of FITS data, search table SYMTAB for a recognizable key word in the form KEYWORDn where 'n' is an integer between 1 and NLIMIT(i) where i corresponds to the position of the keyword in SYMTAB. If NLIMIT(i)=0 KEYWORD must match the SYMTAB entry exactly.

```
GTWCRD (ICARD, NOSYM, NLIMIT, SYMTAB, FITBLK, NPNT, CARD, SYMBOL,
  *   TABNO, NFOUND, END, IERR)
Inputs:
   ICARD   I            the card image (1-36) in FITS data block.
   NOSYM   I            the number of entries in key word table.
   NLIMIT  I(NOSYM)     Upper limit on 'n'. 0 means KEYWORD must
                        match symbol table value exactly.
   SYMTAB  C(NOSYM)*8   Keywords.
   FITBLK  C*2880       the block of FITS header cards.
Outputs:
   NPNT    I            Pointer in CARD after "="
```

```
      CARD    C*80          Card image.
      SYMBOL  C*8           Symbol before "="
      TABNO   I             KEYWORD matches SYMTAB(TABNO).
      NFOUND  I             Value of 'n' for KEYWORDn.
      END     L             True if end card found, else false.
      IERR    I             0=match found, 1=no match on otherwise
                            valid keyword, 2=card ends or other trouble
```

14.7.10 IDWCRD

IDWCRD (IDentify Wild CaRD) search table SYMTAB for a recognizable key word in the form KEYWORD-n where 'n' is an integer between 1 and NLIMIT(i) where i corresponds to the position of the keyword in SYMTAB. If NLIMIT(i)=0 KEYWORD must match the SYMTAB entry exactly.

```
   IDWCRD (CARD, NOSYM, NLIMIT, SYMTAB, NPNT, SYMBOL, TABNO, NFOUND,
     *   IERR)
   Inputs:
      CARD    C*80          Input card image.
      NOSYM   I             the number of entries in key word table.
      NLIMIT  I(NOSYM)      Upper limit on 'n'. 0 means KEYWORD must
                            match symbol table value exactly.
      SYMTAB  C(NOSYM)*8    Keywords.
   Outputs:
      NPNT    I             Pointer in CARD after "="
      SYMBOL  C*8           Symbol before "=".
      TABNO   I             KEYWORD matches SYMTAB(TABNO).
      NFOUND  I             Value of 'n' for KEYWORDn.
      IERR    I             0=match found, 1=no match on otherwise
                            valid keyword, 2=card ends or other trouble
```

14.7.11 R3DTAB

This routine will read the data section of a FITS 3-D table and copy the data to the AIPS version of the table.

```
   R3DTAB (FDVEC, TBIND, DPTR, INBLK, NAXIS, IBLK, TAPBUF, IERR)
  Inputs:
     FDVEC    I(50)       File descriptor vector for TAPIO input
     DPTR     I(128,2)    Data Pointers, used in table file control.
     INBLK    I           Number of FITS blocks for table file.
     NAXIS    I(2)        Length of columns (in char), number of rows.
  In/Out:
     TBIND    I           Buffer pointer in TAPBUF
     IBLK     I(512)      Disk Table file I/O buffer.
     TAPBUF   I(*)        Tape I/O buffer.
  Outputs:
     IERR     I           Error code. 0=ok.
```

14.7.12 RWTAB

This routine will read the data section of a FITS extension file of type TABLE and decode this information using data obtained from the header section of the extension file, and write the AIPS version of a table file. Limited to tables lines <= 2880 characters.

```
RWTAB (FDVEC, TBIND, DPTR, INBLK, NAXIS, IBLK, TAPBUF, IERR)
Inputs:
   FDVEC    I(50)      File descriptor vector for TAPIO input
   DPTR     I(128,2)   Data Pointers, used in table file control.
   INBLK    I          Number of FITS blocks for table file.
   NAXIS    I(2)       Length of columns (in char), number of rows.
In/Out:
   TBIND    I          Buffer pointer in TAPBUF
   IBLK     I(>=512)   Disk Table file I/O buffer.
   TAPBUF   I(*)       Tape I/O buffer.
Outputs:
   IERR     I          Error code. 0=ok.
```

14.7.13 TABAXI

Parses FITS cards, searching for specification of required table header keywords.

```
TABAXI (CARDS, NAXIS, NAXIES, PCOUNT, GCOUNT, IERR)
Inputs:
   CARDS    C(*)*80 Input First FITS cards of a TABLE
   NAXIS    I       Number of dimensions
Outputs:
   NAXIES   I(NAXIS) Length of each dimension
   PCOUNT   I       Random parameter count
   GCOUNT   I       Groupt count
   IERR     I       0=ok, 4=NAXIES error
                    5=PCOUNT error, 6=GOUNT error,
                    An error message will be printed.
```

14.7.14 TABHDK

Interprets TABle HeaDer Keywords on an input card image and puts the data for recognized symbols into a set of output arrays. Selected 'SINGLDSH' keywords are kept; the rest are ignored

```
TABHDK (CARD, HLUN, HBLK, MAXKEY, NUMKEY, KEYWRD, KEYVAL, KEYCHR,
  *   KEYTYP, IERR)
Inputs:
   CARD     C*80     FITS Card image
   HLUN     I        History file LUN.
   HBLK     I(256)   History file I/O buffer.
   MAXKEY   I        Maximum number of keys in arrays
In/Out:
   NUMKEY   I        Location to store next keyword
Outputs:
   KEYWRD   C(*)*8   Keywords
   KEYVAL   D(*)     List of arbitrary numeric/logical values:
                        KEYVAL(n) => Value in D.
                        logicals coded as -1.0=>F, 1.0=>T.
   KEYCHR   C(*)*8   List of character keyword values values.
   KEYTYP   I(*)     Keyword type codes.
   IERR     I        error code 0=ok. 1=error.
Commons:
   /EXTHDR/    Extension file values.
   /THDR/      Header values for a tables extension file.
```

14.7.15 TABHDR

Will read a FITS table header until an END card is found. It will obtain and interpret each card image and put the data for the symbols it recognizes into a set of output arrays. Selected 'SINGLDSH' keywords are kept; the rest are ignored

```
TABHDR (FDVEC, TBIND, ICARD, HLUN, HBLK, TABTYP, NUMKEY, KEYWRD,
  *   KEYVAL, KEYCHR, KEYTYP, TAPBUF, FITBLK, IERR)
Inputs:
   FDVEC    I(50)    File descriptor vector for TAPIO input
   ICARD    I        The card number (1-36) in block to interpret
   HLUN     I        History file LUN.
   HBLK     I(256)   History file I/O buffer.
   TABTYP   I        Table type 0=>FITS ASCII, 1=> FITS 3-D
In/Out:
   TBIND    I        Buffer pointer in FITBLK
   TAPBUF   I(*)     TAPIO i/o buffer in use
   FITBLK   C*2880   A block of FITS header data.
   NUMKEY   I        on Input the max. number of keywords, on
                     output the number of arbitrary keyword/value
                     pairs found.
Outputs:
   KEYWRD   C(*)*8   Keywords
   KEYVAL   D(*)     List of arbitrary numeric/logical values:
                        KEYVAL(n) => Value in D.
                        logicals coded as -1.0=>F, 1.0=>T.
   KEYCHR   C(*)*8   List of character keyword values values.
   KEYTYP   I(*)     Keyword type codes.
   IERR     I        error code 0=ok. 1=error.
Commons:
   /EXTHDR/    Extension file values.
   /THDR/      Header values for a tables extension file.
```

14.8 References

Wells, Greisen, and Harten 1981, *Astronomy and Astrophysics* Supplement series, vol. 44, pp 363 - 370.
Greisen and Harten, 1981, *Astronomy and Astrophysics* Supplement Series, vol. 44, pp 371 - 374. Harten,

Grosbøl, Tritton, Greisen and Wells 1984, draft reproduced in the IAU Commission 9 *Astronomical Image Processing Circular.* Also appeared in *Mem. S. A. It.*, 56, 437 (1985).

THIS PAGE DELIBERATELY BLANK