FITSIO
A Subroutine Interface to FITS Format Files

Version 3.31

William D. Pence
HEASARC
Office for Guest Investigator Programs
Code 668
Goddard Space Flight Center
Greenbelt, MD 20771
USA

February 2, 1993

Contents

Chapter 1

Introduction

FITSIO is a machine-independent Fortran-77 subroutine interface for reading and writing data files in the FITS format (Flexible Image Transport System, Astronomy and Astrophysics Supplement, 44, 363). This package was written to provide a powerful yet simple interface for accessing FITS files which will run on most commonly used computers and workstations. This version of FITSIO explicitly supports the ASCII table extension and the proposed Binary table (BINTABLE) and Image extensions. FITSIO provides random access I/O capabilities which allow users to read or write information in the FITS file in any order. The FITSIO subroutines insulate the user from having to deal with the complicated formating details in the FITS file, however, it is assumed that users have a general knowledge about the structure and usage of FITS files.

The FITSIO package was originally developed for use by the HEASARC (High Energy Astrophysics Science Archive Research Center) at the NASA Goddard Space Flight Center to convert various existing and newly acquired astronomical data sets into FITS format. The latest version of the FITSIO source code, documentation, and example programs can be obtained from the HEASARC via anonymous ftp by typing the following command at the computer operating system prompt:

```
ftp tetra.gsfc.nasa.gov    (or, ftp 128.183.8.77)
```

Then type the following responses (comments appear in brackets and should not be typed on the command line):

```
ftp> user anonymous
Password:  [type your own username as the password]
ftp> cd pub/fitsio   [to move to the fitsio subdirectory]
ftp> ls              [to see a list of available files]
ftp> get read.me     [contains latest information about FITSIO]
ftp> get fitsio.doc  [complete user documentation for FITSIO]
ftp> get ...         [continue getting any other desired files]
ftp> quit            [close the ftp connection and exit]
```

Any questions, bug reports, or suggested enhancements related to the FITSIO package should be sent to the author:


```
Dr. William Pence                  Telephone:  (301) 286-4599
HEASARC                            (SPAN) LHEAVX::PENCE or 15407::PENCE
Code 668                           (Internet) pence@tetra.gsfc.nasa.gov
NASA/Goddard Space Flight Center
Greenbelt, MD 20771
USA
```

Any questions about the definition of the FITS format or other general FITS issues should be directed to the NASA Office of Standards and Technology:


```
NOST, Code 633.2, NASA Goddard Space Flight Center
Greenbelt MD 20771
USA
Internet: nost@nssdca.gsfc.nasa.gov
DECNET: NSSDCA::NOST
(301) 286-3575
```

Chapter 2

Making the FITSIO Library

2.1 Machine Specific Versions

The FITSIO package consists of about 20000 lines of Fortran source code contained in about 300 subroutines. The large majority of subroutines are written in strict Fortran-77 (except for the occasional use of INTEGER*2 variables) and are contained in the file fitsio.for. These subroutines can be compiled and run on any machine which supports Fortran-77. An additional small set of machine-specific subroutines is provided for the following computers (support for other machines may be added at a later date):

```
Machine Type                    Source Code Files
------------                    ----------------
DEC VAX/VMS                      FITSVAX.FOR VMSIEEER.MAR VMSIEEED.MAR
DEC DECstations (Ultrix)         FITSDEC.FOR
SUN workstations                 FITSSUN.FOR
   (this also works on Silicon Graphics IRIS workstations
    and NeXT computers, with minor modification)
IBM mainframes                   FITSIBM.FOR
IBM PCs (and compatibles)
  - w/Microsoft Fortran V5.0     FITSPC.FOR
  - w/Lahey fortran              FITSLAHY.FOR
Amiga                            FITSAMI.FOR
Macintosh w/Absoft MacFortran II FITSMAC.F
```

In addition, a version of FITSIO is available which runs in the IRAF environment and provides a set of SPP callable interface routines. For more information on this version, see the following section of this guide.

Both the machine-independent set of subroutines (in fitsio.for) and the appropriate set of machine-specific subroutines must be combined and compiled for use in linking application programs. In the VAX/VMS version there are also 2 assembly MACRO subroutines called vmsieeer.mar and vmsieeed.mar which must be included in the FITSIO library. A sample command file called 'makevms.com' is included in the FITSIO directory which can be used to create the FITSIO library on a VAX/VMS machine. Note that when compiling the VAX version, one must not use the /check = bounds option to check for array overflows.

On some machines (e.g., Unix and PCs) it may be advantageous to compile each FITSIO subroutine separately before combining them into a library. This can significantly reduce the size of the final executable file since only the subroutines that are actually required will be linked into the code. The file 'Makefile' is an example of a command file which can be used to build the FITSIO library on various Unix machines. Similarly, the 'makepc.for' file is a program which can be used to build a library on an IBM PC compatible machine.

On some computers with limited memory it may be necessary to divide the FITSIO.FOR file into several smaller pieces before it can be successfully compiled (e.g., on IBM PCs).

2.2 The IRAF SPP Compatible Version of FITSIO

The original version of FITSIO was written in Fortran and uses Fortran read and write statements to access the FITS files. Later it became desirable to be able to use the FITSIO interface within the IRAF environment, therefore, an IRAF compatible version of FITSIO was developed in collaboration with Doug Tody at NOAO. This version of FITSIO uses the IRAF VOS (Virtual Operating System) file I/O calls to access the FITS files rather than using Fortran I/O,

The following source files are needed to build the IRAF compatible version of FITSIO:

```
fitsio.for  - The Fortran source code for the main body of FITSIO subroutines
fitsio.h    - The SPP include file, used by fitspp.x and fitssppb.x
fitsspp.com - The SPP common block used by fitsspp.x
fitsspp.x   - The low-level FITSIO subroutines, written in SPP
fitssppb.x  - The top-level SPP interface to FITSIO
mkpkg       - a sample make file which builds the FITSIO library
              (this mkpkg file requires the mkfort executable file
               which is supplied with IRAF)
```

The SPP version of FITSIO consists of 3 layers of code. At the bottom there is the fitsspp.x SPP source code which contains the IRAF-specific subroutines to read and write data to the FITS file. The middle layer of subroutines are contained in the fitsio.for file which is the same file used in every version of FITSIO. This layer is completely portable and contains no machine or environment dependencies. Finally, a thin interface layer of SPP subroutines are provided in the fitssppb.x file. This top layer of routines is provided for the convenience of SPP programmers and simply performs the conversion between SPP character variables and Fortran character strings.

A sample mkpkg file is provided which will build the FITSIO library in IRAF. This is intended as only an example. A more complete example of how to build the FITSIO library within the IRAF environment is provided in the FTOOLS package of FITS file utility software, which is also available from the HEASARC.

Once the FITSIO library has been build, IRAF programmers may then call either the SPP or the Fortran subroutine interface to FITSIO; both interfaces end up calling the same low-level SPP VOS interface routines to actually read or write the FITS files. IRAF programs written in Fortran (using either the IRAF IMFORT or the STScI F77 interface to access the IRAF VOS) can simply call the Fortran subroutine interface to FITSIO as specified in later sections of this document. From the programmer's point of view, this is no different than using a machine-specific implementation of FITSIO (e.g., when using the fitssun.for source file on a SUN workstation).

Programs written in SPP, on the otherhand, may call the SPP specific layer of FITSIO subroutines. These routines are identical to the Fortran subroutines except that the SPP subroutines have names beginning with 'fs' rather than 'ft' (e.g., 'call fsopen' rather than 'call ftopen'). The only purpose of this thin layer of SPP routines is to convert any SPP character variables into Fortran character strings before calling the corresponding FITSIO Fortran subroutine. Similarly, any character strings that are returned by the Fortran FITSIO subroutine are converted into SPP CHAR variables before being passed back to the calling SPP routine.

SPP programmers, therefore, should follow the definition of all the FITSIO interface subroutines given in this document, except that the SPP routines have names beginning with 'fs' rather than 'ft', and SPP CHAR variables should be substituted for any Fortran character string variables in the subroutine calling arguments. SPP programmers also need to be aware of the character string length definitions that are used throughout the SPP FITSIO interface as defined in the fitsio.h include file. The SPP versions of the interface routines behave identically to the Fortran versions and have exactly the same calling sequences, except for the following few exceptions:

* FSPKNx: This family of subroutines only supports a single comment string, not an array of comment strings as in the Fortran routines.

* FSPCLS, FSGCVS, and FSGCFS: These 3 routines all have an additional argument not present in the corresponding Fortran routines. This integer argument, DIM1, follows the character array argument and specifies the size of the 1st dimension of the 2-dimensional CHAR array, e.g.,

```
call fspcls(ounit, frow, felem, nelem, nulval, array, DIM1, anynul, status)
```

* FSGIOU: This new subroutine has been added to the SPP interface and is not present in the Fortran FITSIO interface. It simply returns an unused file I/O unit number which may then be used as input to the fsopen or fsinit routines.

```
call fsgiou( iounit )
```

Chapter 3

FITSIO Usage Conventions

3.1 Current Header Data Unit (CHDU)

A basic concept used throughout this interface is that of the Current Header Data Unit (CHDU). When a FITS file is first opened, the CHDU is defined to be the primary header and data array (i.e., the first HDU in the FITS file). Subroutines are provided to move to other HDUs within the FITS file or to append a new HDU to the end of the FITS file. The other subroutines which read or write header keywords or data values always operate only within the CHDU.

3.2 Subroutine Names

All FITSIO subroutine names begin with the letters 'ft' to distinguish them from other subroutines and are 5 or 6 characters long. Users should not name their own subroutines beginning with 'ft' to avoid conflicts. (The SPP interface routines all begin with 'fs').

Subroutines which read or get information from the FITS file have names beginning with 'ftg'. Subroutines which write or put information into the FITS file have names beginning with 'ftp'.

3.3 Subroutine Families

Many of the subroutines come in families which differ only in the datatype of the associated parameter(s) . The datatype of these subroutines is indicated by the last letter of the subroutine name (e.g., 'j' in 'ftpkyj') as follows:

```
x - bit
b - character*1 (unsigned byte)
i - short integer (I*2)
j - integer (I*4)
e - real exponential floating point (R*4)
f - real fixed-format floating point (R*4)
d - double precision real floating-point (R*8)
g - double precision fixed-format floating point (R*8)
c - complex reals (pairs of R*4 values)
m - double precision complex (pairs of R*8 values)
l - logical (L*4)
s - character string
```

Notes: Some Fortran compilers support a byte datatype such as INTEGER*1, LOGICAL*1, or BYTE, which can sometimes be used instead of CHARACTER*1 variables. Some machines permit passing a numeric datatype (such as INTEGER*1) to the FITSIO subroutines which are expecting a CHARACTER*1 datatype. This is not possible on all machines, however, (e.g., on a VAX one must use the VAX-specific %DESCR function). It is also important to remember that FITS always interprets the byte value as an unsigned byte (values ranging from 0 to 255).

The double precision complex datatype is not a standard Fortran-77 datatype. If a particular Fortran compiler does not directly support this datatype, then one may instead pass an array of pairs of double precision values to these subroutines. The first value in each pair is the real part, and the second is the imaginary part.

3.4 Implicit Data Type Conversion

Many of the data I/O subroutines have the ability to perform implicit data type conversion. This means that the data type of the subroutine parameter does not need to be the same as the data type of the value in the FITS file. More specifically, the implicit data type conversion will be performed for b,i,j,e,f,d and g data types, when getting a FITS header keyword value (FTGKYx and FTGKNx subroutines) and when putting or getting values to or from the primary array or a table. No data type conversion is performed to or from the string (s), logical (l), complex (c), or double complex (m) data types. No checks for overflow conditions are performed during this forced datatype conversion.

3.5 Binary Table Bit Format Columns

The FITSIO subroutines internally handle bit array columns (with TFORM = 'nX') as Byte array columns. For instance if TFORM = '13X', then this is interpreted as if TFORM = '2B'. The FTGCX and FTPCLX subroutines may be used to read or write individual bits in these fields.

3.6 Data Scaling

When reading b,i,j,e, or d data values in the primary array (or a table column) the values will be scaled automatically by the BSCALE and BZERO (or TSCALnnn and TZEROnnn) header keyword values if they are present. The returned data will have

```
output value = (FITS value) * BSCALE + BZERO
```

(corresponding formula using TSCALnnn and TZEROnnn are used when reading from table columns). The ftpscl and fttscl subroutines may be used to override the scaling parameters defined in the header.

When writing b,i,j,e, or d data to the primary array or to a table column the data values may be scaled by defining the scaling parameters with the ftpscl and fttscl subroutines. Note that even if the BSCALE or TSCALnnn keywords exist in the FITS header, the ftpscl or fttscl subroutine still must be called to cause the data to be scaled as they are written to the FITS file. If scaling is performed, the scaled output value will have

```
FITS value = ((input value) - BZERO) / BSCALE
```

(corresponding formula using TSCALnnn and TZEROnnn are used when writing to table columns). Rounding to the nearest integer, rather than truncation is performed when writing integer datatypes to the FITS file.

3.7 Error Status Values

The last parameter in nearly every subroutine is the returned error status value which is both an input and an output parameter. A returned positive value for this parameter indicates an error was detected; negative values are reserved for non-fatal warning messages. A listing of all the possible status code values is given at the end of this document. The FITSIO library uses an 'inherited status' convention for the status parameter which means that if a subroutine is called with a positive input value of the status parameter, then the subroutine will exit immediately without changing the value of the status parameter. Thus, if one passed the status value returned from each FITSIO routine as input to the next FITSIO subroutine, then whenever an error is detected all further FITSIO processing will cease. This convention can simplify the code of application programs since it is not necessary to check the value of the status parameter after every single FITSIO subroutine call. If, for example, a program contains a sequence of several FITSIO calls, one can just check the status value after the last call. Since the returned status values are generally unique, it should be possible to determine which subroutine originally returned the error status.

3.8 Binary Table Variable Length Array Facility

The FITSIO package supports reading and writing data to variable length fields of a binary table which have TFORM = 'rPt' where r is an optional repeat count and t is the datatype code (e.g., L, X, I, E, etc.). The same subroutines which read and write data in an ordinary fixed length binary table extension (FTPCLx, FTGCVx, FTGCFx) are also used for variable length fields, however, the subroutine parameters take on a slightly different interpretation.

When writing to a variable length field, the entire array of values for a given row of the table must be written with a single call to FTPCLx. The total length of the array is calculated from (NELEM+FELEM-1). One cannot append more elements to an existing field at a later time because any attempt to append more elements will simply overwrite all the data which was previously written. Note also that the new data will be written to a new area of the 'heap' and the heap space used by the previous write cannot be reclaimed. For this reason it is advised that each row of a variable length field only be written once. An exception to this general rule occurs when setting elements of an array as undefined. One must first write a dummy value into the array with FTPCLx, and then call FTPCLU to flag the desired elements as undefined. The rows of a table may be written in any order. It is also possible to start writing the table without knowing the final total size of the heap (as specified by the PCOUNT header keyword). One can simple use a dummy value for PCOUNT initially, and then update the value after the table has been completely written (it is also necessary to update the total length of the data area with the FTDDEF subroutine). Note, however, it is not possible to increase the total number of rows in the table over what was initially declared in the FTBDEF subroutine because the heap starts immediately after the last row, unless one has used the FTPTHP subroutine to leave a gap between the end of the fixed length table and the heap (see below).

When reading from a variable length array field one can only read as many elements as exist in that row of the table; reading does not automatically continue with the next row of the table as occurs when reading an ordinary fixed length table field. Attempts to read more than this will return an error. One can determine the length of each row of the field with the FTGDES subroutine.

When reading or writing to an ASCII character field (e.g., TFORM = 'PA') only a single character string is read or written. The number of characters in the string is specified with the NELEM parameter (the FELEM parameter is ignored).

The FTPDES subroutine is useful in situations where multiple rows of a variable length column have the identical array of values. One can simply write the array once for the first row, and then use FTPDES to write the same descriptor values into the other rows; all the rows will then point to the same storage location thus saving disk space.

By default the heap starts immediately following the end of the fixed length table, (i.e., at byte location = NAXIS1 x NAXIS2). If desired, the starting location of the heap can be changed by using the FTPTHP subroutine. FTPTHP must be called after the table structure is defined (with FTBDEF) and after the required header keywords have been written, but before any data are written to the table. This subroutine also automatically writes the THEAP keyword to the binary table extension header.

Chapter 4

Usage Guidelines and Suggestions

4.1 Reading an existing FITS file

The following sequence of subroutine calls illustrate a simple example of reading an existing FITS file. (The example programs which were distributed along with the FITSIO code (wsimple.for, rsimple.for, listhead.for) also provide useful examples of the FITSIO usage).

```
 1. Open the file with FTOPEN.
 2. Read any desired header keywords with FTGHPR or FTGKYx.
 3. Read the primary data, if any, with FTGPVx or FTGPFx.
 4. Repeat steps 2 and 3 until all the desired information has been read.
 5. Move to another extension with FTMAHD or FTMRHD.
 6. Read any extension header keywords (e.g. with FTGHTB, FTGHBN or FTGKYx)
 7. Read any columns of data from the extension (e.g. with FTGCVx or FTGCFx)
 8. Repeat steps 6 and 7 until all the information has been read.
 9. Repeat steps 5 through 8 for any other extensions.
10. Close the file with FTCLOS.
```

4.2 Creating a new FITS file

The following sequence of subroutine calls illustrate a simple example of writing a new FITS file:

```
 1. Create the new file with FTINIT.
 2. Write the required primary array keywords with FTPHPR.
 3. Write any additional keywords with FTPKYx.
 4. Define the structure of the primary array with FTPDEF.
 5. Write the primary array data, if any, with FTPPRx.
 6. Create another extension, if desired, with FTCRHD.
 7. Write required header keywords for the extension with FTPHTB or FTPHBN.
 8. Write any additional keywords with FTPKYx
 9. Define the extension data structure with FTADEF or FTBDEF.
10. Write data to the extension, one column at a time with FTPCLx
11. Repeat steps 6 - 10 for any more extensions.
12. Close the FITS file with FTCLOS
```

Note that the application program should not explicitly write the required 'END' keyword at the end of each header; the FITSIO interface will automatically append the END record whenever the header is closed As a standard practice, users should always read back any FITS files that they have created to ensure that the header values and data structure are correct and self-consistent.

4.3 Advanced Usage

The FITSIO subroutines allow a great deal of flexibility in accessing FITS files which can only be briefly mentioned here. It is not necessary to read or write the information in the FITS file in the same order in which it is stored. One may, for example, read some data from one extension, then move back to read some other keywords from the primary header.

It is also not required to know the total size of a FITS data array or table before beginning to writing the header keywords; one can go back and modify the NAXISn keywords (with FTMKYx) to match the actual size of the data array after it has been completely written. One must then also use the ftddef subroutine to set the internal FITSIO parameters to the actually size of the data unit.

4.4 Limitations and Restrictions

There are a few limitation built into the FITSIO interface that users should be aware of:

* The maximum number of columns allowed in an ASCII or binary table is defined by the parameter called 'nf' used throughout the code and is currently set to 512. By definition, a FITS table may not have more than 999 columns because the TFORMnnn and TTYPEnnn keywords allow only a 3 diget sequence number. There is no limit to the number of rows allowed in a table. Similarly, there is no intrinsic limit to the size of the primary array or to the number of groups in a 'random grouped' FITS file.

* The maximum number of FITS files that may be simultaneously opened by a single program is defined by the 'nb' parameter and is currently set to 12, except in the IBM PC versions where the maximum is only 3 files to reduce memory requirements. To increase this limit, one must globally increase the value of this parameter throughout the FITSIO code. One must also add a corresponding common block buffer (e.g., common blocks ft00b1, ft00b2, ft00b3, ... ft0b12) in the following subroutines: ftgbyx, ftgcbf, ftpbyx, ftpcbf, and ftzero. The nested set of IF...ELSE IF statements in each of these subroutine must then be extended to deal with the additional file buffers.

* The maximum number of extensions allowed in a FITS file is 128 as defined by the 'ne' parameter in the code.

Chapter 5

Acknowledgements

The author would like to thank the many individuals who have helped to improve the FITSIO package by either reporting problems or suggesting useful enhancements. These include:

```
Steve Allen
Keith Arnaud
Kent Blackburn
G Bodammer
Robin Corbet
Lucio Chiappetti
Richard Fink
Phil Hodge
Mark Levine
Todd Karakaskian
Don Jennings
Claire Larkin
Rob Managan
John Mattox
Carsten Meyer
Stefan Mochnacki
Bruce O'Neel
Clive Page
Arvind Parmar
Jeff Pedelty
Tim Pearson
Chris Rogers
Arnold Rots
Barry Schlesinger
Robin Stebbins
Allyn Tennant
Peter Teuben
Doug Tody
Steve Walton
Dan Whipple
Nelson Zarate
```

Chapter 6

Subroutine Calling Sequence Definitions

This section defines the calling sequence of parameters for all the user callable Fortran FITSIO subroutines. The SPP interface subroutines have the same arguments (except as noted above) but have names that begin with 'fs' rather than 'ft'. A right arrow symbol is used in this document to separate the input parameters from the output parameters in the definition of each subroutine. This symbol is not actually part of the calling sequence. (But as described above, the status parameter is both an input and an output parameter). An alphabetical list and definition of all the parameters is given at the end of this section.

6.1 FITS File Open and Close Subroutines:

1 Open an existing FITS file with readonly or readwrite access

```
FTOPEN(unit,filename,rwmode, > blocksize,status)
```

2 Open and initialize a new empty FITS file

```
FTINIT(unit,filename,blocksize, > status)
```

3 Close a FITS file previously opened with ftopen or ftinit

```
FTCLOS(unit, > status)
```

4 Get the value of an unused I/O unit number which may then be used as input to FTOPEN or FTINIT. This routine is only available in the SPP version of FITSIO.

```
FSGIOU( > iounit )
```

6.2 Subroutines to Select or Copy the Current Header Data Unit (CHDU)

1 Move to a specified (absolute) HDU in the FITS file

```
FTMAHD(unit,nhdu, > hdutype,status)
```

2 Move to a new (existing) HDU forward or backwards relative to the CHDU

```
FTMRHD(unit,nmove, > hdutype,status)
```

3 Create (append) a new empty HDU at the end of the FITS file

```
FTCRHD(unit, > status)
```

4 Copy the entire CHDU from the FITS file associated with IUNIT to the CHDU of the FITS file associated with OUNIT. This will overwrite any data previously in the OUNIT CHDU. Space will be reserved for MOREKEYS additional keywords in the output header if there is not already enough space.

```
FTCOPY(iunit,ounit,morekeys, > status)
```

6.3 Subroutines to specify the structure of a new HDU

1 Reserve space in the CHU for MOREKEYS more header keywords. This subroutine may be called to reserve space for keywords which are to be written at a later time, after the data unit or subsequent extensions have been written to the FITS file. If this subroutine is not explicitly called, then the initial size of the FITS header will be limited to the space available at the time that the first data is written to the associated data unit. FITSIO has the ability to dynamically add more space to the header if needed, however it is more efficient to preallocate the required space if the size is known in advance.

```
FTHDEF(unit,morekeys, > status)
```

2 Define the structure of the primary array or IMAGE extension

```
FTPDEF(unit,bitpix,naxis,naxes,pcount,gcount, > status)
```

3 Define the structure of an ASCII table (TABLE) extension

```
FTADEF(unit,rowlen,tfields,tbcol,tform,nrows > status)
```

4 Define the structure of a binary table (BINTABLE) extension

```
FTBDEF(unit,tfields,tform,varidat,nrows > status)
```

5 Define the zero indexed byte offset of the 'heap' measured from the start of the binary table data. By default the heap is assumed to start immediately following the regular table data, i.e., at location NAXIS1 x NAXIS2. This routine is only relevant for binary tables which contain variable length array columns (with TFORMnnn = 'Pt'). This subroutine also automatically writes the value of theap to a keyword in the extension header. This subroutine must be called after the required keywords have been written (with ftphbn) and after the table structure has been defined (with ftbdef) but before any data is written to the table.

```
FTPTHP(unit,theap, > status)
```

6 Redefine the size of the Current Data Unit - this overrides the length of the data unit as previously defined by ftpdef, ftadef, or ftbdef. This is useful if one does not know the total size of the data unit until after the data have been written. The size (in bytes) of an ASCII or Binary table is given by NAXIS1 * NAXIS2. (Note that to determine the value of NAXIS1 it is often more convenient to read the value of the NAXIS1 keyword from the output file, rather than computing the row length directly from all the TFORM keyword values).

```
FTDDEF(unit,bytlen, > status)
```

6.4 Low-Level FITS Header I/O Subroutines

1 Put (append) an 80-character record into the CHU

```
FTPREC(unit,card, > status)
```

2 Put (append) a COMMENT keyword into the CHU

```
FTPCOM(unit,comment, > status)
```

3 Put (append) a HISTORY keyword into the CHU

```
FTPHIS(unit,history, > status)
```

4 Put (append) the DATE keyword into the CHU. The keyword value will contain the current system date as a character string in 'dd/mm/yy' format. If a DATE keyword already exists in the header, then this subroutine will simply update the keyword value in-place with the current date.

```
FTPDAT(unit, > status)
```

5 Put (append) a new keyword into the CHU.

```
FTPKY[JLS](unit,keyword,keyval,comment, > status)
FTPKY[EDFG](unit,keyword,keyval,decimals,comment, > status)
```

6 Put (append) a numbered sequence of keywords into the CHU. One may append the same comment to every keyword (and eliminate the need to have an array of identical comment strings, one for each keyword) by including the ampersand character as the last non-blank character in the (first) COMMENTS string parameter. This same string will then be used for the comment field in all the keywords. (Note that the SPP version of these routines only supports a single comment string).

```
FTPKN[JLS](unit,keyroot,startno,no_keys,keyvals,comments, > status)
FTPKN[EDFG](unit,keyroot,startno,no_keys,keyvals,decimals,comments, >
          status)
```

7 Get the nth 80-character header record from the CHU

```
FTGREC(unit,key_no, > card,status)
```

8 Get the name, value (as a string), and comment of the nth keyword in CHU

```
FTGKYN(unit,key_no, > keyword,value,comment,status)
```

9 Get the 80-character header record for the named keyword

```
FTGCRD(unit,keyword, > card,status)
```

10 Get a keyword value and comment from the CHU

```
FTGKY[EDJLS](unit,keyword, > keyval,comment,status)
FTGKEY(unit,keyword, > value,comment,status)
```

11 Get a sequence of numbered keyword values

```
FTGKN[EDJLS](unit,keyroot,startno,max_keys, > keyvals,nfound,status)
```

12 Modify (overwrite) the nth 80-character header record in the CHU

```
FTMREC(unit,key_no,card, > status)
```

13 Modify (overwrite) the 80-character header record for the named keyword in the CHU. This can be used to overwrite the name of the keyword as well as it's value and comment fields.

```
FTMCRD(unit,keyword,card, > status)
```

14 Modify (overwrite) the comment field of an existing keyword in the CHU

```
FTMCOM(unit,keyword,comment, > status)
```

15 Modify the value and comment fields of an existing keyword in the CHU. Optionally, one may modify only the value field and leave the comment field unchanged by setting the input COMMENT parameter equal to the ampersand character (&).

```
FTMKY[JLS](unit,keyword,keyval,comment, > status)
FTMKY[EDFG](unit,keyword,keyval,decimals,comment, > status)
```

16 Delete an existing keyword record. The space previously occupied by the keyword is reclaimed by moving all the following header records up one slot in the header.

```
FTDKEY(unit,keyword, > status)
```

17 Return the number of existing keywords in the CHU and the remaining space available to write additional keywords in the CHU. (returns KEYSADD = -1 if an unlimited number of keywords could be written).

```
FTGHSP(iunit, > keysexist,keysadd,status)
```

6.5 High-Level FITS Header I/O Subroutines

These subroutines provide a simple method of reading or writing most of the keyword values that are normally required in a FITS files. These subroutines are provided for convenience only and are not required to be used. If preferred, users may call the lower-level subroutines described in the previous section to individually read or write the required keywords. Note that the PCOUNT, GCOUNT and EXTEND keywords are not required in the primary header and are only written by the ftphpr subroutine if pcount is not equal to zero, gcount is not equal to zero or one, and if extend is TRUE, repectively. Similarly, the optional TUNITnnn and EXTNAME keywords are written by ftphtb or ftphbn only if the input string values are not blank.

It is important to realize that when creating a new FITS file that in addition to writing the required keywords to the header, one must also initialize the internal FITSIO buffers which define the structure of the FITS primary array or extension by calling one of the following subroutines: ftpdef, ftadef, or ftbdef. One common mistake is to forget to call ftpdef when attempting to create a null, or zero-length primary array.

NOTE: the MAXDIM parameter was added to the subroutines which read the headers keywords and the names of all these subroutines were changed beginning with version 3.21 of FITSIO. The previous version of these subroutines still remain in the FITSIO subroutine library, so that existing software can still be linked to the FITSIO library without modification.

1 Put primary header or IMAGE extension keywords into the CHU. When writing to an IMAGE extension, the SIMPLE and EXTEND parameters are ignored.

```
FTPHPR(unit,simple,bitpix,naxis,naxes,pcount,gcount,extend, > status)
```

2 Get primary header or IMAGE extension keywords from the CHU. When reading from an IMAGE extension the SIMPLE and EXTEND parameters are ignored.

```
FTGHPR(unit,maxdim > simple,bitpix,naxis,naxes,pcount,gcount,extend,
       status)
```

3 Put the ASCII table header keywords into the CHU

```
FTPHTB(unit,rowlen,nrows,tfields,ttype,tbcol,tform,tunit,extname, >
       status)
```

4 Get the ASCII table header keywords from the CHU

```
FTGHTB(unit,maxdim > rowlen,nrows,tfields,ttype,tbcol,tform,tunit,
       extname,status)
```

5 Put the binary table header keywords into the CHU

```
FTPHBN(unit,nrows,tfields,ttype,tform,tunit,extname,varidat > status)
```

6 Get the binary table header keywords from the CHU

```
FTGHBN(unit,maxdim, > nrows,tfields,ttype,tform,tunit,extname,varidat,
       status)
```

6.6 Define Data Scaling Parameters and Undefined Pixel Flags

These subroutines define or modify the internal parameters used by FITSIO to either scale the data or to represent undefined pixels. These subroutines do not create or modify the corresponding header keyword values, if any, and it is the user's responsibility to ensure that the two are consistent.

1 Reset the scaling factors in the primary array or image extension; does not change the BSCALE and BZERO keyword values and only affects the automatic scaling performed when the data elements are written/read to/from the FITS file. When writing to a new FITS file, this subroutine must be called after the call to ftpdef. (NOTE: BSCALE and BZERO must be declared as Double Precision variables).

```
FTPSCL(unit,bscale,bzero, > status)
```

2 Reset the scaling parameters for a table column; does not change the TSCALnnn or TZEROnnn keyword values and only affects the automatic scaling performed when the data elements are written/read to/from the FITS file. When writing to a new FITS extension, this subroutine must be called after the call to ftadef or ftbdef. (NOTE: TSCAL and TZERO must be declared as Double Precision variables).

```
FTTSCL(unit,colnum,tscal,tzero, > status)
```

3 Define the integer value to be used to signify undefined pixels in the primary array or image extension. This is only used if BITPIX = 8, 16, or 32. This does not create or change the value of the BLANK keyword in the header. When writing to a new FITS file, this subroutine must be called after the call to ftpdef.

```
FTPNUL(unit,blank, > status)
```

4 Define the string to be used to signify undefined pixels in a column in an ASCII table. This does not create or change the value of the TNULLn keyword. When writing to a new FITS file, this subroutine must be called after the call to ftadef.

```
FTSNUL(unit,colnum,snull > status)
```

5 Define the value to be used to signify undefined pixels in an integer column in a binary table. This does not create or change the value of the TNULLn keyword. When writing to a new FITS file, this subroutine must be called after the call to ftbdef.

```
FTTNUL(unit,colnum,tnull > status)
```

6.7 FITS Primary Array or IMAGE Extension I/O Subroutines

These subroutines put or get data values in the primary data array (i.e., the first HDU in the FITS file) or an IMAGE extension. The data array is represented as a single one-dimensional array of pixels regardless of the actual dimensionality of the array, and the FPIXEL parameter gives the position within this 1-D array of the first pixel to read or write. Automatic data type conversion is performed for numberic data (except for complex data types) if the data type of the primary array (defined by the BITPIX keyword) differs from the data type of the array in the calling subroutine. The data values are also scaled by the BSCALE and BZERO header values as they are being written or read from the FITS array. (The ftpscl subroutine can be used to define the scaling parameters when writing data to the FITS array or to override the header scaling values when reading the FITS array).

Two sets of subroutines are provided to read the data array which differ in the way undefined pixels are handled. The first set of routines (FTGPVx) simply return an array of data elements in which undefined pixels are set equal to a value specified by the user in the 'nullval' parameter. An additional feature of these subroutines is that if the user sets nullval = 0, then no checks for undefined pixels will be performed, thus increasing the speed of the program. The second set of routines (FTGPVx) returns the data element array and, in addition, a logical array which defines whether the corresponding data pixel is undefined. The latter set of subroutines may be more convenient to use in some circumstances, however it requires an additional array of logical values which can be unwieldy when working with large data arrays. Also for programmer convenience, sets of subroutines to directly read or write 2 and 3 dimensional arrays have been provided, as well as a set of subroutines to read or write any contiguous rectangular subset of pixels within the n-dimensional array.

1 Put elements into the data array

```
FTPPR[BIJED](unit,group,fpixel,nelements,values, > status)
```

2 Set data array elements as undefined

```
FTPPRU(unit,group,fpixel,nelements, > status)
```

3 Get elements from the data array. Undefined array elements will be returned with a value = nullval, unless nullval = 0 in which case no checks for undefined pixels will be performed.

```
FTGPV[BIJED](unit,group,fpixel,nelements,nullval, > values,anyf,status)
```

4 Get elements and nullflags from data array. Any undefined array elements will have the corresponding flagvals element set equal to .TRUE.

```
FTGPF[BIJED](unit,group,fpixel,nelements, > values,flagvals,anyf,status)
```

5 Put values into group parameters

```
FTPGP[BIJED](unit,group,fparm,nparm,values, > status)
```

6 Get values from group parameters

```
FTGGP[BIJED](unit,group,fparm,nparm, > values,status)
```

The following 4 subroutines transfer FITS images with 2 or 3 dimensions to or from a data array which has been declared in the calling program. The dimensionality of the FITS image is passed by the naxis1, naxis2, and naxis3 parameters and the declared dimensions of the program array are passed in the dim1 and dim2 parameters. Note that the program array does not have to have the same dimensions as the FITS array, but must be at least as big. For example if a FITS image with NAXIS1 = NAXIS2 = 400 is read into a program array which is dimensioned as 512 x 512 pixels, then the image will just fill the lower left corner of the array with pixels in the range 1 - 400 in the X an Y directions. This has the effect of taking a contiguous set of pixel value in the FITS array and writing them to a non-contiguous array in program memory (i.e., there are now some blank pixels around the edge of the image in the program array).

7 Put 2-D image into the data array

```
FTP2D[BIJED](unit,group,dim1,naxis1,naxis2,image, > status)
```

8 Put 3-D cube into the data array

```
FTP3D[BIJED](unit,group,dim1,dim2,naxis1,naxis2,naxis3,cube, > status)
```

9 Get 2-D image from the data array

```
FTG2D[BIJED](unit,group,nullval,dim1,naxis1,naxis2, > image,anyf,status)
```

10 Get 3-D cube from the data array

```
FTG3D[BIJED](unit,group,nullval,dim1,dim2,naxis1,naxis2,naxis3, >
             cube,anyf,status)
```

The followin 2 subroutines transfer a rectangular subset of pixels in a FITS N-dimensional image to or from an array which has been declared in the calling program. The fpixels and lpixels parameters are integer arrays which specify the starting and ending pixels in each dimension of the FITS image that are to be read or written. The array parameter is treated simply as a large one-dimensional array of the appropriate datatype containing the pixel values; The pixel values in the FITS array are read/written from/to this program array in strict sequence without any gaps; it is up to the calling routine to correctly interpret the dimensionality of this array.

11 Put an arbitrary data subsection into the data array.

```
FTPSS[BIJED](unit,group,naxis,naxes,fpixels,lpixels,array, > status)
```

12 Get an arbitrary data subsection from the data array.

```
FTGSS[BIJED](unit,group,nullval,naxis,naxes,fpixels,lpixels, >
             array,anyf,status)
```

6.8 FITS ASCII and Binary Table Data I/O Subroutines

6.8.1 Column Information Subroutines

1 Get the table column number corresponding to a table column name where the name is defined by the TTYPEnnn keyword.

```
FTGCNO(unit,exact,colname, > colnum,status)
```

2 Get information about an existing ASCII table column. (NOTE: TSCAL and TZERO must be declared as Double Precision variables).

```
FTGACL(unit,colnum, >
       ttype,tbcol,tunit,tform,tscal,tzero,snull,tdisp,status)
```

3 Get information about an existing binary table column. (NOTE: TSCAL and TZERO must be declared as Double Precision variables). DATATYPE is a character string which returns the datatype of the column as defined by the TFORMn keyword (e.g., 'I', 'J','E', 'D', etc.). In the case of an ASCII character column, DATATYPE will have a value of the form 'An' where 'n' is an integer expressing the width of the field in characters. For example, if TFORM = '160A8' then FTGBCL will return DATATYPE='A8' and REPEAT=20.

```
FTGBCL(unit,colnum, >
       ttype,tunit,datatype,repeat,tscal,tzero,tnull,tdisp,status)
```

6.8.2 Low-Level Table Access Subroutines

The following subroutines provide low-level access to the data in ASCII or binary tables and are mainly useful as an efficient way to copy all or part of a table from one FITS file to another. These routines simply read or write the specified number of consecutive bytes in an ASCII or binary table, without regard for column boundaries or the row length in the table. The first two subroutines read or write consecutive ASCII characters in a table to or from a character string variable, while the last two subroutines read or write consecutive bytes to or from a variable declared as a numeric data type (e.g., INTEGER, INTEGER*2, REAL, DOUBLE PRECISION). These routines do not perform any machine dependent data conversion or byte swapping, except that conversion to/from ASCII format is performed by the FTGTBS and FTPTBS routines on machines which do not use ASCII character codes in the internal data representations (e.g., on IBM mainframe computers).

1 Read a consecutive string of ASCII characters from an ASCII or binary table into a character variable (spanning columns and multiple rows if necessary)

```
FTGTBS(unit,frow,startchar,nchars, > string,status)
```

2 Write a consecutive string of characters to an ASCII or binary table from a character variable (spanning columns and multiple rows if necessary)

```
FTPTBS(unit,frow,startchar,nchars,string, > status)
```

3 Read a consecutive string of bytes from an ASCII or binary table into a numeric variable (spanning columns and multiple rows if necessary)

```
FTGTBB(unit,frow,startchar,nchars, > array,status)
```

4 Write a consecutive string of bytes to an ASCII or binary table from a numeric variable (spanning columns and multiple rows if necessary)

```
FTPTBB(unit,frow,startchar,nchars,array, > status)
```

6.8.3 High-Level Table Access Subroutines

These subroutines put or get data values in the current ASCII or Binary table extension. Automatic data type conversion is performed for numerical data types (B,I,J,E,D) if the data type of the column (defined by the TFORM keyword) differs from the data type of the calling subroutine. The data values are also scaled by the TSCALnnn and TZEROnnn header values as they are being written to or read from the FITS array. The fttscl subroutine can be used to define the scaling parameters when writing data to the table or to override the header scaling values when reading from the table.

In the case of binary tables with vector elements, the 'felem' parameter defines the starting pixel within the element vector. This parameter is ignored with ASCII tables. Similarly, in the case of binary tables the 'nelements' parameter specifies the total number of vector values read or written (continuing on subsequent rows if required) and not the number of table elements. Two sets of subroutines are provided to get the column data which differ in the way undefined pixels are handled. The first set of routines (ftgcv) simply return an array of data elements in which undefined pixels are set equal to a value specified by the user in the 'nullval' parameter. An additional feature of these subroutines is that if the user sets nullval = 0, then no checks for undefined pixels will be performed, thus increasing the speed of the program. The second set of routines (ftgcf) returns the data element array and in addition a logical array of flags which defines whether the corresponding data pixel is undefined.

1 Put elements into an ASCII or binary table column (in the CDU). (The SPP FSPCLS routine has an additional integer argument after the VALUES character string which specifies the size of the 1st dimension of this 2-D CHAR array).

```
FTPCL[SLBIJEDCM](unit,colnum,frow,felem,nelements,values, > status)
```

2 Put bit values into a binary byte ('B') or bit ('X') table column (in the CDU). LRAY is an array of logical values corresponding to the sequence of bits to be written. If LRAY is true then the corresponding bit is set to 1, otherwise the bit is set to 0.

```
FTPCLX(unit,colnum,frow,fbit,nbit,lray, > status)
```

3 Set table elements in a column as undefined

```
FTPCLU(unit,colnum,frow,felem,nelements, > status)
```

4 Get elements from an ASCII or binary table column (in the CDU). These routines return the values of the table column array elements. Undefined array elements will be returned with a value = nullval, unless nullval = 0 (or = ' ' for ftgcvs) in which case no checking for undefined values will be performed. The ANYF parameter is set to true if any of the returned elements are undefined. (Note: the ftgcl routine simple gets an array of logical data values without any checks for undefined values; use the ftgcfl routine to check for undefined logical elements). (The SPP FSGCVS routine has an additional integer argument after the VALUES character string which specifies the size of the 1st dimension of this 2-D CHAR array).

```
FTGCL(unit,colnum,frow,felem,nelements, > values,status)
FTGCV[SBIJEDCM](unit,colnum,frow,felem,nelements,nullval, >
                values,anyf,status)
```

5 Get elements and null flags from an ASCII or binary table column (in the CHDU). These routines return the values of the table column array elements. Any undefined array elements will have the corresponding flagvals element set equal to .TRUE. The ANYF parameter is set to true if any of the returned elements are undefined. (The SPP FSGCFS routine has an additional integer argument after the VALUES character string which specifies the size of the 1st dimension of this 2-D CHAR array).

```
FTGCF[SLBIJEDCM](unit,colnum,frow,felem,nelements, >
                 values,flagvals,anyf,status)
```

6 Get bit values from a binary byte ('B') or bit ('X') table column (in the CDU). LRAY is an array of logical values corresponding to the sequence of bits to be read. If LRAY is true then the corresponding bit was set to 1, otherwise the bit was set to 0.

```
FTGCX(unit,colnum,frow,fbit,nbit, > lray,status)
```

7 Get the descriptor for a variable length column in a binary table. The descriptor consists of 2 integer parameters: the number of elements in the array and the starting offset relative to the start of the heap.

```
FTGDES(unit,colnum,rownum, > nelements,offset,status)
```

8 Put the descriptor for a variable length column in a binary table. This subroutine can be used in conjunction with FTGDES to enable 2 or more arrays to point to the same storage location to save storage space if the arrays are identical.

```
FTPDES(unit,colnum,rownum,nelements,offset, > status)
```

6.9 General Utility Subroutines

The following utility subroutines may be useful for certain applications:

1 Return the revision number of the fitsio library. This subroutine returns the current revision number of the FITSIO software library. The revision number will be incremented whenever any modifications or enhancements are made to the code.

```
FTVERS( > version)
```

2 Return the descriptive text string corresponding to a FITSIO error status code. The 30-character length string contains a brief description of the cause of the error.

```
FTGERR(status, > errtext)
```

3 Convert a character string to uppercase (operates in place).

```
FTUPCH(string)
```

4 Test that the keyword name contains only legal characters: A-Z,0-9, hyphen, and underscore.

```
FTTKEY(keyword,status)
```

5 Parse a header keyword record. This subroutine parses the input header record to return the value (as a character string) and comment strings. If the keyword has no value (column 9 is not equal to '='), then the value string is returned blank and the comment string is set equal to column 11 - 80 of the input string.

```
FTPSVC(card, > value,comment,status)
```

6 Construct a sequence keyword name (ROOT + nnn). This subroutine appends the sequence number to the root string to create a keyword name (e.g., 'NAXIS' + 2 = 'NAXIS2')

```
FTKEYN(keyroot,seq_no, > keyword,status)
```

7 Determine the datatype of a keyword value string. This subroutine parses the keyword value string (usually columns 11-30 of the header record) to determine its datatype.

```
FTDTYP(value, > dtype,status)
```

8 Parse the 'TFORM' binary table column format string. This subroutine parses the input TFORM character string and returns the integer datatype code, the repeat count of the field, and, in the case of character string fields, the length of the unit string. The following datatype codes are used:

```
Datatype                DATTYP value
bit, X                  11 (interpreted as a byte array, B)
byte, B                 11
logical, L              14
ASCII character, A      16
short integer, I        21
integer, J              41
real, E                 42
double precision, D     82
complex                 83
double complex          163
```

```
FTBNFM(tform, > dattyp,repeat,width,status)
```

9 Calculate the starting column position and total ASCII table width based on the input array of ASCII table TFORM values. The SPACE input parameter defines how many blank spaces to leave between each column (it is recommended to have one space between columns for better human readability).

```
FTGABC(tfields,tform,space, > rowlen,tbcol,status
```

10 Parse a template string and return a formated 80-character string suitable for appending to (or deleting from) a FITS header file. This subroutine is useful for parsing lines from an ASCII template file and reformatting them into legal FITS header records. The formated string may then be passed to the FTPREC, FTMCRD, or FTDKEY subroutines to append or modify a FITS header record.

```
FTGTHD(template, > card,hdtype,status)
```

The input TEMPLATE character string generally should contain 3 tokens: (1) the KEYNAME, (2) the VALUE, and (3) the COMMENT string. The TEMPLATE string must adhere to the following format:

- The KEYNAME token must begin in columns 1-8 and be a maximum of 8 characters long. If the first 8 characters of the template line are blank then the remainder of the line is considered to be a FITS comment (with a blank keyword name). A legal FITS keyword name may only contain the characters A-Z, 0-9, and '-' (minus sign) and underscore. This subroutine will automatically convert any lowercase characters to uppercase in the output string. If KEYNAME = 'COMMENT' or 'HISTORY' then the remainder of the line is considered to be a FITS COMMENT or HISTORY record, respectively.

- The VALUE token must be separated from the KEYNAME token by one or more spaces and/or an '=' character. The datatype of the VALUE token (numeric, logical, or character string) is automatically determined and the output CARD string is formatted accordingly. The value token may be forced to be interpreted as a string (e.g. if it is a string of numeric digits) by enclosing it in single quotes.

- The COMMENT token is optional, but if present must be separated from the VALUE token by at least one blank space. A leading '/' character may be used to mark the beginning of the comment field, otherwise the comment field begins with the first non-blank character following the value token.

- One exception to the above rules is that if the first non-blank character in the template string is a minus sign ('-') then the following token is interpretted as the name of a keyword which is to be deleted from the FITS header.

The HDTYPE output parameter indicates how the returned CARD string should be interpreted:

```
hdtype                  interpretation
------         -------------------------------------------------
   -1          CARD(1:8) contains the name of a keyword to be deleted
               from the FITS header.

    0          append the CARD string to the FITS header if the
               keyword does not already exist, otherwise update
               the value/comment if the keyword is already present
               in the header.

    1          simply append this keyword to the FITS header (CARD
               is either a HISTORY or COMMENT keyword).

    2          This is a FITS END record; it should not be written
               to the FITS header because FITSIO automatically
               appends the END record when the header is closed.
```

EXAMPLES: The following lines illustrate valid input template strings:

```
INTVAL 7 This is an integer keyword
RVAL           34.6   /     This is a floating point keyword
EVAL=-12.45E-03  This is a floating point keyword in exponential notation
lval F This is a boolean keyword
```

```
            This is a comment keyword with a blank keyword name
SVAL1 = 'Hello world'   /  this is a string keyword
SVAL2  '123.5'  this is also a string keyword
sval3  123+  /  this is also a string keyword with the value '123+    '
- DATE / delete the DATE keyword from a FITS header.
```

Chapter 7

Summary of all FITSIO Subroutines

FITS File Open and Close Subroutines:

```
FTOPEN(unit,filename,rwmode, > blocksize,status)
FTINIT(unit,filename,blocksize, > status)
FTCLOS(unit, > status)
FSGIOU(> iounit)
```

Subroutines to Select or Copy the Current Header Data Unit (CHDU)

```
FTMAHD(unit,nhdu, > hdutype,status)
FTMRHD(unit,nmove, > hdutype,status)
FTCRHD(unit, > status)
FTCOPY(iunit,ounit,morekeys, > status)
```

Subroutines to specify the structure of a new HDU

```
FTHDEF(unit,morekeys, > status)
FTPDEF(unit,bitpix,naxis,naxes,pcount,gcount, > status)
FTADEF(unit,rowlen,tfields,tbcol,tform,nrows > status)
FTBDEF(unit,tfields,tform,varidat,nrows > status)
FTPTHP(unit,theap, > status)
FTDDEF(unit,bytlen, > status)
```

Low-Level FITS Header I/O Subroutines

```
FTPREC(unit,card, > status)
FTPCOM(unit,comment, > status)
FTPHIS(unit,history, > status)
FTPDAT(unit, > status)
FTPKY[JLS](unit,keyword,keyval,comment, > status)
FTPKY[EDFG](unit,keyword,keyval,decimals,comment, > status)
FTPKN[JLS](unit,keyroot,startno,no_keys,keyvals,comments, > status)
FTPKN[EDFG](unit,keyroot,startno,no_keys,keyvals,decimals,comments, >
           status)
FTGREC(unit,key_no, > card,status)
FTGKYN(unit,key_no, > keyword,value,comment,status)
FTGKY[EDJLS](unit,keyword, > keyval,comment,status)
FTGKEY(unit,keyword, > value,comment,status)
```

```
FTGKN[EDJLS](unit,keyroot,startno,max_keys, > keyvals,nfound,status)
FTMREC(unit,key_no,card, > status)
FTMCOM(unit,keyword,comment, > status)
FTMKY[JLS](unit,keyword,keyval,comment, > status)
FTMKY[EDFG](unit,keyword,keyval,decimals,comment, > status)
FTDKEY(unit,keyword, > status)
FTGHSP(iunit, > keysexist,keysadd,status)
```

High-Level FITS Header I/O Subroutines

```
FTPHPR(unit,simple,bitpix,naxis,naxes,pcount,gcount,extend, > status)
FTGHPR(unit,maxdim, > simple,bitpix,naxis,naxes,pcount,gcount,extend,
       status)
FTPHTB(unit,rowlen,nrows,tfields,ttype,tbcol,tform,tunit,extname, >
       status)
FTGHTB(unit,maxdim, > rowlen,nrows,tfields,ttype,tbcol,tform,tunit,
       extname,status)
FTPHBN(unit,nrows,tfields,ttype,tform,tunit,extname,varidat > status)
FTGHBN(unit,maxdim, > nrows,tfields,ttype,tform,tunit,extname,varidat,
       status)
```

Define Data Scaling Parameters and Undefined Pixel Flags.

```
FTPSCL(unit,bscale,bzero, > status)
FTTSCL(unit,colnum,tscal,tzero, > status)
FTPNUL(unit,blank, > status)
FTSNUL(unit,colnum,snull > status)
FTTNUL(unit,colnum,tnull > status)
```

FITS Primary Array or IMAGE Extension I/O Subroutines

```
FTPPR[BIJED](unit,group,fpixel,nelements,values, > status)
FTPPRU(unit,group,fpixel,nelements, > status)
FTGPV[BIJED](unit,group,fpixel,nelements,nullval, > values,anyf,status)
FTGPF[BIJED](unit,group,fpixel,nelements, > values,flagvals,anyf,status)
FTPGP[BIJED](unit,group,fparm,nparm,values, > status)
FTGGP[BIJED](unit,group,fparm,nparm, > values,status)
FTP2D[BIJED](unit,group,dim1,naxis1,naxis2,image, > status)
FTP3D[BIJED](unit,group,dim1,dim2,naxis1,naxis2,naxis3,cube, > status)
FTG2D[BIJED](unit,group,nullval,dim1,naxis1,naxis2, > image,anyf,status)
FTG3D[BIJED](unit,group,nullval,dim1,dim2,naxis1,naxis2,naxis3, >
             cube,anyf,status)
FTGSS[BIJED](unit,group,nullval,naxis,naxes,fpixels,lpixels, >
             array,anyf,status)
FTPSS[BIJED](unit,group,naxis,naxes,fpixels,lpixels,array, > status)
```

FITS ASCII and Binary Table Data I/O Subroutines

```
Column Information Subroutines:
    FTGCNO(unit,exact,colname, > colnum,status)
    FTGACL(unit,colnum, >
           ttype,tbcol,tunit,tform,tscal,tzero,snull,tdisp,status)
    FTGBCL(unit,colnum, >
           ttype,tunit,datatype,repeat,tscal,tzero,tnull,tdisp,status)

Low-Level Table Access Subroutines:
    FTGTBS(unit,frow,startchar,nchars, > string,status)
```

```
    FTPTBS(unit,frow,startchar,nchars,string, > status)
    FTGTBB(unit,frow,startchar,nchars, > array,status)
    FTPTBB(unit,frow,startchar,nchars,array, > status)

High-Level Table Access Subroutines:
    FTPCL[SLBIJEDCM](unit,colnum,frow,felem,nelements,values, > status)
    FTPCLX(unit,colnum,frow,fbit,nbit,lray, > status)
    FTPCLU(unit,colnum,frow,felem,nelements, > status)
    FTGCL(unit,colnum,frow,felem,nelements, > values,status)
    FTGCX(unit,colnum,frow,fbit,nbit, > lray,status)
    FTGCV[SBIJEDCM](unit,colnum,frow,felem,nelements,nullval, >
                   values,anyf,status)
    FTGCF[SLBIJEDCM](unit,colnum,frow,felem,nelements, >
                     values,flagvals,anyf,status)
    FTGDES(unit,colnum,rownum, > nelements,offset,status)
    FTPDES(unit,colnum,rownum,nelements,offset, > status)
```

General Utility Subroutines

```
    FTVERS( > version)
    FTGERR(status, > errtext)
    FTUPCH(string)
    FTTKEY(keyword,status)
    FTPSVC(card, > value,comment,status)
    FTKEYN(keyroot,seq_no, > keyword,status)
    FTDTYP(value, > dtype,status)
    FTBNFM(tform,dattyp,repeat,width,status)
    FTGTHD(template,card,hdtype,status)
```

Chapter 8

Parameter Definitions

```
anyf - (logical) set to TRUE if any of the returned data values are undefined
array - (any datatype except character) array of bytes to be read or written.
bitpix - (integer) bits per pixel: 8, 16, 32, -32, or -64
blank - (integer) value used for undefined pixels in integer primary array
blocksize - (integer) 2880-byte logical record blocking factor
          (if 0 < blocksize < 11) or the actual block size in bytes
          (if 10 < blocksize < 28800).  As of version 3.3 of FITSIO,
          blocksizes greater than 2880 are no longer supported.
bscale - (double precision) scaling factor for the primary array
bytlen - (integer) length of the data unit, in bytes
bzero - (double precision) zero point for primary array scaling
card - (character*80) header record to be read or written
colname - (character) ASCII name of the column
colnum - (integer) number of the column (first column = 1)
comment - (character) the keyword comment field
comments - (character array) keyword comment fields
cube - 3D data cube of the appropriate datatype
datatyp - (integer) symbolic code of the binary table column datatype
datatype - (character) datatype (format) of the binary table column
dtype - (character) datatype of the keyword ('C', 'L', 'I',  or 'F')
                C = character string
                L = logical
                I = integer
                F = floating point number
decimals - (integer) number of decimal places to be displayed
dim1 - (integer) actual size of the first dimension of the image or cube array
dim2 - (integer) actual size of the second dimension of the cube array
errtext - (character*30) descriptive error message corresponding to error number
exact - (logical) true if exact case matching of column names is to be performed
extend - (logical) true if there may be extensions following the primary data
extname - (character) value of the EXTNAME keyword (if not blank)
felem - (integer) first pixel of the element vector (ignored for ASCII tables)
fparm - (integer) sequence number of the first group parameter to read or write
fpixel - (integer) the first pixel position
fpixels - (integer array) the first included pixel in each dimension
filename - (character) name of the FITS file
flagvals - (logical array) True if corresponding data element is undefined
fbit - (integer) first bit in the field to be read or written
```

```
frow - (integer) beginning row number (first row of table = 1)
gcount - (integer) value of the GCOUNT keyword (usually = 1)
group - (integer) sequence number of the data group (=0 for non-grouped data)
hdtype - (integer) header record type: -1=delete;  0=append or replace;
                   1=append; 2=this is the END keyword
hdutype - (integer) type of HDU: 0=primary array, 1=ASCII table, 2=binary table
history - (character) the HISTORY keyword comment string
image - 2D image of the appropriate datatype
iounit - (integer) value of an unused I/O unit number
iunit - (integer) logical unit number associated with the input FITS file, 1-199
key_no - (integer) sequence number (starting with 1) of the keyword record
keyroot - (character) root string for the keyword name
keysexist - (integer) number of existing keyword records in the CHU
keysadd -(integer) number of new keyword records which can fit in the CHU
keyval - value of the keyword in the appropriate datatype
keyvals - (array) value of the keywords in the appropriate datatype
keyword - (character*8) name of a keyword
lray - (logical array) array of logical values corresponding to the bit array
lpixels - (integer array) the last included pixel in each dimension
maxdim - (integer) dimensioned size of the NAXES, TTYPE, TFORM or TUNIT arrays
max_keys - (integer) maximum number of keywords to search for
morekeys - (integer) will leave space in the header for this many more keywords
naxes - (integer array) size of each dimension in the FITS array
naxis - (integer) number of dimensions in the primary array or IMAGE extension
naxis1 - (integer) length of the X/first axis of the FITS array
naxis2 - (integer) length of the Y/second axis of the FITS array
naxis3 - (integer) length of the Z/third axis of the FITS array
nbit - (integer) number of bits in the field to read or write
nchars - (integer) number of characters to read and return
nelements - (integer) number of data elements to read or write
nhdu - (integer) absolute number of the HDU to move to (1st HDU = 1)
nmove - (integer) number of HDUs to move (+ or -), relative to current position
nfound - (integer) number of keywords found (highest keyword number)
no_keys - (integer) number of keywords to write in the sequence
nparm - (integer) number of group parameters to read or write
nrows - (integer) number of rows in the table
nullval - value to represent undefined pixels, of the appropriate datatype
offset - (integer) byte offset in the heap to the first element of the array
ounit - (integer) logical unit number associated with the output FITS file 1-199
pcount - (integer) value of the PCOUNT keyword (usually = 0)
repeat - (integer) length of element vector (e.g. 12J); ignored for ASCII table
rowlen - (integer) length of a table row, in characters or bytes
rownum - (integer) number of the row (first row = 1)
rwmode - (integer) file access mode: 0 = readonly, 1 = readwrite (ignored
         on some machines, e.g., SUNs)
seq_no - (integer) the sequence number to append to the keyword root name
simple - (logical) does the FITS file conform to all the FITS standards
snull - (character) value used to represent undefined values in ASCII table
space - (integer) number of blank spaces to leave between ASCII table columns
startchar - (integer) first character in the row to be read
startno - (integer) value of the first keyword sequence number (usually 1)
status - (integer) returned error status code (0 = OK)
string - (character) returned character string
tbcol - (integer array) column number of the first character in each field
```

```
tdisp - (character) Fortran type display format for the table column
template-(character) template string for a FITS header record
tfields - (integer) number of fields (columns) in the table
tform - (character array) format of the column; allowed values are:
         For ASCII tables:  Iw, Aw, Fww.dd, Eww.dd, or Dww.dd
         For binary tables: rL, rX, rB, rI, rJ, rA, rAw, rE, rD, rC, rM
         where 'w'=width of the field, 'd'=no. of decimals, 'r'=repeat count
         Note that the 'rAw' form is non-standard and is  not defined in the
         current draft of the Binary Tables definition document.
theap - (integer) zero indexed byte offset of starting address of the heap
         relative to the beginning of the binary table data
tnull - (integer) value used to represent undefined values in binary table
ttype - (character array) label for table column(s)
tscal - (double precision) scaling factor for table column
tunit - (character array) physical unit for table column(s)
tzero - (double precision) scaling zero point for table column
value - (character) the keyword value string
values - array of data values of the appropriate datatype
varidat - (integer) size in bytes of the 'variable length data area'
           following the binary table data (usually = 0)
unit - (integer) logical unit number associated with the FITS file (1-199)
version - (real) current revision number of the library
width - (integer) width of the character string field
```

Chapter 9

FITSIO Error Status Codes

```
  0  OK, no error
101  illegal logical unit number; must be between 1 - 199, inclusive
102  too many FITS files open at once; all internal buffers full
103  unable to find requested file to open; does it exist?
104  error opening existing file
105  error creating new FITS file; (does a file with this name already exist?)
106  error writing record to FITS file
107  end-of-file encountered while reading record from FITS file
108  error reading record from file
109  FITS record blocking factor out of range; must be between 1-28800
110  error closing FITS file
111  too many fields in table; internal array dimensions too small
112  Cannot modify file with readonly access

201  failed to write keyword record; not enough room in header
202  specified keyword name was not found in the header
203  specified header record number is out of bounds
204  keyword value field is blank
205  keyword value string is missing the closing quote character
206  error while appending sequence number to keyword root name
207  illegal character in keyword name or header record
208  keyword does not have expected name. Keyword out of sequence?
209  keyword does not have expected integer value
210  header keyword is missing "= " in columns 9-10 of the string
211  illegal BITPIX keyword value
212  illegal NAXIS keyword value
213  illegal NAXISnnn keyword value: must be 0 or positive integer
214  illegal PCOUNT keyword value
215  illegal GCOUNT keyword value
216  illegal TFIELDS keyword value
217  illegal ASCII or binary table width value (NAXIS1)
218  illegal number of rows in ASCII or binary table (NAXIS2)
219  column name (TTYPE keyword) not found
220  illegal SIMPLE keyword value
221  could not find the required SIMPLE header keyword
222  could not find the required BITPIX header keyword
223  could not find the required NAXIS header keyword
224  could not find all the required NAXISnnn keywords in the header
```

```
225  could not find the required XTENSION header keyword
226  the CHDU is not an ASCII table extension
227  the CHDU is not a binary table extension
228  could not find the required PCOUNT header keyword
229  could not find the required GCOUNT header keyword
230  could not find the required TFIELDS header keyword
231  could not find all the required TBCOLnnn keywords in the header
232  could not find all the required TFORMnnn keywords in the header
233  the CHDU is not an IMAGE extension
241  binary table row width is not equal to the sum of the field widths
251  unrecognizable type of FITS extension
252  unrecognizable FITS record
261  unable to parse the TFORM keyword value string
262  unrecognizable TFORM datatype code

302  only the first HDU in the file many be defined as the Primary Array
303  must define data unit structure before attempting this operation
304  attempt to move to negative file record number
305  cannot move directly to specified HDU: unknown starting location
306  attempted to read or write a negative number of bytes in the FITS file
307  illegal starting row number for table read or write operation
308  illegal starting element number for table read or write operation
309  attempted to read or write character string in non-character table column
310  attempted to read or write logical value in non-logical table column
311  illegal ASCII table TFORM format code for attempted operation
312  illegal binary table TFORM format code for attempted operation
313  error while converting value to formated character string
314  value for undefined pixels has not been defined
315  error during internal read of character string into integer or real value
316  illegal value in logical value column of the binary table
317  attempted to read or write descriptor in a non-descriptor field
318  attempted to read data from a zero length variable length array
319  attempted to read or write past end of variable length array

401  error attempting to convert an integer to a formated character string
402  error attempting to convert a real value to a formated character string
403  cannot convert a quoted string keyword to an integer
404  attempted to read a non-logical keyword value as a logical value
405  cannot convert a quoted string keyword to a real value
406  cannot convert a quoted string keyword to a double precision value
407  error attempting to read character string as an integer
408  error attempting to read character string as a real value
409  error attempting to read character string as a double precision value
411  illegal number of decimal places while formating floating point value
```