The FITSIO Cookbook

How to Read and Write FITS Format Files

Version 1.1

William D. Pence
HEASARC
Code 668
Goddard Space Flight Center
Greenbelt, MD 20771
USA

April 1995

Contents

Chapter 1

Introduction to FITSIO

This is the FITSIO Cookbook that contains an annotated listing of various computer programs that read and write files in FITS format using the FITSIO subroutine interface. These examples are actual working programs which users may adapt and modify for their own purposes. This Cookbook serves as a companion to the FITSIO User's Guide (described below) that provides more complete documentation on all the available FITSIO subroutines.

1.1 What is FITSIO?

FITSIO is a large library of subroutines that make it easier for programmers to read and write data files in FITS format. FITS stands for the Flexible Image Transport System and is the standard data format used within (and to some extent, outside) the astronomical community for archiving and transporting data files. When the FITS format was first developed in the late 1970s it was only used for simple n-dimensional arrays of data (such as 2-D images), so it was relatively easy for programmers to write software to directly read and write data in FITS format. The FITS format has grown in complexity, however, with the addition of various 'extensions' to store tabular data, so that it has become increasingly difficult and time consuming for programmers to support software to read or write all the various possible FITS formats. For this reason, the FITSIO subroutine library was developed in the early 1990s to provide a simple, machine-independent interface which programmers can use to read and write FITS data files. Most of the internal details of the FITS file are hidden by this interface, so programmers can concentrate on simply accessing the information they need, without having to worry about all the complicated FITS formatting rules.

The basic FITSIO library is written in ANSI standard Fortran-77 and is supported on all commonly used computers and operating systems. There are also available sets of C macro and SPP wrapper routines for the convenience of programmers of those languages. (SPP is the programming language used in the IRAF data analysis system).

1.2 Sources of information on FITSIO and FITS

All the FITSIO documentation and source code are freely available electronically via anonymous ftp from the legacy.gsfc.nasa.gov site in the /software/fitsio directory. FITSIO programmers should be sure to obtain a copy of the FITSIO User's Guide (in the files fitsio.doc or fitsio.tex or fitsio.ps) which gives a detailed description of the FITSIO user interface.

The FITSIO documentation assumes that the user has some familiarity with the definition and structure of FITS files. The NASA/GSFC Science Office of Standards and Technology (NOST) can provide a wide range of information and help regarding FITS formats. In particular, the NOST maintains 2 very useful documents: 'A User's Guide for FITS' and the 'NOST FITS Standard'. The former gives an introduction to the history and structure of FITS files, while the latter provides the authoritative definition of FITS formats. The NOST can be contacted via the internet at nost@nssdca.gsfc.nasa.gov, or by telephone at (301) 286-3575. They maintain an anonymous ftp site at nssdca.gsfc.nasa.gov which contains the above documents as well as other FITS software and information.

1.3 A FITS Primer

This section gives a brief overview of the structure of FITS files. Users should refer to the extensive documentation available from the NOST, as described above, for more detailed information on FITS formats.

A FITS file consists of one or more Header + Data Units (HDUs), where the first required HDU is called the Primary HDU, or Primary Array. Other optional HDUs may follow the primary, and are called Extensions. There are currently 4 common types of HDUs:

- Primary Array - the first HDU; consists of an N-dimensional array of pixels
- Image Extension - same structure as the primary array, but occurs after the first HDU
- ASCII Table Extension - rows and columns of data in ASCII character format
- Binary Table Extension - rows and columns of data in binary representation.

In every case the HDU consists of an ASCII Header Unit followed by an optional Data Unit. For historical reasons, each Header or Data unit must be an exact multiple of 2880 8-bit bytes long. Any unused space is padded with fill characters (ASCII blanks or NULs depending on the type of unit).

Each Header Unit consists of any number of 80-character keyword 'card images' which have the general form:

```
KEYNAME =  value  / comment string
```

The keyword names are a maximum of 8 characters long, and only uppercase letters, the digits 0-9, the hyphen, and the underscore character are allowed. The keyword name is (usually) followed by an equal sign and a space character in columns 9 - 10 of the record, followed by the value of the keyword which may be either an integer, a floating point number, a character string (enclosed in single quotes), or a boolean value (T or F). There are many rules governing the exact format of a keyword record (see the NOST FITS Standard) so it is better to rely on standard interface software like FITSIO to correctly construct or to parse the keyword records rather than try to deal with them from scratch. The last keyword in the header is always the 'END' keyword, with no value or comment fields.

Each Header Unit begins with a number of required keywords, depending on the type of Extension. These required keywords specify the size and format of the following Data Unit. Other optional keywords may be provided to describe other aspects of the data, such as the units or scaling values. Other COMMENT or HISTORY keywords may also be added to further document the data file.

The optional Data Unit immediately follows the last 2880-byte block in the Header Unit. Some HDUs do not have a Data Unit and only consist of the Header Unit.

If there is more than one HDU in the FITS file, then the Header Unit of the next HDU immediately follows the last 2880-byte block of the previous Data Unit (or Header Unit if there is no Data Unit).

1.4 Basic Conventions used in FITSIO

1.4.1 Current Header Data Unit (CHDU)

A basic concept used throughout the FITSIO 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). Most of the FITSIO subroutines only operate within the CHDU and will only read or write header keywords or data values in the CHDU. Other FITSIO routines are provided to move to other HDUs within the FITS file or to append a new HDU to the end of the FITS file.

1.4.2 Subroutine Names

All the fortran FITSIO subroutine names begin with the letters 'ft....' and are 5 or 6 characters long. Programmers should avoid naming their own subroutines beginning with 'ft' to prevent conflicts. In general, routines that write or put information into the FITS file have names beginning with 'ftp...' and routines that read or get information from FITS files have names beginning with 'ftg...'. The C macro and SPP wrapper routines have the identical calling arguments as do the fortran routines, and the names are the same except that they begin with 'FC' or 'fs' respectively, instead of 'ft'.

1.4.3 Subroutine Families

Many of the FITSIO 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
```

1.4.4 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. If the data types differ, then FITSIO will automatically perform the data type conversion when passing the data to or from the FITS file. See the FITSIO User's guide for a more complete description of this feature.

1.4.5 Data Scaling

One of the features of FITS is that 2 keywords called BSCALE and BZERO (or TSCALn and TZEROn in the case of columns in a table) may be used to define a linear scaling factor and offset that should be applied to the raw data values in the FITS file. The FITSIO routines automatically check for the presence of these keywords and will by default apply this scaling to the data values as they are read before the values are passed to the application program. Similarly, FITSIO will transparently apply the inverse scaling when writing data to a FITS file.

1.4.6 Error Status Values and the Error Message Stack

The last parameter in nearly every subroutine is the integer error status value that 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 the 'FITSIO User's Guide'. 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 passes the status value returned from each FITSIO routine as input to the next FITSIO subroutine, then whenever an error occurs all further FITSIO processing will cease. This convention can simplify the error checking in application programs because 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 distinctive, it should be possible to determine which subroutine originally returned the error status. The example programs listed in this cookbook generally only check the status value at the end of the program for simplicity, but it would be better programming practice to check the status value more frequently to trap any errors closer to their source.

FITSIO also generates an internal stack of error messages whenever an error occurs. These error messages in general provide a much more detailed explanation of the cause of the error than is provided by the error status number alone. It it recommended that the error message stack be printed out whenever a program detects a FITSIO error. The PRINTERROR subroutine given in this Cookbook provides an example of how to make use of the error message stack.

Chapter 2

Example FITSIO Programs

The following sections give listings and descriptions of various working programs that manipulate FITS files. The source code for all these routines is available from the FITSIO anonymous ftp site at legacy.gsfc.nasa.gov in the file /software/fitsio/cookbook.f. Programmers may freely use or modify any of these programs as they wish for their own purposes on the understanding that they are provided without warranty. Any bugs, problems, or suggestions regarding these programs should be sent to the author at internet address pence@tetra.gsfc.nasa.gov.

For each program the source listing is given on the right-hand page, with annotated comments on the facing page. The number of each comment corresponds to the statement number in the program listing.

2.1 WRITEIMAGE - write a FITS 2-D image

This routine creates a new FITS 2-dimensional image.

1. The STATUS parameter must be initialized before using FITSIO. A positive value of STATUS is returned whenever a serious error occurs. FITSIO uses an 'inherited status' convention, which means that if a subroutine is called with a positive input value of STATUS, then the subroutine will exit immediately, preserving the status value. Thus if one passes the status value returned from each FITSIO subroutine as input to the next FITSIO subroutine, then whenever an error occurs all further FITSIO processing will cease. This convention simplifies the error checking in an application program because it is not necessary to check the returned status value after every FITSIO call. The status values are distinctive, so it is usually possible to determine which FITSIO subroutine originally returned the non-zero error status. See the PRINTERROR subroutine for further discussion of the error status values and the related error message stack in FITSIO. For simplicity, this program only checks the status value at the end of the program, but it is usually better practice to check the status value more frequently.

2. The DELETEFILE routine (described later in the Utilities section of the Cookbook) simply deletes the FITS file if it already exists so that we can then regenerate it.

3. FTGIOU returns an unused Fortran logical unit number for use when opening a file. This routine must only be used if ALL unit numbers used by the program are allocated though it, because it will have no knowledge of any other unit numbers that the program may have otherwise used. Use of this routine is not required, and programmers may use some other technique to allocate unit numbers in their programs if preferred.

4. FTINIT creates an new empty FITS file with the specified name and blocksize. Normally the default blocksize of 1 (= 2880 bytes) should be used.

5. FTPHPR writes all the required header keywords for the primary array. BITPIX = 16 means that the image pixels will consist of 16-bit integers. The size of the image is given by the NAXES values. The EXTEND = TRUE parameter indicates that the FITS file may contain extensions following the primary array.

6. FTPPRJ writes the image pixel values to the FITS primary array. The last letter of the subroutine name defines the datatype of the array argument; in this case the 'J' indicates that the array has an integer*4 datatype.

7. FTPKYJ illustrates how to write an optional keyword to the header. In this case, the EXPOSURE keyword has a value of 1500 with a comment string = 'Total Exposure Time'.

8. The FITS file must always be closed before exiting the program. Any unit numbers allocated with FTGIOU must be freed with FTFIOU.

9. PRINTERROR is a general routine to print out error messages and is described later in the Utilities section of this cookbook.

```
      subroutine writeimage

C     Create a FITS primary array containing a 2-D image

      integer status,unit,blocksize,bitpix,naxis,naxes(2)
      integer i,j,group,fpixel,nelements,array(300,200)
      character filename*80
      logical simple,extend

 1    status=0
C     Name of the FITS file to be created:
      filename='ATESTFILEZ.FITS'

C     Delete the file if it already exists, so we can then recreate it
 2    call deletefile(filename,status)

C     Get an unused Logical Unit Number to use to open the FITS file
 3    call ftgiou(unit,status)

C     create the new empty FITS file
      blocksize=1
 4    call ftinit(unit,filename,blocksize,status)

C     initialize parameters about the FITS image (300 x 200 16-bit integers)
      simple=.true.
      bitpix=16
      naxis=2
      naxes(1)=300
      naxes(2)=200
      extend=.true.

C     write the required header keywords
 5    call ftphpr(unit,simple,bitpix,naxis,naxes,0,1,extend,status)

C     initialize the values in the image with a linear ramp function
      do j=1,naxes(2)
          do i=1,naxes(1)
              array(i,j)=i+j
          end do
      end do

C     write the array to the FITS file
      group=1
      fpixel=1
      nelements=naxes(1)*naxes(2)
 6    call ftpprj(unit,group,fpixel,nelements,array,status)

C     write another optional keyword to the header
 7    call ftpkyj(unit,'EXPOSURE',1500,'Total Exposure Time',status)

C     close the file and free the unit number
 8    call ftclos(unit, status)
      call ftfiou(unit, status)

C     check for any error, and if so print out error messages
 9    if (status .gt. 0)call printerror(status)
      end
```

2.2 WRITEASCII - write a FITS ASCII table

This routine writes a simple FITS ASCII table containing 3 columns and 6 rows. For convenience, the ASCII table extension is appended to the FITS image file created previously by the WRITEIMAGE routine.

1. The STATUS parameter must always be initialized. See the WRITEIMAGE program for a more detailed discussion.
2. FTGIOU allocates a logical unit number for use when opening a file. See the WRITEIMAGE program for a more detailed discussion.
3. The FITS file is opened with WRITE access by specifying READWRITE = 1.
4. FTCRHD creates a new empty FITS extension following the current extension and moves to it. In this case, FITSIO was initially positioned on the primary array when the FITS file was first opened, so FTCRHD appends an empty extension and moves to it. All future FITSIO calls then operate on the new extension (which will be an ASCII table).
5. FTGABC is a convenient subroutine for calculating the total width of the table and the starting position of each column in an ASCII table. Any number of blank spaces (including zero) may be inserted between each column of the table, as specified by the NSPACE parameter.
6. FTPHTB writes all the required header keywords which define the structure of the ASCII table. NROWS and TFIELDS give the number of rows and columns in the table, and the TTYPE, TBCOL, TFORM, and TUNIT arrays give the column name, starting position, format, and units, respectively of each column. The values of the ROWLEN and TBCOL parameters were previously calculated by the FTGABC routine.
7. FTPCLS writes the string values to the NAME column (column 1) of the table. The FTPCLJ and FTPCLE routines write the diameter (integer) and density (real) value to the 2nd and 3rd columns. The FITSIO routines are column oriented, so it is usually easier to read or write data in a table in a column by column order rather than row by row.
8. The FITS file must always be closed before exiting the program. Any unit numbers allocated with FTGIOU must be freed with FTFIOU.
9. PRINTERROR is a general routine to print out error messages and is described later in the Utilities section of this cookbook.

```
      subroutine writeascii

C     Create an ASCII table containing 3 columns and 6 rows

      integer status,unit,readwrite,blocksize,tfields,nrows,rowlen
      integer nspace,tbcol(3),diameter(6), colnum,frow,felem
      real density(6)
      character filename*40,extname*16
      character*16 ttype(3),tform(3),tunit(3),name(6)
      data ttype/'Name','Diameter','Density'/
      data tform/'A8','I6','F4.2'/
      data tunit/' ','km','g/cm'/
      data name/'Mercury','Venus','Earth','Mars','Jupiter','Saturn'/
      data diameter/4880,12112,12742,6800,143000,121000/
      data density/5.1,5.3,5.52,3.94,1.33,0.69/

 1    status=0
C     Name of the FITS file to append the ASCII table to:
      filename='ATESTFILEZ.FITS'

C     Get an unused Logical Unit Number to use to open the FITS file
 2    call ftgiou(unit,status)

C     open the FITS file, with write access
 3    readwrite=1
      call ftopen(unit,filename,readwrite,blocksize,status)

C     append a new empty extension onto the end of the primary array
 4    call ftcrhd(unit,status)

C     define parameters for the ASCII table (see the above data statements)
      tfields=3
      nrows=6
      extname='PLANETS_ASCII'

C     calculate the starting position of each column, and the total row length
      nspace=1
 5    call ftgabc(tfields,tform,nspace,rowlen,tbcol,status)

C     write the required header parameters for the ASCII table
 6    call ftphtb(unit,rowlen,nrows,tfields,ttype,tbcol,tform,tunit,
     &            extname,status)

C     write names to the first column, diameters to 2nd col., and density to 3rd
      frow=1
      felem=1
      colnum=1
 7    call ftpcls(unit,colnum,frow,felem,nrows,name,status)
      colnum=2
      call ftpclj(unit,colnum,frow,felem,nrows,diameter,status)
      colnum=3
      call ftpcle(unit,colnum,frow,felem,nrows,density,status)

C     close the FITS file and free the unit number
 8    call ftclos(unit, status)
      call ftfiou(unit, status)

C     check for any error, and if so print out error messages
 9    if (status .gt. 0)call printerror(status)
      end
```

2.3 WRITEBINTABLE - write a FITS binary table

This routine creates a FITS binary table, or BINTABLE, containing 3 columns and 6 rows. This routine is nearly identical to the previous WRITEASCII routine, except that the call to FTGABC is not needed, and FTPHBN is called rather than FTPHTB to write the required header keywords.

1. The STATUS parameter must always be initialized. See the WRITEIMAGE program for a more detailed discussion.

2. FTGIOU allocates a logical unit number for use when opening a file. See the WRITEIMAGE program for a more detailed discussion.

3. The FITS file is opened with WRITE access by specifying READWRITE = 1.

4. FTMAHD moves the internal FITSIO pointer to the second HDU (as specified here by the second parameter) which in this case is the ASCII table extension created by the previous WRITEASCII routine.

5. FTCRHD creates a new empty FITS extension following the current extension and moves to it. All future FITSIO calls now operate on this new extension which will become a binary table.

6. FTPHBN writes all the required header keywords which define the structure of the binary table. NROWS and TFIELDS gives the number of rows and columns in the table, and the TTYPE, TFORM, and TUNIT arrays give the column name, format, and units, respectively of each column.

7. FTPCLS writes the string values to the NAME column (column 1) of the table. The FTPCLJ and FTPCLE routines write the diameter (integer) and density (real) value to the 2nd and 3rd columns. The FITSIO routines are column oriented, so it is usually easier to read or write data in a table in a column by column order rather than row by row. Note that the identical subroutine calls are used to write to either ASCII or binary FITS tables.

8. The FITS file must always be closed before exiting the program. Any unit numbers allocated with FTGIOU must be freed with FTFIOU.

9. PRINTERROR is a general routine to print out error messages and is described later in the Utilities section of this cookbook.

```
      subroutine writebintable

C     Create a binary table containing 3 columns and 6 rows

      integer status,unit,readwrite,blocksize,hdutype,tfields,nrows
      integer varidat,diameter(6), colnum,frow,felem
      real density(6)
      character filename*40,extname*16
      character*16 ttype(3),tform(3),tunit(3),name(6)
      data ttype/'Name','Diameter','Density'/
      data tform/'8A','1J','1E'/
      data tunit/' ','km','g/cm'/
      data name/'Mars','Jupiter','Saturn','Uranus','Neptune','Pluto'/
      data diameter/6800,143000,121000,47000,45000,6000/
      data density/3.94,1.33,0.69,1.56,2.27,1.0/

 1    status=0
C     Name of the FITS file to append the ASCII table to:
      filename='ATESTFILEZ.FITS'

C     Get an unused Logical Unit Number to use to open the FITS file
 2    call ftgiou(unit,status)

C     open the FITS file, with write access
 3    readwrite=1
      call ftopen(unit,filename,readwrite,blocksize,status)

C     move to the last (2nd) HDU in the file
 4    call ftmahd(unit,2,hdutype,status)

C     append/create a new empty HDU onto the end of the file and move to it
 5    call ftcrhd(unit,status)

C     define parameters for the binary table (see the above data statements)
      tfields=3
      nrows=6
      extname='PLANETS_BINARY'
      varidat=0

C     write the required header parameters for the binary table
 6    call ftphbn(unit,nrows,tfields,ttype,tform,tunit,
     &            extname,varidat,status)

C     write names to the first column, diameters to 2nd col., and density to 3rd
      frow=1
      felem=1
      colnum=1
 7    call ftpcls(unit,colnum,frow,felem,nrows,name,status)
      colnum=2
      call ftpclj(unit,colnum,frow,felem,nrows,diameter,status)
      colnum=3
      call ftpcle(unit,colnum,frow,felem,nrows,density,status)

C     close the FITS file and free the unit number
 8    call ftclos(unit, status)
      call ftfiou(unit, status)

C     check for any error, and if so print out error messages
 9    if (status .gt. 0)call printerror(status)
      end
```

2.4 COPYHDU - copy HDUs to a new file

This routine copies selected HDUs from one FITS file to another. In this example the input FITS file has 3 HDUs (the Primary array, a TABLE extension, and a BINTABLE extension). The 1st and 3rd HDU are copied into a new output FITS file.

1. The STATUS parameter must always be initialized. See the WRITEIMAGE program for a more detailed discussion.
2. The DELETEFILE routine (described later in the Utilities section of the Cookbook) simply deletes the output FITS file if it already exists so that we can then regenerate it.
3. FTGIOU allocates a logical unit number for use when opening a file. See the WRITEIMAGE program for a more detailed discussion.
4. The input FITS file is opened with READONLY access by specifying READWRITE = 0.
5. FTINIT creates a new empty output FITS file.
6. FTCOPY copies the current HDU from the input FITS file to the output file. The MOREKEY parameter allows one to reserve space for additional header keywords when the HDU is created. FITSIO will automatically insert more header space if required, so programmers do not have to reserve space ahead of time, although it is more efficient to do so if it is known that more keywords will be appended to the header.
7. FTCRHD creates a new empty FITS extension in the output file following the current extension and moves to it.
8. FTMAHD moves the internal FITSIO pointer to the 3rd HDU (as specified here by the second parameter) which in this case is the binary table extension created by the previous WRITE-BINARY routine.
9. FTCOPY now copies the binary table from the input FITS file to the output file.
10. The FITS file must always be closed before exiting the program.
11. Any unit numbers allocated with FTGIOU must be freed with FTFIOU. Giving -1 for the value of the first argument causes all previously allocated unit numbers to be released.
12. PRINTERROR is a general routine to print out error messages and is described later in the Utilities section of this cookbook.

```
      subroutine copyhdu

C     copy the 1st and 3rd HDUs from the input file to a new FITS file

      integer status,inunit,outunit,readwrite,blocksize,morekeys,hdutype
      character infilename*40,outfilename*40

 1    status=0
C     Name of the FITS files:
      infilename='ATESTFILEZ.FITS'
      outfilename='BTESTFILEZ.FITS'

C     Delete the file if it already exists, so we can then recreate it
 2    call deletefile(outfilename,status)

C     Get  unused Logical Unit Numbers to use to open the FITS files
 3    call ftgiou(inunit,status)
      call ftgiou(outunit,status)

C     open the input FITS file, with readonly access
      readwrite=0
 4    call ftopen(inunit,infilename,readwrite,blocksize,status)

C     create the new empty FITS file with the standard block size
      blocksize=1
 5    call ftinit(outunit,outfilename,blocksize,status)

C     copy the primary array from the input file to the output file
      morekeys=0
 6    call ftcopy(inunit,outunit,morekeys,status)

C     append/create a new empty extension on the end of the output file
 7    call ftcrhd(outunit,status)

C     skip to the 3rd extension in the input file
 8    call ftmahd(inunit,3,hdutype,status)

C     copy this extension from the input file to the output file
 9    call ftcopy(inunit,outunit,morekeys,status)

C     close the FITS file and free the unit numbers
 10   call ftclos(inunit, status)
      call ftclos(outunit, status)
 11   call ftfiou(-1, status)

C     check for any error, and if so print out error messages
 12   if (status .gt. 0)call printerror(status)
      end
```

2.5 SELECTROWS - copy rows from one table to another

This routine copies selected rows from an input table into a new output FITS table. In this example all the rows in the input table that have a value of the DENSITY column less that 3.0 are copied to the output table. This program illustrates several generally useful techniques, including:

- how to locate the end of a FITS file
- how to create a table when the total number of rows in the table is not known until the table is completed
- how to efficiently copy entire rows from one table to another.

This program is continued over 2 pages in this cookbook.

1. The STATUS parameter must always be initialized. See the WRITEIMAGE program for a more detailed discussion.
2. FTGIOU allocates a logical unit number for use when opening a file. See the WRITEIMAGE program for a more detailed discussion.
3. The input FITS file is opened with READONLY access, and the output FITS file is opened with WRITE access.
4. FTMAHD moves to the 3rd HDU (or the 2nd extension) in the input FITS file.
5. This do-loop illustrates how to move to the last extension in any FITS file. The call to FTMRHD moves one extension at a time through the FITS file until an 'End-of-file' status value (= 107) is returned.
6. After locating the end of the FITS file, it is necessary to reset the status value to zero and also clear the internal error message stack in FITSIO. The previous 'End-of-file' error will have produced an unimportant message on the error stack which can be cleared with the call to the FTCMSG routine (which has no arguments).
7. FTCRHD appends a new empty HDU on the output file.
8. FTGHSP returns the number of existing keywords in the header as well as the amount of space currently available for more keywords.
9. This do-loop of calls to FTGREC and FTPREC copies all the keywords from the input to the output FITS file. Notice that the specified number of rows in the output table, as given by the NAXIS2 keyword, will be incorrect. This value will be modified later after it is known how many rows will be in the table, so it does not matter how many rows are specified initially.
10. FTRDEF forces FITSIO to scan the keywords that have just been written to the output FITS file to determine the structure of the table. This must be known before FITSIO can write any data to the table.

```
      subroutine selectrows

C     select rows from an input table and copy them to the output table

      integer status,inunit,outunit,readwrite,blocksize,hdutype
      integer nkeys,nspace,naxes(2),nfound,colnum,frow,felem
      integer noutrows,irow,temp(100),i
      real nullval,density(6)
      character infilename*40,outfilename*40,record*80
      logical exact,anynulls

 1    status=0
C     Names of the FITS files:
      infilename='ATESTFILEZ.FITS'
      outfilename='BTESTFILEZ.FITS'

C     Get  unused Logical Unit Numbers to use to open the FITS files
 2    call ftgiou(inunit,status)
      call ftgiou(outunit,status)

C     open the FITS files, with the appropriate read/write access
      readwrite=0
 3    call ftopen(inunit,infilename,readwrite,blocksize,status)
      readwrite=1
      call ftopen(outunit,outfilename,readwrite,blocksize,status)

C     move to the 3rd HDU in the input file (a binary table in this case)
 4    call ftmahd(inunit,3,hdutype,status)

C     move to the last extension in the output file
 5    do while (status .eq. 0)
          call ftmrhd(outunit,1,hdutype,status)
      end do

      if (status .eq. 107)then
C         this is normal; it just means we hit the end of file
          status=0
 6        call ftcmsg
      end if

C     create a new empty extension in the output file
 7    call ftcrhd(outunit,status)

C     find the number of keywords in the input table header
 8    call ftghsp(inunit,nkeys,nspace,status)

C     copy all the keywords from the input to the output extension
 9    do i=1,nkeys
          call ftgrec(inunit,i,record,status)
          call ftprec(outunit,record,status)
      end do

C     force FITSIO to read the output file keywords to define the data structure
 10   call ftrdef(outunit,status)
```

SELECTROWS - (continued)

11. FTGKNJ is used to get the value of the NAXIS1 and NAXIS2 keywords, which define the width of the table in bytes, and the number of rows in the table.

12. FTGCNO gets the column number of the 'DENSITY' column; the column number is needed when reading the data in the column. The EXACT parameter determines whether or not the match to the column names will be case sensitive.

13. FTGCVE reads all 6 rows of data in the 'DENSITY' column. The number of rows in the table is given by NAXES(2). Any null values in the table will be returned with the corresponding value set to -99 (= the value of NULLVAL). The ANYNULLS parameter will be set to TRUE if any null values were found while reading the data values in the table.

14. FTGTBB and FTPTBB are low-level routines to read and write, respectively, a specified number of bytes in the table, starting at the specified row number and beginning byte within the row. These routines do not do any interpretation of the bytes, and simply pass them to or from the FITS file without any modification. This is a faster way of transferring large chunks of data from one FITS file to another, than reading and then writing each column of data individually. In this case an entire row of bytes (the row length is specified by the naxes(1) parameter) is transferred. The datatype of the buffer array (TEMP in this case) is immaterial so long as it is declared large enough to hold the required number of bytes.

15. After all the rows have been written to the output table, the FTMKYJ routine is used to overwrite the NAXIS2 keyword value with the correct number of rows. Specifying '&' for the comment string tells FITSIO to keep the current comment string in the keyword and only modify the value. Because the total number of rows in the table was unknown when the table was first created, any value (including 0) could have been used for the initial NAXIS2 keyword value.

16. The FITS file must always be closed before exiting the program.

17. Any unit numbers allocated with FTGIOU must be freed with FTFIOU. Giving -1 for the value of the first argument causes all previously allocated unit numbers to be released.

18. PRINTERROR is a general routine to print out error messages and is described later in the Utilities section of this cookbook.

```
C     get the width of the table (in bytes) and the number of rows
 11   call ftgknj(inunit,'NAXIS',1,2,naxes,nfound,status)

C     find which column contains the DENSITY values
      exact=.false.
 12   call ftgcno(inunit,exact,'DENSITY',colnum,status)

C     read the DENSITY column values
      frow=1
      felem=1
      nullval=-99.
 13   call ftgcve(inunit,colnum,frow,felem,naxes(2),nullval,
     &            density,anynulls,status)

C     If the density is less than 3.0, copy the row to the output table
      noutrows=0
      do irow=1,naxes(2)
          if (density(irow) .lt. 3.0)then
              noutrows=noutrows+1
 14           call ftgtbb(inunit,irow,1,naxes(1),temp,status)
              call ftptbb(outunit,noutrows,1,naxes(1),temp,status)
          end if
      end do

C     update the NAXIS2 keyword with the correct no. of rows in the output file
 15   call ftmkyj(outunit,'NAXIS2',noutrows,'&',status)

C     close the FITS file and free the unit numbers
 16   call ftclos(inunit, status)
      call ftclos(outunit, status)
 17   call ftfiou(-1, status)

C     check for any error, and if so print out error messages
 18   if (status .gt. 0)call printerror(status)
      end
```

2.6 READHEADER - read and print header keywords

This subroutine will read and print out all the header keywords in every extension of a FITS file.

1. The STATUS parameter must always be initialized. See the WRITEIMAGE program for a more detailed discussion.

2. FTGIOU allocates a logical unit number for use when opening a file. See the WRITEIMAGE program for a more detailed discussion.

3. The FITS file is opened with the READWRITE parameter set = 0 to just give read access to the file. Set it equal to 1 if one needs read and write access to the FITS file. The returned BLOCKSIZE parameter gives the size of each logical record, but this is only relevant on machines such as VAX/VMS which have a record oriented file system.

4. The FTGHSP subroutine returns the number of existing keywords in the current header data unit (CHDU), not counting the required END keyword, and also returns the number of blank records currently available for writing more keywords. FITSIO will automatically add more space for keywords if necessary so programmers generally do not have to be concerned about the amount of free space in the header.

5. The FTGREC routine returns the contents of the particular header keyword specified by the second parameter. This do-loop prints out every keyword record in the current HDU.

6. The FTMRHD subroutine attempts to move to the next HDU, as specified by the second parameter. This subroutine moves by a relative number of HDUs from the current HDU. The related FTMAHD routine may be used to move to an absolute HDU number in the FITS file. If the end-of-file is encountered when trying to move to the specified extension, then a status = 107 is returned.

7. If the program reachs this branch of the IF...ELSEIF block, then the previous attempt to move to the next FITS extension was successful.

8. This branch of the IF...ELSEIF block is reached whenever the end-of-file is detected.

9. The FITS file must always be closed before exiting the program. Any unit numbers allocated with FTGIOU must be freed with FTFIOU.

10. PRINTERROR is a general routine to print out error messages and is described later in the Utilities section of this cookbook.

```
      subroutine readheader

C     Print out all the header keywords in all extensions of a FITS file

      integer status,unit,readwrite,blocksize,nkeys,nspace,hdutype,i
      character filename*80,record*80

 1    status=0

C     Get an unused Logical Unit Number to use to open the FITS file
 2    call ftgiou(unit,status)

C     name of FITS file
      filename='ATESTFILEZ.FITS'

C     open the FITS file, with read-only access
      readwrite=0
 3    call ftopen(unit,filename,readwrite,blocksize,status)

100   continue

C     Determine the number of keywords in the header
 4    call ftghsp(unit,nkeys,nspace,status)

C     Read each 80-character keyword record, and print it out
      do i = 1, nkeys
 5        call ftgrec(unit,i,record,status)
          print *,record
      end do

C     Print out and END record, and a blank line to mark the end of the header
      if (status .eq. 0)then
          print *,'END'
          print *,' '
      end if

C     try moving to the next extension in the FITS file, if it exists
 6    call ftmrhd(unit,1,hdutype,status)

      if (status .eq. 0)then
C         success, so loop back and print out keywords in this extension
 7        go to 100

      else if (status .eq. 107)then
C         hit end of file, so quit
 8        print *,'***** END OF FILE *****'
          status=0
          call ftcmsg
      end if

C     close the file, free the unit number, and exit
 9    call ftclos(unit, status)
      call ftfiou(unit, status)

C     check for any error, and if so print out error messages
 10   if (status .gt. 0)call printerror(status)
      end
```

2.7 READIMAGE - read the pixel values in a FITS image

This example program reads the FITS image and determines the minimum and maximum pixel values. Rather than reading the entire image in at once (which could require a very large array), the image is read in pieces, 100 pixels at a time.

1. The STATUS parameter must always be initialized. See the WRITEIMAGE program for a more detailed discussion.
2. FTGIOU allocates a logical unit number for use when opening a file. See the WRITEIMAGE program for a more detailed discussion.
3. The FITS file is opened with READONLY access.
4. The FTGKNJ routine reads the set of NAXISn indexed keywords. In this case, we expect to find the NAXIS1 and NAXIS2 keywords.
5. A check is made to ensure that both NAXISn keywords were found by FTGKNJ.
6. FTGPVE is used to read in the next group of 100 pixels in the image. This loop is repeated until all the pixels have been read.
7. The FITS file must always be closed before exiting the program. Any unit numbers allocated with FTGIOU must be freed with FTFIOU.
8. PRINTERROR is a general routine to print out error messages and is described later in the Utilities section of this cookbook.

```
      subroutine readimage

C     Read a FITS image and determine the minimum and maximum pixel value

      integer status,unit,readwrite,blocksize,naxes(2),nfound
      integer group,firstpix,nbuffer,npixels,i
      real datamin,datamax,nullval,buffer(100)
      logical anynull
      character filename*80

 1    status=0

C     Get an unused Logical Unit Number to use to open the FITS file
 2    call ftgiou(unit,status)

C     open the FITS file previously created by WRITEIMAGE
      filename='ATESTFILEZ.FITS'
      readwrite=0
 3    call ftopen(unit,filename,readwrite,blocksize,status)

C     determine the size of the image
 4    call ftgknj(unit,'NAXIS',1,2,naxes,nfound,status)

C     check that it found both NAXIS1 and NAXIS2 keywords
 5    if (nfound .ne. 2)then
          print *,'READIMAGE failed to read the NAXISn keywords.'
          return
       end if

C     initialize variables
      npixels=naxes(1)*naxes(2)
      group=1
      firstpix=1
      nullval=-999
      datamin=1.0E30
      datamax=-1.0E30

      do while (npixels .gt. 0)
C         read up to 100 pixels at a time
          nbuffer=min(100,npixels)

 6        call ftgpve(unit,group,firstpix,nbuffer,nullval,
     &            buffer,anynull,status)

C         find the min and max values
          do i=1,nbuffer
              datamin=min(datamin,buffer(i))
              datamax=max(datamax,buffer(i))
          end do

C         increment pointers and loop back to read the next group of pixels
          npixels=npixels-nbuffer
          firstpix=firstpix+nbuffer
      end do

C     print out the min and max values
      print *,'Min and max values in the image are:',datamin,datamax

C     close the file and free the unit number
 7    call ftclos(unit, status)
      call ftfiou(unit, status)

C     check for any error, and if so print out error messages
 8    if (status .gt. 0)call printerror(status)
      end
```

2.8 READTABLE - read data from a FITS table

This example reads and prints out all the data in the ASCII and the binary tables that were previously created by WRITEASCII and WRITEBINTABLE. Note that the exact same FITSIO routines are used to read both types of tables.

1. The STATUS parameter must always be initialized. See the WRITEIMAGE program for a more detailed discussion.

2. FTGIOU allocates a logical unit number for use when opening a file. See the WRITEIMAGE program for a more detailed discussion.

3. The FITS file is opened with READONLY access.

4. We loop though the main block of code 2 times, first to read the ASCII table, then to read the binary table.

5. FTMRHD moves forward one HDU in the FITS file. The HDUTYPE parameter indicates whether the extension is an ASCII table (= 1) or a binary table (= 2).

6. FTGKNS is used to read the indexed array of TTYPEn keywords that specify the names of the columns in the tables.

7. FTGCVS reads the NAMES from the first column of the table.

8. FTGCVJ reads the DIAMETER values from the second column.

9. FTGCVE reads the DENSITY values from the third column.

10. The FITS file must always be closed before exiting the program. Any unit numbers allocated with FTGIOU must be freed with FTFIOU.

11. PRINTERROR is a general routine to print out error messages and is described later in the Utilities section of this cookbook.

```
      subroutine readtable

C     read and print data values from an ASCII or binary table

      integer status,unit,readwrite,blocksize,hdutype,ntable
      integer felem,nelems,nullj,diameter,nfound,irow,colnum
      real nulle,density
      character filename*40,nullstr*1,name*8,ttype(3)*10
      logical anynull

 1    status=0

C     Get an unused Logical Unit Number to use to open the FITS file
 2    call ftgiou(unit,status)

C     open the FITS file previously created by WRITEIMAGE
      filename='ATESTFILEZ.FITS'
      readwrite=0
 3    call ftopen(unit,filename,readwrite,blocksize,status)

C     loop twice, first reading the ASCII table, then the binary table
 4    do ntable=1,2

C         move to the next extension
 5        call ftmrhd(unit,1,hdutype,status)

          print *,' '
          if (hdutype .eq. 1)then
              print *,'Extension ',ntable,' is an ASCII table.'
          else if (hdutype .eq. 2)then
              print *,'Extension ',ntable,' is a binary table.'
          end if

C         read the TTYPEn keywords, which give the names of the columns
 6        call ftgkns(unit,'TTYPE',1,3,ttype,nfound,status)
          write(*,2000)ttype
2000      format(8x,3a10)

C         read the data, one row at a time, and print them out
          felem=1
          nelems=1
          nullstr=' '
          nullj=0
          nulle=0.
          do irow=1,6
              colnum=1
 7            call ftgcvs(unit,colnum,irow,felem,nelems,nullstr,name,
     &                    anynull,status)
              colnum=2
 8            call ftgcvj(unit,colnum,irow,felem,nelems,nullj,diameter,
     &                    anynull,status)
              colnum=3
 9            call ftgcve(unit,colnum,irow,felem,nelems,nulle,density,
     &                    anynull,status)
              write(*,2001)irow,name,diameter,density
2001          format(i4,a10,i10,f10.2)
          end do
      end do

C     close the file and free the unit number
 10   call ftclos(unit, status)
      call ftfiou(unit, status)

C     check for any error, and if so print out error messages
 11   if (status .gt. 0)call printerror(status)
      end
```

Chapter 3

Utility Subroutines

This chapter lists various utility subroutines that are used by the programs in the previous chapter.

3.1 PRINTERROR - print FITSIO error messages

This subroutine prints out the descriptive text corresponding to the error status value and prints out the contents of the internal error message stack generated by FITSIO whenever an error occurs.

1. The FTGERR subroutine returns a descriptive 30-character text string that corresponds to the integer error status number. A complete list of all the error numbers can be found in the back of the FITSIO User's Guide.

2. FITSIO usually generates an internal stack of error messages whenever an error occurs. These messages provide much more information on the cause of the problem than can be provided by the single integer error status value. The FTGMSG subroutine retrieves the oldest message from the stack and shifts any remaining messages on the stack down one position. FTGMSG is called repeatedly until a blank message is returned, which indicates that the stack is empty. Each error message may be up to 80 characters in length. Another subroutine, called FTCMSG, is available to simply clear the whole error message stack in cases where one is not interested in the contents.

```
      subroutine printerror(status)

C     Print out the FITSIO error messages to the user

      integer status
      character errtext*30,errmessage*80

C     check if status is OK (no error); if so, simply return
      if (status .le. 0)return

C     get the text string which describes the error
 1    call ftgerr(status,errtext)
      print *,'FITSIO Error Status =',status,': ',errtext

C     read and print out all the error messages on the FITSIO stack
 2    call ftgmsg(errmessage)
      do while (errmessage .ne. ' ')
          print *,errmessage
          call ftgmsg(errmessage)
      end do
      end
```

3.2 DELETEFILE - delete a FITS file

This subroutine simply deletes a specified FITS file if it exists. Some of the programs in this cookbook which create a test FITS file use this routine to delete the file if it already exists from a previous run of the program.

Programmers may wish to use a similar technique to delete a partially written FITS file if an error occurs that prevents the file from being completed successfully.

1. FTGIOU returns an unused Fortran logical unit number for use when opening the file. This routine must only be used if ALL unit numbers used by the program are allocated though it, because it will have no knowledge of any other unit numbers that the program may have otherwise used.

2. FTOPEN attempts to open the FITS file. There are 3 possible outcomes: a returned status = 0 means the FITS file was opened successfully; status = 103 means the file doesn't exist; any other status value indicates that the file exists but it could not be read successfully, probably because the format does not conform to the FITS standard.

3. FTDELT deletes the file associated with the particular unit number. In more normal circumstances, one would use the FTCLOS subroutine to close a file without deleting it.

4. FTCMSG clears the internal FITSIO error message stack. In this case the previous failure to open the FITS file will have generated a message on the stack which is of no importance.

5. In this case there was some sort of problem reading the FITS file, probably because the format does not conform to the FITS standard. Ignore this error and delete the file anyway after clearing the error message stack of any superfluous messages.

6. FTFIOU frees the logical unit number previously allocated with ftgiou.

```
      subroutine deletefile(filename,status)

C     A simple little routine to delete a FITS file

      integer status,unit,blocksize
      character*(*) filename

C     simply return if status is greater than zero
      if (status .gt. 0)return

C     Get an unused Logical Unit Number to use to open the FITS file
 1    call ftgiou(unit,status)

C     try to open the file, to see if it exists
 2    call ftopen(unit,filename,1,blocksize,status)

      if (status .eq. 0)then
C         file was opened;  so now delete it
 3        call ftdelt(unit,status)
      else if (status .eq. 103)then
C         file doesn't exist, so just reset status to zero and clear errors
          status=0
 4        call ftcmsg
      else
C         there was some other error opening the file; delete the file anyway
          status=0
 5        call ftcmsg
          call ftdelt(unit,status)
      end if

C     free the unit number for later reuse
 6    call ftfiou(unit, status)
      end
```