Chapter 6

Disk files

6.1 Overview

Most images, uv data sets, scratch files, and other information in the AIPS system are kept in cataloged disk files. The purpose of this chapter is to describe the general techniques for accessing data in disk files.

Associated with each image or uv data file may be a number of auxiliary files known as "extension" files containing information about the main file. Examples of extension files are the history file, CLEAN components files and antenna files. Details of the structure of the various files used in AIPS programs are described in Appendix C. Except for the image and uv data files, the details of the file structure will not be described here.

The amount of data in the image and uv data files can be rather large, so it is important that the routines accessing them be relatively efficient. This efficiency comes at the cost of increased complexity. There are a number of features of AIPS I/O routines for handling large amounts of data which are designed for efficiency.

1. Fixed record length. All files internal to AIPS have a fixed logical record length. This allows the I/O routines to block disk transfers into a number of logical records.
2. Large double buffered transfers. The upper level I/O routines automatically make data transfers as large as possible and when possible double buffer the transfers.
3. Visible I/O buffers. To avoid an in-core transfer of all data, most AIPS routines work directly from the I/O buffer.

Extension files are handled somewhat differently. Since the amount of data in these files is rather small, friendlier, but less efficient, techniques are used. Logical records have a fixed length, but the basic I/O routine (TABIO) returns the data in an array which allows implementation of data structures.

This chapter discusses the various aspects of disk files — creating, destroying, reading, writing, etc. The cataloging of these files has been covered in a previous chapter. A typical programmer will not need to understand all of the material in this chapter to program effectively in AIPS. The detailed descriptions of the major routines discussed will be given at the end of the chapter.

6.2 Types of Files

AIPS has two logically different types of files, which on some machines are also physically different. The first type, known as regular disk files, is used mainly for extension files. I/O to this type of file is done in 512-AIPS byte blocks. The second type of file, known as "map" files, is used for image and uv data files. I/O to this type of file is usually done in the double buffered mode with large size transfers. (Double buffering is when the program works out of one half of a buffer, while the other half is being read from, or written to, the external device.) Both of these types of files may be expanded or contracted.

The principle distinction between the two types of files are the file creation and opening routines. Many of the higher level creation and file open routines hide this distinction from the programmer. These routines will be discussed later in this chapter.

6.3 File Management

AIPS has a set of utility routines for creating and managing disk files. The four functions covered in this section are file creation, destruction, extension and contraction.

6.3.1 Creating Files

There are several higher level file creation routines, one for each of several applications. These applications are image files, UV data files, scratch files, general extension files and history files. The basic file creation routine is ZCREAT.

- MCREAT creates and catalogs an image file (type "MA") using the description of the file contained in a catalog header record passed to MCREAT via the common /MAPHDR/ (INCLUDE DCAT.INC). All information in the header defining the size and name of the file must be filled in before calling MCREAT. The catalog header record is described in detail in another chapter.
- UVCREA creates and catalogs a uv data file (type "UV") using the description of the file contained in the catalog header record passed to UVCREA in the common /MAPHDR/ (INCLUDE DCAT.INC). The catalog header record must be sufficiently complete to determine the name, class, etc. and size of the required file.
- SCREAT will create scratch files using the /CFILES/ (INCLUDE DFIL.INC) common system; thus the scratch files will be automatically deleted when the task calls the shutdown routine DIE. Scratch files are cataloged as type "SC" files. Use of SCREAT is described in more detail in the chapter describing tasks.
- TABINI. The creation of most tables extension files is hidden from the casual programmer in the create/open/initialize routine TABINI. TABINI will be discussed in more detail in the chapter on tables.
- HICREA. The creation of history files is normally hidden in the upper level routine HISCOP. The use of HISCOP and HICREA are described in more detail in the chapter on writing tasks.
- ZCREAT. The basic file creation routine is ZCREAT. If none of the other file creation routines are applicable, then use ZCREAT. ZCREAT needs the physical name of the file and the size of the file in bytes. ZCREAT does not catalog the file created.

6.3.2 Example Using ZCREAT

The use of ZCREAT is demonstrated in the following:

```
      INTEGER   IRET, NX, NY, NP(2), NBYTE, IVOL, CNO, IVER, IERR
      CHARACTER PHNAME*48
      LOGICAL   MAP
      PARAMETER (MAP = .TRUE.)
      INCLUDE 'INCS:DDCH.INC'
           ...
C                                       NX, NY are the size of an
C                                       image.  Determine file size.
      NP(1) = NX
      NP(2) = NY
      CALL MAPSIZ (2, NP, NBYTE)
C                                       Size now in NBYTE
```

```
C                                        Make physical name.
C                                        IVOL = disk number
C                                        CNO = catalog slot number
C                                        IVER = extension file
C                                            version number.
C                                            1 for main cataloged files.
      CALL ZPHFIL ('MA', IVOL, CNO, IVER, PHNAME, IERR)
C                                        filename now in PHNAME.
C                                        (error if IERR not 0)
C                                        Create file of type 'MA'
      CALL ZCREAT (IVOL, PHNAME, NBYTE, MAP, IERR)
C                                        Test for errors...
```

In the example above, a map file was created large enough to hold a NX by NY image using the routine MAPSIZ to compute the correct size for the file. To catalog this file a catalog header record should be constructed and calls made to CATDIR and CATIO *before* the call to ZPHFIL to get the catalog slot number needed to form the physical name of the file. A detailed description of the calling sequence for ZCREAT can be found at the end of this chapter. (In practice, one would use MCREAT to catalog and create the file shown in the example above.)

6.3.3 Destruction Routines

There are a number of special purpose file destruction routines; the basic file destruction routine is ZDESTR. A brief description is given here of these utility routines; a description of the call sequence is given at the end of this chapter.

- MDESTR will delete a catalog entry for a file, delete all extension files for that file, and then delete the file. The file must be in the REST state. Since catalog files can be marked "WRITE — Destroy if task fails" which will cause the shutdown routine DIE to destroy the file there is seldom a need to call MDESTR directly. MDESTR will destroy either cataloged image or uv data files.

- SNDY will destroy scratch files described in the /CFILES/ common (INCLUDE DFIL.INC). SNDY is called by the shutdown utility DIE so tasks do not have to call it separately.

- ZDESTR is the basic file destruction routine. ZDESTR will not uncatalog the file destroyed. CATDIR should be used to uncatalog a cataloged file destroyed.

6.3.4 Expansion and Contraction of Files

All files can be both expanded and compressed. Since most extension file access is by TABIO, the expansion of extension files is hidden from the programmer. Expansion of files is done with routine ZEXPND and compression is done using routine ZCMPRS. Details of the call sequences of these routines are given at the end of this chapter.

6.4 I/O to Disk Files

There are a number of steps necessary in order to access a disk file. Normal Fortran I/O hides a number of these steps but they are all visible in at least some AIPS applications. This increased complexity of the I/O system gives the programmer a high degree of control over how the I/O is actually done. One or more of the steps in accessing a file may be performed with a single call. In general, access of a disk file is as follows:

1. Forming the physical name of the file. The AIPS utility ZPHFIL is always used for this purpose. The name is derived from file type, the disk number, catalog slot number, version number and user ID number. Also a revision code is usually included in the physical name of the file so that versions of AIPS with incompatable file formats can coexist in the same directories. The file type of image files is "MA", of uv data files is "UV" and of scratch files is "SC". The disk number and catalog slot number for cataloged files may have to be obtained from the AIPS utility routine CATDIR before calling ZPHFIL. This step is incorporated in a number of routines such as SCREAT, TABINI and MAPOPN.

2. Opening the file. This is done with routine ZOPEN for binary files and ZTOPEN or ZTXOPN for text files. In either case, the file must be given a logical unit number (LUN) and the opening routine returns a pointer to the AIPS I/O table (FTAB) which, with the LUN, must be used in all subsequent calls. This step is incorporated in the routines TABINI and MAPOPN.

3. Initializing the transfers. The AIPS higher level I/O routines need to be told a number of parameters about the data transfers, such as whether a read or write is desired, the size and number of logical records, and the location and size of the buffer to be used. In several cases the range of data desired can also be specified. This step is usually done in one of the specialized routines to be described later.

4. Data transfers. This is when the data is transferred from the disk to the specified buffer or vice versa. Actual data transfers are done by Direct Memory Access (DMA) and are usually in large blocks for "map" files and in 512-byte blocks for non-map (extension) files. Since the transfers usually consist of a number of logical records, the programmer is unaware of when transfers actually take place. Because the programs frequently work directly from the I/O buffer, many of the I/O routines return a pointer to the first word in the buffer of the next logical record.

5. Flushing the buffer (writing only). When all calls to disk write routines are complete, there may still be data in the buffer which has not been written. In this case, a call must be made to the appropriate I/O routine telling it to flush the buffer to disk.

6. Closing the file. When all operations on a file are complete the file needs to be closed. This is usually done with an explicit call to the appropriate close routine.

6.4.1 Upper Level I/O Routines

There are a number of AIPS upper level I/O routines which do most of the bookkeeping. The following is a short description of the more commonly used of these; detailed descriptions of the call sequences are found at the end of the chapter. The use of many of these routines is discussed later in this chapter.

- TABINI opens and initializes an table extension file, will create and catalog the extension file if necessary. See the chapter on tables for more details.
- TABIO does random access mixed reads and writes to extension tables. TABIO deals with one logical record at a time in an array which can be used as a data structure. TABIO takes care of file expansion and other bookkeeping chores. Requires initialization by TABINI.
- MAPOPN finds a cataloged image or uv data file in the catalog, opens it and returns the catalog header and marks the catalog status.
- MINIT initializes I/O for image files; can specify a subimage for reads.
- MDISK does double buffered I/O for image files; requires initialization by MINIT.
- UVINIT initializes I/O for uv data files; can specify a starting visibility record number.
- UVDISK does double buffered I/O for uv data files; requires initialization by UVINIT.
- MAPCLS closes a cataloged image or uv data file, updates the catalog header block if requested and clears the catalog status.

6.4.2 Logical unit numbers

Many logical unit numbers in AIPS have special meanings which indicate to the I/O routines what kind of device or file is involved. The information about which LUN corresponds to which device is contained in a table (DEVTAB) in the device characteristics common (INCLUDE DDCH.INC). AIPS has 50 defined LUN values, i.e., DEVTAB has 50 entries, and the type of device or file type for each LUN is given in DEVTAB with the following codes:

```
DEVTAB(LUN) = 0   LUN is for disk file requiring I/O control area in
                  FTAB.  Multi-record I/O is possible.
DEVTAB(LUN) = 1   Device not requiring I/O control area in FTAB.
                  I/O done by Fortran (terminals, printer/plotter).
DEVTAB(LUN) = 2   LUN is for device requiring I/O control area in
                  FTAB.  Multi-record I/O not allowed (e.g., tapes)
DEVTAB(LUN) = 3   Similar to 1.  VAX uses this code to defer opens
                  from ZOPEN to ZTOPEN for text files.
DEVTAB(LUN) = 4   LUN is for TV device requiring special I/O routine
                  and normal I/O control area in FTAB.
```

In addition, many LUNs have predefined values as shown in the following table.

LUN	Use
1	Line printer
2	Plotter
3	Reserved
4	Input to batch processors
5	Input CRT
6	Output CRT
7	Graphics CRT
8	Array Processor (roller)
9	TV device
10	POPS "run" files
11	POPS "help" files
12	Log/error file (used by MSGWRT).
13	Task communication file.
14	POPS "memory" file
15	Catalog files.
16 - 25	Map (image or uv data) files.
26	Graphics files
27 - 30	General (non-map) disk files.
31 - 3?	Magnetic tape drives (31 - 30+NTAPED)

6.4.3 Contents of the Device Characteristics Commons

The device Characteristics commons, obtained from the INCLUDE DDCH.INC contains a number of useful parameters about the host system.

6.4.4 Image Files

A disk image file contains an ordered, binary sequence of pixel values with logical records consisting of single "rows" of the image. The pixel values are arranged in the order defined in the catalog header block, the first axis going the fastest. Blanking of pixels is allowed by use of a special value (magic value blanking) specified by the header. For more information about the catalog header and the typical axes used, see the chapter on the catalog.

Image files are stored on the disk with each row beginning on a block boundary. An exception to this is when multiple rows will fit into a single block, in which case multiple rows can be in a given disk block. In this latter case, rows are not allowed to span block boundaries.

Opening Image Files

The simplest way to find, open and close a cataloged image file is with the routines MAPOPN and MAPCLS. These routines and the alternate ways to find an image in the catalog are discussed in the chapter on the catalog and details of the call sequence are found at the end of this chapter.

If the use of MAPOPN and MAPCLS is not appropriate to open and close the image file, then the routines ZPHFIL, ZOPEN and ZCLOSE are to be used to (1) form the physical name of the file, (2) open the file, both in the AIPS and system tables, and (3) close the file when done. The details of these routines are given at the end of this chapter. These operations are demonstrated in the following example.

```
      INTEGER    IRET, CNO, IVOL, IVER, LUN, IND
      LOGICAL    MAP, EXCL, WAIT
      CHARACTER  PHNAME*48
      PARAMETER (MAP = .TRUE.)
      PARAMETER (EXCL = .TRUE.)
      PARAMETER (WAIT = .TRUE.)
      PARAMETER (LUN = 16)
           .
           .
      DATA IVER /1/
           .
           .
C                                       Make physical name.
C                                       'MA' = file type
C                                       IVOL = disk number
C                                       CNO = catalog slot number
C                                           (arbitrary for
C                                           uncataloged files).
C                                       IVER = extension file
C                                           version number.
C                                           1 for main cataloged
C                                           files. Arbitrary
C                                           otherwise.
      CALL ZPHFIL ('MA', IVOL, CNO, IVER, PHNAME, IRET)
C                                       filename now in PHNAME.
C                                       (error if IRET not 0)
C                                       Open file
      CALL ZOPEN (LUN, IND, IVOL, PHNAME, MAP, EXCL, WAIT, IRET)
C                                       Test for errors (IRET not 0)
           .
           .
    ( I/O to file )
```

```
             .
             .
C                                         Close file.
      CALL ZCLOSE (LUN, IND, IRET)
```

MINIT and MDISK

Once the image file is opened, I/O is normally initialized by a call to MINIT; I/O is done by calls to MDISK, with a final call to MDISK to flush the buffer, if necessary. MINIT sets up the bookkeeping for one plane of an image at a time; if multiple planes are to be read, multiple calls to MINIT must be made. A rectangular window in a given plane can be specified to MINIT, and it can be instructed to read or write the rows in reverse order by reversing the values of WIN(2) and WIN(4). A subimage cannot be specified for write.

Due to the use of buffer pointers, MDISK must be called for WRITE *before* placing data into the buffer. This produces a rather strange logic flow, but is necessary. Details of the call sequences to MINIT and MDISK are given at the end of this chapter.

Multi-plane Images (COMOFF)

If the image has more than two dimensions, planes parallel to the first plane can be accessed using the block offset argument to MINIT. The subroutine COMOFF is to be used to compute the block offset. The block offset is an integer whose value for the first plane is 1. COMOFF returns a value which is to be added to the block offset for the first plane.

An example of the use of COMOFF to compute the block offset:

```
      INTEGER   BLKOF, PLARR(5), IERR
      INCLUDE 'DHDR.INC'
      INCLUDE 'DDCH.INC'
      INCLUDE 'DCAT.INC'
            .
            .
C                                         Get second plane on third
C                                         axis, first pixel on
C                                         the remaining axes.
      PLARR(1) = 2
      PLARR(2) = 1
      PLARR(3) = 1
      PLARR(4) = 1
      PLARR(5) = 1
C                                         PLARR specifies desired plane
C                                         Use header block from DCAT.INC
      CALL COMOFF (CATBLK(K2DIM), CATBLK(K2NAX), PLARR, BLKOF, IERR)
C                                         Add block offset for first
C                                         plane.
      BLKOF = BLKOF + 1
C                                         BLKOF now contains the value
C                                         to send to MINIT to get the
C                                         specified plane.
```

A detailed description of the call sequence for COMOFF is given at the end of this chapter.

Example of MINIT and MDISK

In the following is an example in which two files are read, the pixel values are added and a third file is written.

```
      SUBROUTINE FLADD (NX, NY, ISCR1, ISCR2, ISCR3, IERR)
C-----------------------------------------------------------------------
C   FLADD adds the values in the scratch files in the /CFILES/ common
C   (include DFIL.INC) number ISCR1 and ISCR2 and writes them in the
C   /CFILES/ scratch file number ISCR3
C   Inputs:
C      NX, NY  I  Number of pixels per row and number of rows
C      ISCR1   I  /CFILES/ scratch file number of first input file
C      ISCR2   I  /CFILES/ scratch file number of second input file
C      ISCR3   I  /CFILES/ scratch file number of output file
C   Output:
C      IERR    I  Return code, 0=>OK, otherwise error.
C-----------------------------------------------------------------------
      INTEGER   NX, NY, ISCR1, ISCR2, ISCR3, IERR
C
      INTEGER   FIND1, FIND2, FIND3, BIND1, BIND2, BIND3, BO,
     *   WIN(4), BUFSZ1, BUFSZ2, BUFSZ3, LUN1, LUN2, LUN3
      LOGICAL   T, F
      CHARACTER FILE*48
      REAL      BUFF1(4096), BUFF2(4096), BUFF3(4096)
      PARAMETER (T = .TRUE.)
      PARAMETER (F = .FALSE.)
      INCLUDE 'INCS:DMSG.INC'
      INCLUDE 'INCS:DFIL.INC'
      DATA BO, WIN /1, 4*0/
C                                       Use LUNs 16, 17, 18
      DATA LUN1, LUN2, LUN3 /16,17,18/
C-----------------------------------------------------------------------
C                                       Set buffer sizes
      BUFSZ1 = 4096 * 2
      BUFSZ2 = 4096 * 2
      BUFSZ3 = 4096 * 2
C                                        Open and init ISCR1
      CALL ZPHFIL ('SC', SCRVOL(ISCR1), SCRCNO(ISCR1), 1, FILE, IERR)
      CALL ZOPEN (LUN1, FIND1, SCRVOL(ISCR1), FILE, T, F, T, IERR)
C                                        Check for error
      IF (IERR.NE.0) THEN
         WRITE (MSGTXT,1000) IERR, 'READ', 1
         GO TO 990
         END IF
      CALL MINIT ('READ', LUN1, FIND1, NX, NY, WIN, BUFF1, BUFSZ1, BO,
     *   IERR)
C                                        Check for error
      IF (IERR.NE.0) THEN
         WRITE (MSGTXT,1010) IERR, 'READ', 1
         GO TO 990
         END IF
C                                        Open and init ISCR2
      CALL ZPHFIL ('SC', SCRVOL(ISCR2), SCRCNO(ISCR2), 1, FILE, IERR)
      CALL ZOPEN (LUN2, FIND2, SCRVOL(ISCR2), FILE, T, F, T, IERR)
```

```
C                                       Check for error
      IF (IERR.NE.0) THEN
         WRITE (MSGTXT,1000) IERR, 'READ', 1
         GO TO 990
         END IF
      CALL MINIT ('READ', LUN2, FIND2, NX, NY, WIN, BUFF2, BUFSZ2, BO,
     *   IERR)
C                                       Check for error
      IF (IERR.NE.0) THEN
         WRITE (MSGTXT,1010) IERR, 'READ',2
         GO TO 990
         END IF
C                                       Open and init ISCR3
      CALL ZPHFIL ('SC', SCRVOL(ISCR3), SCRCNO(ISCR3), 1, FILE, IERR)
      CALL ZOPEN (LUN3, FIND3, SCRVOL(ISCR3), FILE, T, F, T, IERR)
C                                       Check for error
      IF (IERR.NE.0) THEN
         WRITE (MSGTXT,1000) IERR, 'WRIT'
         GO TO 990
         END IF
      CALL MINIT ('WRIT', LUN3, FIND3, NX, NY, WIN, BUFF3, BUFSZ3, BO,
     *   IERR)
C                                       Check for error
      IF (IERR.NE.0) THEN
         WRITE (MSGTXT,1010) IERR, 'WRIT'
         GO TO 990
         END IF
C                                       Loop, adding rows.
      DO 110 I = 1,NY
C                                       Read ISCR1
         CALL MDISK ('READ', LUN1, FIND1, BUFF1, BIND1, IERR)
C                                       Check for error
         IF (IERR.NE.0) THEN
            WRITE (MSGTXT,1060) IERR, 'READ', 1
            GO TO 990
            END IF
C                                       Read ISCR2
         CALL MDISK ('READ', LUN2, FIND2, BUFF2, BIND2, IERR)
C                                       Check for error
         IF (IERR.NE.0) THEN
            WRITE (MSGTXT,1060) IERR, 'READ',2
            GO TO 990
            END IF
C                                       Write ISCR3
         CALL MDISK ('WRIT', LUN3, FIND3, BUFF3, BIND3, IERR)
C                                       Check for error
         IF (IERR.NE.0) THEN
            WRITE (MSGTXT,1060) IERR, 'WRIT'
            GO TO 990
            END IF
C                                       Add row.
         DO 100 J = 1,NX
C                                      Note: buffer pointer is to
C                                      first element so need zero
```

```
C                                       relative index for each pixel.
            J1 = J - 1
            BUFF3(BIND3+J1) = BUFF1(BIND1+J1) + BUFF2(BIND2+J1)
 100        CONTINUE
 110     CONTINUE
C                                        Flush buffer.
      CALL MDISK ('FINI', LUN3, FIND3, BUFF3, BIND3, IERR)
C                                        Check for error
      IF (IERR.NE.0) THEN
         WRITE (MSGTXT,1060) IERR, 'FINI'
         GO TO 990
         END IF
C                                        Close files.
      CALL ZCLOSE (LUN1, FIND1, IERR)
      IF (IERR.NE.0) THEN
         WRITE (MSGTXT,1060) IERR, 'CLOS', 1
         GO TO 990
         END IF
      CALL ZCLOSE (LUN2, FIND2, IERR)
      IF (IERR.NE.0) THEN
         WRITE (MSGTXT,1060) IERR, 'CLOS', 2
         GO TO 990
         END IF
      CALL ZCLOSE (LUN3, FIND3, IERR)
      IF (IERR.NE.0) THEN
         WRITE (MSGTXT,1060) IERR, 'CLOS', 3
         GO TO 990
         END IF
C                                       Finished OK.
      GO TO 999
C                                       An error has occurred - send
C                                       message
 990  CALL MSGWRT (8)
C
 999  RETURN
C-----------------------------------------------------------------------
 1000 FORMAT ('FLADD: ERROR',I3,' OPEN FOR ',A4,' FILE',I2)
 1010 FORMAT ('FLADD: ERROR',I3,' INIT FOR ',A4,' FILE',I2)
 1060 FORMAT ('FLADD: ERROR',I3,1X,A4,'ING FILE',I2)
      END
```

MINSK and MSKIP

There are some operations, such as transposing images, in which it is convenient to read every n'th row of an image. The pair of routines MINSK and MSKIP will do this operation. Descriptions of these routines can be found at the end of this chapter.

6.4.5 Image File Manipulation Routines

There are a number of AIPS utility routines available to operate on files. Many of these involve copying data from catalog files to scratch files or vice versa. Details of the call sequences to these routines are given at the end of this chapter.

- PLNGET reads a selected portion of a selected plane from a cataloged file and writes it into a specified scratch file. The output file will be zero padded and a shift of the center may be specified.

- PLNPUT writes a subregion of a scratch file image into a cataloged image.

6.4.6 UV Data Files

Interferometers take samples of the visibility (also called coherence) function of a wavefront at random locations so these data must be stored differently from images. Also, this data may be in a variety of forms, calibrated or raw, one source per file or many. The following sections describe these uv data files.

Single-source Files

The simplest form of a uv data file is the single source file which contains data from a single celestial source and is usually assumed to be calibrated and edited. A given visibility record consists of all data taken on a given baseline (with a pair of antennas) at a given time. Thus, this record may contain data for a number of frequencies and/or polarizations. Each measurment consists of a triplet of values giving the real part, the imaginary part and the weight of a given visibility sample. (This may be modified for compressed data; see the section on compressed data). The visibility records in this type of file may be in an arbitrary order depending on the application.

Each uv data file needs one or more antenna (AN) tables to describe the locations and other properties of the antennas used. The visibility data contains, in a coded form, the numbers of the antennas involved in each baseline. The antenna numbers refer to entries in the Antenna table.

There are occasions where data is sampled at a number of relatively arbitrary frequencies. For these cases we have introduced a frequency-like axis called IF. The offset of the frequency of the reference pixel in each IF group of data from the file reference frequency is given in the FQ table. Since these sets of frequencies may change in a given file (e.g. bandwidth synthesis or rotation measure studies) a random parameter in the data file labeled ('FQID') points to a given entry in the FQ table. In general, a single source file will contain only a single FQ id.

Multi-source Files

In order to allow the use of calibration and editing software, multi-source files contain data from more than one source. In addition, the data are in relatively raw form and have associated calibration and editing tables which must be applied before the data are used. This type of file has an index and must be in strict time-baseline order. The structure of multi-source data files is very similar to the single source file. The use of multi-source files is described in detail in the chapter on Calibration and Editing in Volume 2.

The principal difference between the single-source files and the multi-source files is the addition, in the latter, of a source number random parameter and a number of associated tables. Several of these tables are described in the following:

- SU table. This table contains the information specific to a given source (e.g., position)

- NX table. This table contains an index for the file, telling when each source was observed.

- CL tables. These tables contain the information necessary to calibrate the data.

- FG tables. These tables contain the information necessary to flag bad data.

Read access to multi-source files is through the routines UVGET and CALCOP. UVGET selects, reformats, flags and calibrates data as specified and returns one visibility per call after setup. CALCOP will copy all selected records after setup by UVGET. The details of the call sequences of these routines are given at the end of this chapter. These routines handle all of the I/O chores described in this chapter and will also work for single source data files.

Compressed Data

AIPS supports a "Compressed" format for uv data. In this form there is a single "Weight" and scaling random parameter in each visibility record and the real and imaginary parts of the correlation values are packed into a single REAL value with magic value blanking. The details of the packing is machine dependent and is implemented via the "Z" routines ZUVPAK and ZUVXPN. Compressed data can be identified by a dimension of 1 on the "COMPLEX" (first) axis of the data array. The calibration package (UVGET) will automatically unpack visibility data.

Subarrays

Since uv data sets frequently contain data from physically separate arrays, AIPS uv data sets can contain "sub arrays". This is necessary so that the physical identity of each antenna in a visibility record can be uniquely established. Each subarray has its own antenna file, which contains the true frequency and date of observation and the locations and other information about each antenna.

When uv data sets are concatenated, the u, v and w terms of each subsequent data set are converted to wavelengths at the reference frequency defined by the first data set. The subarray number is encoded into the baseline number in each visibility record. The older practice of offsetting times by (subarray-1) $*$ 5 days is being phased out, but still appears in some applications.

Visibility record structure

AIPS uv data is organized in the data file in the same way that similar data is organized on a FITS random groups format tape. Each logical record consists of all data on a given baseline for a given integration period; that is, all polarizations, frequencies, and IFs are contained in a given logical record. The first portion of a logical record is a list of the "random" parameters such as u, v, time, etc. Following the random parameters comes a regular array of data, which is very similar to a small image file. The length and structure of the visibility logical record is fixed in a given data base, but may vary from one data base to another. Records may span disk sector boundaries.

The random parameters can be in any order, but the names of only the first 14 are kept in the catalog header record; this list defines the order in which the values occur. The labels for the normal u, v and w random parameters are "UU-L-SIN", "VV-L-SIN", "WW-L-SIN" indicating that the coordinates correspond to the tangent point of the data computed using sine projection and the units are wavelengths at the reference frequency. The label for the time random parameter is "TIME1" for historical reasons and the label for the baseline parameter is "BASELINE". The label for the source number random parameter is "SOURCE"; the source number points to an entry in the source (SU) table. Other "standard" but optional random parameters are "FQID" for the FQ table identifier and "WEIGHT" and "SCALE" for compressed data.

The regular portion of the array is like an image array in that the order of the axes is arbitrary. In practice for uncompressed data, the first axis should be the COMPLEX axis (real, imaginary, weight). As in image files, the RA, Dec and frequency (for continuum data) are dummy axes which provide a place to store the values for these parameters.

A "regular" axis, which is not intrinsically regular, is what will be referred to as IFs. These are the results of separate receivers (either at RF or IF) which are randomly spaced, but have one or more regularly spaced frequency channels. The pixel number of these IFs points to an entry in the FQ table which gives the frequency offset from the reference frequency for that IF. The FQ table is accessed by the routine CHNDAT, whose call sequence is given at the end of this chapter. The values of the frequency offsets are allowed to be variable inside of a given data set and the set of frequencies and bandwidths used in a given visibility record is specified by an optional FQ identifier random parameter labeled "FQID". One entry is made in the FQ table for each set of frequencies and/or bandwidths.

The structure of a typical VLA data record from a single source file with a single IF is shown in the following figure.

```
| u, v, w, t, b| R1, I1, W1, R2, I2, W2, R3, I3, W3, R4, I4, W4|
  random             RR          LL          RL          LR
    parameters          rectangular data array
```

The symbols in the above are:

- u = u coordinate in wavelengths at the reference frequency
- v = v coordinate
- w = w coordinate
- t = time in days since reference date given in antenna file for this subarray. (The time may be offset by 5 x (subarray no. - 1))
- b = baseline code; 256 x antenna 1 no. + antenna 2 no. + 0.01 x (subarray no. - 1). (see later section for more details)
- Rn = the real part of a correlator value in Jy.
- In = the imaginary part of a correlator value.
- Wn = the weight assigned to the correlator value. In general, it is arbitrary. Data with Wn ≤ 0 are "flagged" (to be ignored).

AIPS uv data sets may contain data in either true Stokes' parameters or correlator based values for circularly polarized IFs. Since Stokes' parameters are not an inherently ordered set, we have adopted the following convention for the values along the Stokes' axis:

```
Stokes' (or correlator) parameter     Value

             I                          1
             Q                          2
             U                          3
             V                          4
            RR                         -1
            LL                         -2
            RL                         -3
            LR                         -4
            XX                         -5
            YY                         -6
            XY                         -7
            YX                         -8
```

The orientations of the "X and "Y" linearly polarized feeds are defined in the antenna (AN) table.

The order of the visibility records in a single source file may be changed; this is usually done with the task UVSRT. Sorting is done using a two key sort and the current sort order is described in the catalog header record (CATBLK(KITYP)) as a two-character HOLLERITH string. The codes currently defined for the sort order are given in the following table, the first key in the sort order varies more slowly.

```
B => baseline number
T => time order
U => u spatial freq. coordinate
V => v spatial freq. coordinate
W => w spatial freq. coordinate
R => baseline length.
P => baseline position angle.
```

```
X => descending ABS(u)
Y => descending ABS(v)
Z => ascending ABS(u)
M => ascending ABS(v)
* => not sorted
```

As examples of the use of the sort order, the older mapping routines require “XY” sorted data (actually they are happy as long as the first key is “X”), self calibration tasks require “TB” order, etc.

Data Order, UVPGET

The position in the record of the standard random parameters (u,v,w,t,b) and the order of the regular axes can be obtained using the routine UVPGET. UVPGET determines pointers and other information from a uv data file catalog header record in common /MAPHDR/ from include DCAT.INC. These pointers are placed in commons which are obtained by the DUVH.INC INCLUDE. The address relative to the start of a vis record for the real part for a given spectral channel (CHAN), IF (NIF) and Stokes parameter (ICOR) is given by:

```
NRPARM + (CHAN-1) * INCF + (NIF-1) INCIF + ABS (ICOR-ICOR0) * INCS
```

Antenna and Subarray Numbers

Antenna and subarray numbers are coded into a single floating word. Some care must be used in decoding these values. The following example shows how to extract these values from a buffer BUFF with UVDISK pointer IBIND and baseline offset ILOCB from DUVH.INC.

```
      INTEGER   IBASE, BIND, ANT1, ANT2, SUBAR
      REAL      BUFF(*), BASE
             ...
C                                     Extract from buffer
      BASE = BUFF(BIND+ILOCB)
C                                     First antenna number
      ANT1 = (BASE / 256.0) + 0.1
C                                     Second antenna number
      ANT2 = (BASE - ANT1 * 256.0) + 0.1
      IBASE = BASE + 0.1
C                                     Subarray number
      SUBAR = (BASE - IBASE) * 100.0 + 1.1
```

Data Reformatting Routines

The variety of different uv data formats, especially different polarization types, allowed in AIPS uv data bases complicates handling of uv data. If a routine is to read and write uv data, it must be prepared to handle any allowed data type. If the routine is only reading the data, reformatting the data to a standard form is practical. There are a number of reformatting routines available.

Efficient reformatting requires two routines, one to setup arrays of pointers and factors and the second to reformat each record. The following list describes several such pairs; detailed descriptions of the call sequence to the routines can be found at the end of this chapter.

- SET1VS, GET1VS return a single visibility value in true Stokes' parameter (I, Q, U, V) or circular polarization (RCP, LCP). They may be requested to work on multiple frequency channels. Does not allow specification of IF at present; defaults to the first.
- SETVIS, GETVIS return several visibility values in the form of true Stokes' parameter (I, Q, U, V) or circular polarization (RCP, LCP). They may be requested to work on multiple frequency channels. A single IF may be specified.

- DGINIT, DGGET are the most general data selection/Stokes' translation routines.
- UVGET sets up, selects, reformats, calibrates, edits either single- or multi-source data files. After set up by UVGET, CALCOP can be used to copy the contents of a file to another file.

6.4.7 UV Data Access

The following is a discussion of the routines to access UV data.

UVGET and CALCOP

Routine UVGET allows relatively easy access to all kinds of AIPS interferometer uv data from both single- and multi-source files, in either normal or compressed format and can optionally select, calibrate, edit and convert the stokes parameter of the data selected. After an initialization call UVGET returns one visibility at a time. UVGET can apply SN, BL or BP calibration tables and/or make polarization corrections as specified in the AN table to single source files. Most of the communication with UVGET is through the commons in INCLUDE file DSEL.INC which are described in the description of UVGET at the end of this chapter. These values may be initialized using routine SELINI whose description appears at the end of this chapter.

If it is more convienent to operate on a uv data scratch file than on one visibility at a time (e.g. multiple passes throught the data are required) then CALCOP can be used to produce a file containing the selected data with any calibration etc. operations done on them. CALCOP will optionally create the scratch file. A description of CALCOP is given at the end of this chapter.

UVINIT and UVDISK

UV data files may be located and opened using routine MAPOPN.Data are read or written using UVINIT and UVDISK in much the same manner in which image files are read with MINIT and MDISK. One significant difference between UVDISK and MDISK is that UVDISK can be requested to process multiple logical records (NPIO) in a single call. If NPIO is 0, then the largest value consistent with double buffering will be used; if NPIO is too large for the buffer provided, it will be reduced to the largest value consistent with single buffering. This is useful when large amounts of data are to be sent to a sorting routine or to the array processor or to reduce the overhead of many subroutine calls.

Another difference between MINIT and UVINIT is that, unlike MINIT, UVINIT returns the buffer pointer for the first call so the output buffer can be written into *before* the first call to UVDISK.

UVINIT sets up the bookkeeping for UVDISK which does double buffered (if possible) quick-return I/O. UVDISK will run much more efficiently if, on disk, the requested transfer (logical record length x the number of records per call) is an integral number of disk blocks. Otherwise, partial writes or oversize reads will have to be done. Minimum disk I/O is one block.

The buffer size for UVDISK should include an extra NBPS bytes for each buffer for reads, if NPIO records does not correspond to an integral number of disk sectors (NBPS bytes). 2*NBPS extra bytes required for each (single or double) buffer for writes. More details about the call sequence to UVINIT and the use of the FTAB are given at the end of this chapter.

UVDISK reads and writes records of arbitrary length, especially uv visibility data. There are three operations which can be invoked: READ, WRITE and FLUSH (OPcodes "READ", "WRIT" and "FLSH").

If the requested transfers are too large to double buffer with the given buffer size, then UVDISK will single buffer the I/O. If it is possible to do double buffered physical transfers of some multiple of the requested number of records, then this is done.

OPcode = "READ" reads the next sequential block of data as specified to UVINIT and returns the actual number of visibilities, NIO, and the pointer, BIND, to the first word of this data in the buffer.

OPcode="WRIT" collects data in a buffer half until it is full. Then, as many full blocks as possible are written to the disk with the remainder left for the next disk write. For writes, left-over data is transferred to the beginning of the next buffer half to be filled. The value of NIO in the call is the number of visibility records to be added to the buffer and may be fewer than the number specified to UVINIT. On return, NIO is the maximum number which may be sent next time. On return, BIND is the pointer in BUFFER to begin filling new data.

OPcode="FLSH" writes integral numbers of blocks and moves any data left over to the beginning of buffer 1. One exception to this is when NIO ≤ 0, in which case the entire remaining data in the buffer is written (if NIO < 0 then ABS (NIO) visibilities are to be written). After the call, BIND is the pointer in BUFFER for new data. The principal difference between FLSH and WRIT is that FLSH always forces an I/O transfer. This may cause trouble if a transfer of less than 1 block is requested. A call with a nonpositive value of NIO should be the last call and corresponds to a call to MDISK with opcode "FINI".

The input/output argument to UVDISK, NIO, can be very useful for controlling the loop reading and/or writing uv data. The value of NIO for reads is the number of values in the buffer that are available. When the file has been completely read, the value of NIO returned by UVDISK on the next call is 0; this value can be used to determine when all of the data has been read. The example in the following section uses this feature in UVDISK. More details about the call sequence can be found at the end of this chapter.

Example using UVINIT and UVDISK

```
      SUBROUTINE UVCONJ (ISCR1, ISCR2, LUN1, LUN2, BUFF1, BUFF2,
     *   BUFSZ1, BUFSZ2, IERR)
C-----------------------------------------------------------------------
C   UVCONJ takes the complex conjugate of the values in a uv data set
C   in a scratch file in the /CFILES/ common (INCLUDE DFIL.INC) number
C   ISCR1 and writes them in the /CFILES/ scratch  file number ISCR2.
C   The current values in the /UVHDR/ commons (INCLUDE DUVH.INC) are
C   assumed to describe the uv data files.
C   Inputs:
C      ISCR1   I     /CFILES/ scratch file number of input file
C      ISCR2   I     /CFILES/ scratch file number of output file
C      LUN1    I     Logical unit number to use for file 1
C      LUN2    I     Logical unit number to use for file 2
C      BUFF1   R(*)  I/O buffer to use for file 1
C      BUFF2   R(*)  I/O buffer to use for file 2
C      BUFSZ1  I     Size of BUFF1 in AIPS bytes (2*no. words)
C      BUFSZ2  I     Size of BUFF2 in AIPS bytes
C   Inputs from common /UVHDR/ (DUVH.INC)
C      NVIS    I     Number of visibility records
C      LREC    I     logical record length.
C      NRPARM  I     number of random parameters.
C      ICOR0   I     (signed) value of first Stokes' parameter.
C      JLOCF   I     zero relative order of the frequency axis in
C                    the data array.
C      JLOCS   I     relative order of the Stokes' axis.
C      JLOCIF  I     relative order of the IF axis.
C      INCF    I     word increment in the data array between
C                    successive frequencies at the same location on
C                    all other axes.
C      INCS    I     word increment in the data array between
C                    successive Stokes' values.
C      INCS    I     word increment in the data array between
C                    successive IF values.
C   Inputs from common /MAPHDR/ (DCAT.INC)
C      CATBLK  I(256) Catalog header record
C   Output:
C      IERR    I     Return code, 0=>OK, otherwise error.
C-----------------------------------------------------------------------
```

```
      INTEGER   ISCR1, ISCR2, LUN1, LUN2, BUFSZ1, BUFSZ2, IERR
      REAL      BUFF1(*), BUFF2(*)
C
      INTEGER   FIND1, FIND2, BIND1, BIND2, NFREQ, NSTOKE, NIF, I, IV,
     *   IFQ, IST, IIF, NIOIN, NIOUT, INDEX, JCOR0, NILIM, BO, VO
      LOGICAL   T, F
      CHARACTER FILE*48
      PARAMETER (T = .TRUE.)
      PARAMETER (F = .FALSE.)
C                                       Listings of the standard
C                                       INCLUDE files are at the end
C                                       of the chapter on tasks.
      INCLUDE 'INCS:DMSG.INC'
      INCLUDE 'INCS:DUVH.INC'
      INCLUDE 'INCS:DHDR.INC'
      INCLUDE 'INCS:DCAT.INC'
      DATA VO, BO /0,1/
C-----------------------------------------------------------------------
C                                       Take absolute value of first
C                                       Stokes' value.
      JCOR0 = ABS (ICOR0)
C                                       Find dimension of freq
C                                       and Stokes axes.
      NFREQ = CATBLK(KINAX+JLOCF)
      NSTOKE = CATBLK(KINAX+JLOCS)
C                                       May not have IF axis
      NIF = 1
      IF (JLOCIF.GT.0) NIF = CATBLK(KINAX+JLOCIF)
C                                        Open and init ISCR1
      CALL ZPHFIL ('SC', SCRVOL(ISCR1), SCRCNO(ISCR1), 1, FILE, IERR)
      CALL ZOPEN (LUN1, FIND1, SCRVOL(ISCR1), FILE, T, F, T, IERR)
C                                        Check for error
      IF (IERR.NE.0) THEN
         WRITE (MSGTXT,1000) IERR, 'READ', 1
         GO TO 990
         END IF
C                                       Let UVINIT determine #/call
      NIOIN = 0
      CALL UVINIT ('READ', LUN1, FIND1, NVIS, VO, LREC, NIOIN,
     *   BUFSZ1, BUFF1, BO, BIND1, IERR)
C                                        Check for error
      IF (IERR.NE.0) THEN
         WRITE (MSGTXT,1010) IERR, 'READ', 1
         GO TO 990
         END IF
C                                        Open and init ISCR2
      CALL ZPHFIL ('SC', SCRVOL(ISCR2), SCRCNO(ISCR2), 1, FILE, IERR)
      CALL ZOPEN (LUN2, FIND2, SCRVOL(ISCR2), FILE, T, F, T, IERR)
C                                        Check for error
      IF (IERR.NE.0) THEN
         WRITE (MSGTXT,1000) IERR, 'WRIT', 2
         GO TO 990
         END IF
      NIOUT = 0
```

```
      CALL UVINIT ('WRIT', LUN2, FIND2, NVIS, VO, LREC, NIOUT,
     *   BUFSZ2, BUFF2, BO, BIND2, IERR)
      NILIM = NIOUT
      NIOUT = 0
C                                         Check for error
      IF (IERR.NE.0) THEN
         WRITE (MSGTXT,1010) IERR, 'WRIT', 2
         GO TO 990
         END IF
C                                        Loop through data file.
C                                        Read input file
         CALL UVDISK ('READ', LUN1, FIND1, BUFF1, NIOIN, BIND1, IERR)
C                                         Check for error
         IF (IERR.NE.0) THEN
            WRITE (MSGTXT,1060) IERR, 'READ', 1
            GO TO 990
            END IF
C                                        Check if data all read.
         IF (NIOIN.LE.0) GO TO 120
C                                        Loop through records
         DO 100 IV = 1,NIOIN
C                                        Loop through IF
            DO 90 IIF = 1,NIF
C                                        Loop through Stokes' axis
               DO 80 IST = 1,NSTOKE
C                                        Loop through frequency axis
                  DO 80 IFQ = 1,NFREQ
C                                        Compute pointer in the
C                                        buffer to imag. part
                     INDEX = NRPARM + (IFQ-1) * INCF + (IIF-1) * INCIF
     *                  + (IST-JCOR0) * INCS + 1 + (BIND1 - 1)
C                                        Conjugate visibility
                     BUFF1(INDEX) = - BUFF1(INDEX)
 80                  CONTINUE
 90            CONTINUE
C                                        Copy record to output buffer
            CALL RCOPY (LREC, BUFF1(BIND1), BUFF2(BIND2))
C                                        Update buffer pointers
            BIND1 = BIND1 + LREC
            BIND2 = BIND2 + LREC
            NIOUT = NIOUT + 1
C                                         Write output when necessary
            IF (NIOUT.GE.NILIM) THEN
               CALL UVDISK ('WRIT', LUN2, FIND2, BUFF2, BIND2,
     *            NIOUT, IERR)
               NILIM = NIOUT
               NIOUT = 0
C                                         Check for error
               IF (IERR.NE.0) THEN
                  WRITE (MSGTXT,1060) IERR, 'WRIT', 2
                  GO TO 990
                  END IF
               END IF
100         CONTINUE
```

```
C                                        Loop back for more data
 110     GO TO 60
C                                        Finished, flush buffer.
C                                        No more output records.
 120  NIOUT = -NIOUT
      CALL UVDISK ('FLSH', LUN2, FIND2, BUFF2, BIND2, NIOUT, IERR)
C                                         Check for error
      IF (IERR.NE.0) THEN
         WRITE (MSGTXT,1060) IERR, 'FLSH', 2
         GO TO 990
         END IF
C                                         Close files.
 130  CALL ZCLOSE (LUN1, FIND1, IERR)
C                                         Check for error
      IF (IERR.NE.0) THEN
         WRITE (MSGTXT,1060) IERR, 'CLOS', 1
         GO TO 990
         END IF
      CALL ZCLOSE (LUN2, FIND2, IERR)
C                                         Check for error
      IF (IERR.NE.0) THEN
         WRITE (MSGTXT,1060) IERR, 'CLOS', 2
         GO TO 990
         END IF
      IERR = 0
      GO TO 999
C                                         Error.
 990  CALL MSGWRT (8)
 999  RETURN
C-----------------------------------------------------------------------
 1000 FORMAT ('UVCONJ: ERROR',I3,' OPEN FOR ',A4,' FILE',I2)
 1010 FORMAT ('UVCONJ: ERROR',I3,' INIT FOR ',A4,' FILE',I2)
 1060 FORMAT ('UVCONJ: ERROR',I3,1X,A4,'ING FILE',I2)
      END
```

6.4.8 Single Dish Data

AIPS has a limited capacity to handle single dish data; that is measurments of sky brightness at random locations on the sky. The format of single dish data is much like that of interferometer data so that many of the utility routines will work for both. The "COMPLEX" axis still has 3 values (except for the compressed form) which are: the measured sky brightness, any baseline or other offset removed and a weight. In place of the U and V random parameters are sky positions which may be either absolute or in a specified tangent plane projection. The structure of Single dish data is described in the following.

Single dish random parameters

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

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

2. 'TIME1': The time tags for the data are kept in days since the reference day.
3. 'BEAM': This random parameter gives the beam number + 256. The beam offset makes the data look more like uv data and more of the the AIPS uv data tasks will work for this data.
4. 'SCAN': This random parameter gives the scan number. This random parameter is optional.
5. 'SAMPLE': This random parameter gives the sample number in the scan. This random parameter is optional.

Single dish regular axis coordinates

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

1. 'COMPLEX': the "complex" axis consists of the measured brightness, subtracted baseline, and (optional) weight. Magic value blanking is supported. This axis is required.
2. 'STOKES': this axis is used to describe which Stokes' parameters are given; the conventions are the same as used for uv data. This axis is required.
3. 'FREQ': the frequency axis coordinates are in Hz. This axis is required.
4. 'IF': The IF axis is a construct which allows irregularly spaced groups of frequency channels. The IF number specifies an entry in the FQ table which must be present if this axis is present. This axis is optional.
5. 'RA' and 'DEC': the celestial coordinates are given in degrees. The values associated with these axes are irrelevant (although they should be present) for unprojected data. For data with projected coordinates the coordinate values of these axes should be the tangent point, i.e. the position on the sky at which the plane onto which the coordinates are projected is tangent to the celestial sphere and these axes should become 'RA−−−ccc' and 'DEC−−ccc' where ccc is the projection code. These axes are required.

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

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

UVPGET and Single Dish Data

The routine UVPGET that interpretes uv data catalog headers also understands single dish data. The values passed in INCLUDE DUVH.INC which differ for single dish and interferometer data are described in the following:

- TYPUVD. This integer has a value of 1 for unprojected and 2 for projected sky coordinates. (0 indicates interferometer data).
- ILOCU. This integer gives the 0-rel offset from the beginning of the record of the longitude like celestial coordinate random parameter.
- ILOCV. This integer gives the 0-rel offset from the beginning of the record of the latitude like celestial coordinate random parameter.
- ILOCB. This integer gives the 0-rel offset from the beginning of the record of the beam number random parameter if present; -1 if absent.

Access to and Calibration of Single Dish Data

Single dish data may be read with UVINIT/UVDISK in the same manner as interferometer data. There is a calibration system for single dish data that parallels the interferometer system. Access to this system is through the routine SDGET which is the single dish analog to UVGET for interferometer data with most of the control parameters being passsed through the commons in DSEL.INC. SDGET can optionally apply calibration information from the CS table and flagging from the FG table. A detailed description of SDGET is given at the end of this chapter. A more complete description of the calibration system is given in Volume 2.

6.4.9 Extension files

Extension files contain a great variety of different types of data, but usually are small compared to the data files. Thus, for extension file I/O, the routines are friendlier, but less efficient. In many cases, the data stored in extension files consist of logical records which contain different data types and are, in fact, data structures.

One type of extension file is the table. This type of file contains a self-describing header and is useful for most types of data which can be forced into a tabular structure. The principal advantage of tables is that generalized table manipulating routines, including writing to, and reading from, FITS files automatically are available.

TABINI and TABIO

The routines TABINI and TABIO do I/O to extension tables. A single call to TABINI will create an extension table if necessary, catalog it, open the file, and initialize the I/O. TABIO then allows random access, with mixed reads and write allowed, to the extension file. TABINI returns a set of pointers which can be used to access data in a record. In practice, another level of specific routines for each table type is useful to access tables. Use of tables in AIPS is dealt with in more detail in another chapter in this manual.

EXTINI and EXTIO

NOTE: TABINI and TABIO are strongly preferred over EXTINI and EXTIO as EXTINI/EXTIO files are not copied to FITS format files.

The routines EXTINI and EXTIO make I/O to extension files much simpler than the image and uv data routines. A single call to EXTINI will create an extension file if necessary, catalog it, open the file, and initialize the I/O. EXTIO then allows random access, with mixed reads and write allowed, to the extension file. EXTIO copies the data into a specified array so that a data structure can be formed by means of a Fortran equivalence, either an explicit EQUIVALENCE statement or through the use of a common.

The structure of the extension file is a header record of 512 bytes, some of which are used by EXTINI and EXTIO for bookkeeping, but many of which are available for use. Following the header record come the fixed length logical records which are physically blocked in 512 byte blocks. A single logical record may use several physical blocks or several logical records may be in a given 512 byte block. Details of the call sequences for EXTINI and EXTIO and a description of the file header record are given at the end of this chapter.

Simple copies of any and/or all EXTINI-EXTIO files of a given type may be copied with a single call to EXTCOP. A description of the call sequence for EXTCOP is given at the end of the chapter on tasks.

6.4.10 Text files

AIPS uses a number of text files, such as the HELP and RUN files; in addition, there is the capability to read and write arbitrary text files. There are several routines which allow access to text files: ZTXOPN, ZTXIO, ZTXCLS, ZTOPEN, ZTREAD, ZTCLOS, and KEYIN.

- ZTXOPN opens a text file for read or write.

- ZTXIO reads/writes a line from/to a text file opened by ZTXOPN.
- ZTXCLS closes a text file opened via ZTXOPN.
- ZTOPEN opens a text file. It is similar to ZOPEN except that it has an additional input argument (MNAME) which gives the name of the desired file or member.
- ZTREAD returns one 80-character line of text from a file opened by ZTOPEN.
- ZTCLOS closes the text file.
- KEYIN is the AIPS version of the Cal Tech VLBI parsing routine. This a very flexible routine for obtaining values from external text files.

ZTREAD has a number of standard places that it can find text files. These include the RUN file area, the HELP file area, and various source code areas. To access files in the "RUN" area, a file name (PNAME) should be constructed with ZPHFIL with type "RU"; other inputs are dummy. ZTOPEN should then be called with LUN=10 and this value of PNAME.

Arbitrary text files can be read or written using ZTXIO which needs ZTXOPN to open a file and ZTXCLS to close it. Details of the call sequences are given at the end of this chapter and an example of their use follows. In this example a text file whose name is in the CHARACTER variable FILNAM is read. This file contains lines of text no longer than 80 characters.

```
      INTEGER   LUN, FIND, IERR
      LOGICAL   F
      CHARACTER FILNAM*48, LINE*80
      PARAMETER (F = .FALSE.)
      PARAMETER (LUN = 10)
      INCLUDE 'INCS:DMSG.INC'
                     .
                     .
C                                      Open file
      CALL ZTXOPN ('READ', LUN, FIND, FILNAM, F, IERR)
C                                      Check error
      IF (IERR.NE.0) THEN
         WRITE (MSGTXT,1000) IERR, 'OPENING'
         GO TO 990
         END IF
C                                      Loop over file.
C                                      Read next line
 100     CALL ZTXIO ('READ', LUN, FIND, LINE, IERR)
C                                      EOF
         IF (IERR.EQ.2) GO TO 700
C                                      Check error
         IF (IERR.NE.0) THEN
            WRITE (MSGTXT,1000) IERR, 'READING'
            GO TO 990
            END IF
C                                      Process LINE
                     .
                     .
C                                      Loop for next LINE
         GO TO 100
C                                      Close file
 700  CALL ZTXCLS (LUN, FIND, IERR)
C                                      Check error
```

```
      IF (IERR.NE.0) THEN
         WRITE (MSGTXT,1000) IERR, 'CLOSING'
         GO TO 990
         END IF
      GO TO 999
C                                          Error
 990  CALL MSGWRT (8)
C
 999  RETURN
C-----------------------------------------------------------------------
 1000 FORMAT ('ERROR ',I3,1X,A,' TEXT FILE')
      END
```

In the example above, calls to KEYIN could have replaced the calls to ZTXIO.

The file name passed to ZTXOPN should contain a logical pointing to the directory containing the file. In VMS this may be a complete specification of the directory but in Unix an environment must be set outside of AIPS.

Examples:

```
FILNAM='DISK$RES:[USERNAME]CAL.DAT'     (VMS)
FILNAM='MYAREA:CAL.DAT'                 (Unix)
   where MYAREA is an environment variable set before
   starting AIPS:
   %setenv MYAREA /mnt/username
```

6.5 Bottom Level I/O Routines

The routines described so far in this chapter have been relatively high level routines which have hidden a great deal of bookkeeping. In addition, the image and uv data I/O routines work basically sequentially with some data selection ability. Beneath the higher level routines there are, of course, lower level routines. These routines have a great deal more flexibility than the higher level routines, but usually at a cost of a great deal of bookkeeping.

The basic AIPS I/O routines are intrinsically random access, although a data transfer must start on a disk block boundary. "Map" type files (image and uv data) are read with a pair of routines ZMIO and ZWAIT. Non-map (extension) files are read with ZFIO.

6.5.1 ZMIO and ZWAIT

ZMIO initiates a data transfer to or from one of two possible buffer halves and returns without waiting for the operation to complete. ZWAIT is a timing routine which suspends the task until the specified I/O operation is complete. In this manner, I/O and computation can be overlapped.

The I/O commons (INCLUDE DDCH.INC) contain an array, FTAB, which contains AIPS and host system I/O tables. ZOPEN returns a pointer in FTAB to the area to use for a given file. The first 16 integer words of this area are available for AIPS program use, the remainder of an FTAB entry is used for the host system I/O tables. These 16 words are normally used for bookkeeping information (the first always contains the value of the LUN). Examples of the use of the FTAB are found in MINIT, MDISK, MINSK, MSKIP, UVINIT and UVDISK which use ZMIO and ZWAIT. Descriptions of the way these routines use the FTAB are to be found at the end of this chapter. A description of the call arguments to ZMIO and ZWAIT are also found at the end of this chapter.

6.5.2 ZFIO

Extension file I/O and single buffer non-disk I/O is usually done with the routine ZFIO. For disk files, ZFIO reads a 512 byte block from a specified offset in the file. This block size is independent of the true physical block size on the disks being used. The I/O transfer is complete when ZFIO returns.

Details of the call sequence for ZFIO are found at the end of this chapter. An example of the use of ZFIO may be found in the source code for TABINI and TABIO.

6.6 Text of INCLUDE files

6.6.1 DDCH.INC

```
C                                                            Include DDCH.
C                                        AIPS system parameters
      CHARACTER SYSNAM*20, VERNAM*4, RLSNAM*8, DEVNAM(10)*48,
     *   NONNAM(8)*48, MAPNAM(12)*48, SYSTYP*4, SYSVER*8
      HOLLERITH HBLANK
      DOUBLE PRECISION DBLANK
      REAL   XPRDMM, XTKDMM, TIMEDA(15), TIMESG, TIMEMS, TIMESC, TIMECA,
     *   TIMEBA(4), TIMEAP(3), FBLANK, RFILIT(14)
      INTEGER   NVOL, NBPS, NSPG, NBTB1, NTAB1, NBTB2, NTAB2, NBTB3,
     *   NTAB3, NTAPED, CRTMAX, PRTMAX, NBATQS, MAXXPR(2), CSIZPR(2),
     *   NINTRN, KAPWRD, NWDPDP, NBITWD, NCHLIN, NTVDEV, NTKDEV, BLANKV,
     *   NTVACC, NTKACC, UCTSIZ, BYTFLP, USELIM, NBITCH, NCHPRT,
     *   KAP2WD, MAXXTK(2), CSIZTK(2), DASSGN(8,15), SPFRMT, DPFRMT,
     *   NSHORT, TTYCAR, DEVTAB(50), FTAB(1024)
      COMMON /DCHCHM/ SYSNAM, VERNAM, SYSTYP, SYSVER, RLSNAM,
     *   DEVNAM, NONNAM, MAPNAM
      COMMON /DCHCOM/ DBLANK, XPRDMM, XTKDMM, TIMEDA, TIMESG, TIMEMS,
     *   TIMESC, TIMECA, TIMEBA, TIMEAP, FBLANK, RFILIT, HBLANK,
     *   NVOL, NBPS, NSPG, NBTB1, NTAB1, NBTB2, NTAB2, NBTB3, NTAB3,
     *   NTAPED, CRTMAX, PRTMAX, NBATQS, MAXXPR, CSIZPR, NINTRN,
     *   KAPWRD, NWDPDP, NBITWD, NCHLIN, NTVDEV, NTKDEV, BLANKV,
     *   NTVACC, NTKACC, UCTSIZ, BYTFLP, USELIM, NBITCH, NCHPRT,
     *   KAP2WD, MAXXTK, CSIZTK, DASSGN, DEVTAB, SPFRMT, DPFRMT,
     *   NSHORT, TTYCAR
      COMMON /FTABCM/ FTAB
C                                                            End DDCH.
```

6.6.2 DSEL.INC

```
C                                                            Include DSEL.
C                                        Commons for UVGET use
      INTEGER   XCTBSZ, XBTBSZ, XPTBSZ, XSTBSZ, XTTSZ, XBPSZ,
     *   XBPBUF
C                                        XCTBSZ=internal gain table size
      PARAMETER (XCTBSZ=2500)
C                                        XBTBSZ=baseline table size
      PARAMETER (XBTBSZ=3500)
C                                        XPTBSZ=polar. corr. table size
      PARAMETER (XPTBSZ=16384)
C                                        XSTBSZ=Source no. table size
      PARAMETER (XSTBSZ=500)
C                                        XTTSZ=Pol. trans. table size
```

```
      PARAMETER (XTTSZ=MAXIF*MAXCHA*2)
C                                       XBPSZ=max. no. BP time entries
      PARAMETER (XBPSZ=50)
C                                       XBPBUF=internal BP I/O buffer
      PARAMETER (XBPBUF=65536)
C                                       Data selection and control
      INTEGER   ANTENS(50), NANTSL, NSOUWD, SOUWAN(XSTBSZ), SOUWTN(30),
     *   NCALWD, CALWAN(XSTBSZ), CALWTN(30), SUBARR, SMOTYP, CURSOU,
     *   NXKOLS(MAXNXC), NXNUMV(MAXNXC), MVIS, JADR(2,XTTSZ), PMODE,
     *   LRECIN, UBUFSZ, BCHAN, ECHAN, BIF, EIF, NPRMIN, KLOCSU, KLOCFQ,
     *   SELQUA, SMDIV, SMOOTH(3), KLOCIF, KLOCFY, KLOCWT, KLOCSC,
     *   NDECMP, DECMP(2,MAXIF*4), BCHANS, ECHANS, FRQSEL, FSTRED,
     *   FQKOLS(MAXFQC), FQNUMV(MAXFQC)
      LOGICAL   DOSWNT, DOCWNT, DOAWNT, ALLWT, TRANSL, DOSMTH, ISCMP,
     *   DOXCOR, DOACOR, DOWTCL, DOFQSL
      INTEGER   INXRNO, NINDEX, FSTVIS, LSTVIS, IFQRNO
      REAL      TIMRNG(8),  UVRNG(2), INTPRM(3), UVRA(2), TSTART, TEND,
     *   SELFAC(2,XTTSZ), SMTAB(2500), SUPRAD, SELBAN
      CHARACTER SOURCS(30)*16, CALSOU(30)*16, STOKES*4, INTFN*4,
     *   SELCOD*4
      DOUBLE PRECISION UVFREQ, SELFRQ
C                                       Flag table info
      REAL      TMFLST, FLGTND(MAXFLG)
      INTEGER   IFGRNO
      LOGICAL   DOFLAG, FLGPOL(4,MAXFLG)
      INTEGER   FGVER, NUMFLG, FGKOLS(MAXFGC), FGNUMV(MAXFGC),
     *   KNCOR, KNCF, KNCIF, KNCS,
     *   FLGSOU(MAXFLG), FLGANT(MAXFLG), FLGBAS(MAXFLG), FLGSUB(MAXFLG),
     *   FLGBIF(MAXFLG), FLGEIF(MAXFLG), FLGBCH(MAXFLG), FLGECH(MAXFLG)
C                                       CAL table info
      REAL      GMMOD, CURCAL(XCTBSZ), LCALTM, CALTAB(XCTBSZ,2),
     *   CALTIM(3), RATFAC(MAXIF), DELFAC(MAXIF), DXTIME, DXFREQ,
     *   LAMSQ(MAXCHA, MAXIF), IFRTAB(MAXANT, 2), IFR(MAXANT)
      INTEGER   ICLRNO, NCLINR, MAXCLR, CNTREC(2,3)
      LOGICAL   DOCAL, DOAPPL
      INTEGER   CLVER, CLUSE, NUMANT, NUMPOL, NUMIF, CIDSOU(2),
     *   CLKOLS(MAXCLC), CLNUMV(MAXCLC), LCLTAB, LCUCAL, ICALP1, ICALP2,
     *   POLOFF(4,2)
C                                       Baseline table info
      REAL      LBLTM, BLTAB(XBTBSZ,2), BLFAC(XBTBSZ), BLTIM(3)
      INTEGER   IBLRNO, NBLINR
      LOGICAL   DOBL
      INTEGER   BLVER, BLKOLS(MAXBLC), BLNUMV(MAXBLC), IBLP1, IBLP2
C                                       Polarization table.
      REAL      POLCAL(2,XPTBSZ), PARAGL(2,MAXANT), PARTIM
      INTEGER   PARSOU
      LOGICAL   DOPOL
C                                       Bandpass table
      DOUBLE PRECISION BPFREQ(MAXIF)
      REAL      PBUFF(XBPBUF), TIMENT(XBPSZ), BPTIM(3), LBPTIM, CHNBND
      CHARACTER BPNAME*48
      INTEGER   IBPRNO, NBPINR, ANTPNT(2), NVISM, NVISS, NVIST
      INTEGER   BPVER, BPKOLS(MAXBPC), BPNUMV(MAXBPC), NANTBP, NPOLBP,
     *   NIFBP, NCHNBP, BCHNBP, DOBAND, ANTENT(XBPSZ,MAXANT),
```

```
     *   BPDSK, BPVOL, BPCNO, USEDAN(MAXANT), BPGOT(2),
     *   KSNCF, KSNCIF, KSNCS, MXANUM
C                                      Channel 0 stuff
      INTEGER   FSTVS3, LREC3, LSTVS3, NREAD3, FSTRD3, KLOCW3,
     *   KLOCS3, NDECM3, DECM3(2,MAXIF*4), BIND3, RECNO3, LENBU3
      LOGICAL   ISCMP3, DOUVIN
C                                      File specification.
      INTEGER   IUDISK, IUSEQ, IUCNO, IULUN, IUFIND, ICLUN, IFLUN,
     *   IXLUN, IBLUN, IPLUN, IQLUN, LUNSBP, BPFIND, CATUV(256),
     *   CATBLK(256)
      REAL      USEQ, UDISK
      CHARACTER UNAME*12, UCLAS*6, UFILE*48
C                                      I/O buffers
      INTEGER   CLBUFF(1024), FGBUFF(512), NXBUFF(512), BLBUFF(512),
     *   BPBUFF(32767), FQBUFF(512)
      REAL      UBUFF(8192)
C                                      Character common
      COMMON /SELCHR/ SOURCS, CALSOU, STOKES, INTFN, SELCOD, UNAME,
     *   UCLAS, UFILE, BPNAME
C                                      Common for UVGET use
C                                      Data selection and control
      COMMON /SELCAL/ UVFREQ, SELFRQ,
     *   USEQ, UDISK, TIMRNG, UVRNG, INTPRM, UVRA, TSTART, TEND, UBUFF,
     *   SELFAC, SMTAB, SUPRAD, SELBAN,
     *   INXRNO, NINDEX, FSTVIS, LSTVIS, IFQRNO,
     *   DOSWNT, DOCWNT, DOAWNT, ALLWT, TRANSL, DOSMTH, ISCMP, DOXCOR,
     *   DOACOR, DOWTCL, DOFQSL,
     *   CLBUFF, FGBUFF, NXBUFF, BLBUFF, BPBUFF, FQBUFF,
     *   IUDISK, IUSEQ, IUCNO, IULUN, IUFIND, ICLUN, IFLUN, IXLUN,
     *   IBLUN, IPLUN, IQLUN, LUNSBP, BPFIND, CATUV, ANTENS, NANTSL,
     *   NSOUWD, SOUWAN, SOUWTN, NCALWD, CALWAN, CALWTN,
     *   SUBARR, SMOTYP, CURSOU, NXKOLS, NXNUMV, FQKOLS, FQNUMV,
     *   MVIS, JADR, PMODE,
     *   LRECIN, UBUFSZ, BCHAN, ECHAN, BIF, EIF, NPRMIN, KLOCSU,
     *   KLOCFQ, SELQUA, SMDIV, SMOOTH, KLOCIF, KLOCFY, KLOCWT,
     *   KLOCSC, NDECMP, DECMP, BCHANS, ECHANS, FRQSEL, FSTRED
C                                      FLAG table info
      COMMON /CFMINF/ TMFLST, FLGTND, IFGRNO, DOFLAG, FLGPOL,
     *   FGVER, NUMFLG, FGKOLS, FGNUMV, KNCOR, KNCF, KNCIF, KNCS,
     *   FLGSOU, FLGANT, FLGBAS, FLGSUB, FLGBIF, FLGEIF, FLGBCH, FLGECH
C                                      CAL table info
      COMMON /CGNINF/ GMMOD, CURCAL, LCALTM, CALTAB, CALTIM, RATFAC,
     *   DELFAC, DXTIME, DXFREQ,
     *   ICLRNO, NCLINR, MAXCLR, CNTREC,
     *   DOCAL, DOAPPL,
     *   CLVER, CLUSE, NUMANT, NUMPOL, NUMIF, CIDSOU, CLKOLS, CLNUMV,
     *   LCLTAB, LCUCAL, ICALP1, ICALP2, POLOFF,
     *   LAMSQ, IFRTAB, IFR
C                                      BL table info
      COMMON /CBLINF/ LBLTM, BLTAB, BLTIM, BLFAC,
     *   IBLRNO, NBLINR,
     *   DOBL,
     *   BLVER, BLKOLS, BLNUMV, IBLP1, IBLP2
C                                      Pol. table
```

```
      COMMON /CPLINF/ POLCAL, PARAGL, PARTIM, PARSOU, DOPOL
C                                        BP table
      COMMON /CBPINF/ BPFREQ,
     *   PBUFF, TIMENT, BPTIM, LBPTIM, CHNBND,
     *   IBPRNO, NBPINR, ANTPNT, NVISM, NVISS, NVIST,
     *   BPVER, BPKOLS, BPNUMV, NANTBP, NPOLBP, NIFBP, NCHNBP, BCHNBP,
     *   DOBAND, ANTENT, BPDSK, BPVOL, BPCNO, USEDAN, BPGOT,
     *   KSNCF, KSNCIF, KSNCS, MXANUM
C                                        Channel 0 common
      COMMON /CHNZ/ FSTVS3, LREC3, LSTVS3, NREAD3, FSTRD3, KLOCW3,
     *   KLOCS3, NDECM3, DECM3, BIND3, RECNO3, LENBU3,
     *   ISCMP3, DOUVIN
C
      COMMON /MAPHDR/ CATBLK
C                                                          End DSEL.
```

6.6.3 DUVH.INC

```
C                                                          Include DUVH.
C                                        If you change this include you
C                                        must also change common
C                                        /CATHDR/ in DBCON
C                                        Include for uv header info
      INTEGER   NVIS
      INTEGER   ILOCU, ILOCV, ILOCW, ILOCT, ILOCB, ILOCSU, ILOCFQ,
     *   JLOCC, JLOCS, JLOCF, JLOCR, JLOCD, JLOCIF, NRPARM, LREC,
     *   NCOR, INCS, INCF, INCIF, ICOR0, TYPUVD
      CHARACTER   SOURCE*8, ISORT*2
      DOUBLE PRECISION FREQ, RA, DEC
      COMMON /UVHDR/ FREQ, RA, DEC, NVIS, ILOCU, ILOCV, ILOCW, ILOCT,
     *   ILOCB, ILOCSU, ILOCFQ, JLOCC, JLOCS, JLOCF, JLOCR, JLOCD,
     *   JLOCIF, INCS, INCF, INCIF, ICOR0, NRPARM, LREC, NCOR, TYPUVD
      COMMON /UVHCHR/ SOURCE, ISORT
C                                                          End DUVH.
```

6.7 Routines

6.7.1 CALCOP

Routine to copy selected data from one data file to another optionally applying calibration and editing information. The input file should have been opened with UVGET. Both files will be closed on return from CALCOP. Note: UVGET returns the information necessary to catalog the output file. The output file will be compressed if necessary at completion of CALCOP.

```
CALCOP (DISK, CNOSCR, BUFFER, BUFSZ, IRET)
Inputs:
   DISK     I       Disk number for catalogd output file.
                    If .LE. 0 then the output file is a /CFILES/
                    scratch file.
   BUFFER   R(*)    Work buffer for writing.
   BUFSZ    I       Size of BUFFER in bytes.
Input via common:
   LREC     I       (/UVHDR/) length of vis. record in R words.
   NRPARM   I       (/UVHDR/) number of R random parameters.
```

```
In/out:
   CNOSCR   I       Catalog slot number for if cataloged file;
                    /CFILES/ scratch file number if a scratch file,
                    IF DISK=CNOSCR=0 then the scratch is created.
                    On output = Scratch file number if created.
In/out via common:
   CATBLK   I(256)  Catalog header block from UVGET
                    on output with actual no. records
   NVIS     I       (/UVHDR/) Number of vis. records.
Output:
   IRET     I       Error code: 0 => OK,
                       > 0 => failed, abort process.
Usage notes:
(1) UVGET with OPCODE='INIT' MUST be called before CALCOP to setup
    for calibration, editing and data translation.  If an output
    cataloged file is to be created this should be done after the
    call to UVGET.
(2) Uses AIPS LUN 24
```

6.7.2 CHNDAT

Routine to create/fill/read CH/FQ extension tables. We are phasing out CH tables, so this routine will read them, but will only write FQ tables.

```
CHNDAT (OPCODE, BUFFER, DISK, CNO, VER, CATBLK, LUN,
  *   NIF, FOFF, ISBAND, FREQID, IERR)
Inputs:
   OPCODE   C*4      Operation code:
                     'WRIT' = create/init for write or read
                     'READ' = open for read only
   BUFFER   I(512)   I/O buffer and related storage, also defines
                     file if open.
   DISK     I        Disk to use.
   CNO      I        Catalog slot number
   CATBLK   I(256)   Catalog header block.
   LUN      I        Logical unit number to use
   FREQID   I        Frequnecy ID #, if FQ tables exists
Input/Output:
   VER      I        CH file version
   NIF      I        Number of IFs.
   FOFF     D(*)     Frequency offset in Hz from ref. freq.
                        True = reference + offset.
   ISBAND   I(*)     Sideband of each IF.
                     -1 => 0 video freq. is high freq. end
                      1 => 0 video freq. is low freq. end
Output:
   IERR     I        Return error code, 0=>OK, else TABINI or TABIO
                     error, -1 => tried to create/write an FQ table
```

6.7.3 COMOFF

Compute the block offset of a 2-D map plane in a NDIM-dimensional map from the beginning of the map.

```
COMOFF (NDIM, NAX, DEPTH, BLKOF, IERR)
```

```
Inputs:
   NDIM      I   Number of axes in map
   NAX(7)    I   Number of pixels on each axis
   DEPTH(5 ) I   Depth of required plane along other axes
Outputs:
   BLKOF     I   Block offset
   IERR      I   Error return 0 = OK, 1= error in NDIM
```

6.7.4 DGGET

Gets requested data from visibility record, reformatting if needed. REQUIRES setup by DGINIT to set values of MVIS, JADR, SELFAC and ALLWT.

```
DGGET (VISIN, IND, MVIS, JADR, SELFAC, ALLWT, VISOUT, DROP)
Inputs:
   VISIN  R(IND,*)   Input visibility array
   IND    I          First dimension of VISIN (CATBLK(KINAX))
   MVIS   I          Number of visibilities in requested output
                     format.
   JADR   I(2,*)     Pointers to the first and second visibility
                     input records to be used in the output record.
                     If JADR(1,n) is negative use IABS (JADR(1,n))
                     and multiply the visibility by i (=SQRT(-1))
   SELFAC R(2,*)     Factors to be multiplied by the first and
                     second input vis's to make the output vis.
   ALLWT  L          Flag, = .TRUE. if all visibilities must have
                     positive weight.
  Output:
   VISOUT R(3,*)     Output visibility record
   DROP   L         .TRUE. if all data in record flaged.
```

6.7.5 DGINIT

Sets up tables for selecting data from vis. record. Checks if requested data in data base. Requires catalog header record from include DCAT.INC and setup of commons in INCLUDE DUVH.INC by UVPGET before call.

Note: STOKES='HALF' will work if only partial information (i.e. 1 polarization) is available in the data.

```
DGINIT (STOKES, BCHAN, ECHAN, BIF, EIF, MVIS, JADR, SELFAC, ALLWT,
  *   PMODE, IERR)
Inputs:
   STOKES   C*4      Desired output data format: 'I','V','Q','U',
                     'IQU','IQUV','IV','RR','LL','RL','LR'
                     'HALF' (=parallel pol.), 'FULL' (=RR,LL,RL,LR)
   BCHAN    I        First channel desired.
   ECHAN    I        Last channel desired.
   BIF      I        First IF desired.
   EIF      I        Last IF desired.
Input from common /MAPHDR/
   CATBLK   I(256)   Catalog header record.
Output:
   MVIS     I        Number of visibilities in requested output
                     format.
```

```
   JADR     I(2,*)   Pointers to the first and second visibility
                     input records to be used in the output record.
                     If JADR(1,n) is negative use IABS (JADR(1,n))
                     and multiply the visibility by i (=SQRT(-1))
   SELFAC   R(2,*)   Factors to be multiplied by the first and
                     second input vis's to make the output vis.
   ALLWT    L        Flag, = .TRUE. if all visibilities must have
                     positive weight.
   PMODE    I        Polarization mode:
                         1 = I,    2 = V,   3 = Q
                         4 = U,    5 = IQU, 6 = IQUV
                         7 = IV,   8 = RR,  9 = LL
                        10 = RL,  11 = LR, 12 = parallel (RR,LL)
                        13 = (RR,LL,RL,LR)
   IERR     I        Error flag. 0 => ok, 1 = unrecognized stokes,
                        2 = data unavailable.
```

6.7.6 EXTINI

EXTINI creates/opens an extension file. If a file is created it is cataloged by a call to CATIO which saves the updated CATBLK.

```
EXTINI (OPCODE, PTYP, VOL, CNO, VER, CATBLK, LUN, IND,
  *   LREC, NREC, BUFFER, IERR)
Inputs:
   OPCODE   C*4      Operation code, 'READ' => read only,
                                     'WRIT' => read/write
   PTYP     C*2      Physical extension type (eg. 'CC')
   VOL      I        Volume number
   CNO      I        Catalog slot number
   LUN      I        Logical unit number to use.
   NREC     I        Number of logical rec. for create/extend
In/out:
   VER      I        in: Version number: (<= 0 => write a new one,
                         read the latest one)
                     out: Version number used.
   CATBLK   I(256)   Catalog block of cataloged file, ext info is
                     updated if necessary.
   LREC     I        in: Record length in units of REALs (write new)
                     out: Logical record length (in units of REALs)
                          for read/write old files
   BUFFER   I(*)     Work buffer, at least 1024 bytes in size,
                     more if logical record longer than 512 bytes
                     out: Header info. for EXTIO
Output:
   IND      I        FTAB pointer.
   IERR     I        Return error code. 0 => OK
                                        1 => bad input.
                                        2 => could not find or open
                                        3 => create/I/O problem.
Useage notes:
For sequential access, EXTINI leaves pointers for EXTIO such that
if IRNO .le. 0 reads will begin at the start of the file and writes
will begin after the last previous record.
```

```
File should be marked 'WRIT' if the file is to be created.

Header record:
Each extension file using this system must have the first physical
(512 bytes) record containing necessary information.  In addition
space in this first record not reserved can be used for other
purposes.  The header record contains the following:

I   word(s)                  description
   1           # 512-byte records in the existing file
   2           # logical records to extend the file when req.
   3           max. # of logical records
   4           current number of logical records
   5           # bytes per value
   6           # values per logical record.
   7           # of logical records per physical record, if neg then
               the # of physical records per logical record.
 8 - 10        Creation task name (6 Hollerith characters)
11 - 16        Creation date, time
17 - 28        File name (48 Hollerith characters)
  29           Volume number on which file resides.
30 - 32        Last write-access task (6 Hollerith characters)
33 - 38        Last write-access time,date
39 - 56        reserved. (53-56 used by EXTIO:
                          53 = # I   words per logical record.
                          54 = IOP sent to EXTINI
                          55 = current physical record no.
                               (doesn't include header rec.)
                          56 = current logical rec. no.
57 -256        Available for use.
```

6.7.7 EXTIO

EXTIO does random access I/O to an extension files. Mixed reads and writes are allowed if EXTINI was called with 'WRIT'.

```
EXTIO (OPCODE, LUN, IND, IRNO, RECORD, BUFFER, IERR)
Inputs:
   OPCODE         C*4  Opcode 'READ','WRIT','CLOS'
   LUN            I    Logical unit number
   IND            I    FTAB pointer
   IRNO           I    Logical record no. 0=> next.
   RECORD(*)      I    Array containing record to be written
   BUFFER(*)      I    Work buffer = 512 bytes + enough 512 byte
                       blocks for at least one full logical record.
Output:
   RECORD(*)      I    Array containing record read.
   BUFFER(*)      I    buffer.
   IERR           I    Return error code 0 => OK
                             1 => file not open
                             2 => input error
                             3 => I/O error
                             4 => attempt to read past end of data
                                  or write past log. or phys.
```

```
                                    record 32766.
IMPORTANT NOTE:  the contents of BUFFER should not be changed
except by EXTIO between the time EXTINI is called until the file
is closed.  The exception is that the user portion of the header
record is available.
```

6.7.8 GETVIS

GETVIS gets and reformats uv data. Requires setup by SETVIS.

```
GETVIS (MODE, MVIS, JADR, SFACT, ALLWT, DATA, WT,
  *   VIS, IERR)
Inputs:
   MODE          I    Operation number (see SETVIS).
                      When MODE = 2 or 3 and RL and LR are given
                      the U visibility is multiplied by i.
   MVIS          I    Number of visibilities wanted.
   JADR(2,MVIS)  I    Pointers set by SETVIS.
   SFACT(2,MVIS) R    Factors set by SETVIS.
   ALLWT         L    Flag set by SETVIS, if .TRUE. all relevant
                      weights must be positive.
   DATA(3,*)     R    Visibility portion of input data.
Outputs:
   WT            R    Average weight.
   VIS(MVIS)     CMPX Visibilities.
   IERR          I    Error code, 0=>OK,
                               1 => bad weights.(data flagged).
                               2 = bad input.
```

6.7.9 GET1VS

GET1VS gets and reformats uv data. Returns one Stokes' type per frequency channel. Requires setup by SET1VS.

```
GET1VS (MODE, MVIS, JADR, JINC, SFACT, ALLWT, STOKES,
  *   DATA, WT, VIS, IRET)
Inputs:
   MODE          I    Operation number (see SET1VS).
                      When MODE = 3 and RL and LR are given,
                      the U visibility is multiplied by i.
   MVIS          I    Number of visibilities wanted.
   JADR(2)       I    Pointers set by SET1VS.
   JINC          I    Increment between vis.
   SFACT(2)      R    Factors set by SET1VS.
   ALLWT         L    If true all vis are required.
   STOKES        L    True if input data true Stokes'.
                      Used for UPOL only.
   DATA(3,*)     R    Visibility portion of input data.
Outputs:
   WT            R    Average weight.
   VIS(MVIS)     CMPX Visibilities.
   IRET          I    Error code, 0=>OK,
                               1 => bad weights.(data flagged).
                               2 = bad input.
```

6.7.10 KEYIN

Standard Fortran version of the CIT VLBI Keyin subroutines. These subroutines read keyed parameters on cards images. The text file should be opened via a call to ZTXOPN before the first call to KEYIN and closed via a call to ZTXCLS after the last call. (HINT: use LUN = 10) Note: in this version time like entries in the form hh:mm:ss will be returned in hours.

```
   KEYIN (KEYS, VALUES, VALCHR, N, ENDMRK, MODE, LUN,
     *   FIND, IERR)
  Inputs:
     KEYS(N)    C*8  Array of parameter names .
                     Array names should have the last characters
                     indicate the element number.  Should all be
                     in upper case characters.
     N          I    number of parameters (dimension of keys, values)
     ENDMRK     C*8  special keyword to indicate end of input
     MODE       I    1 = turn on reflection, 0 = turn off
                     2 = interactive mode (prompts for input,
                         no reflection, no limit on errors)
                     3 = Pass values until ENDMRK, File should not
                         contain keywords.
                     Note: currently only reads from a file.
     LUN        I    LUN to read from (used in call to ZTXOPN)
     FIND       I    FTAB pointer for input. (from ZTXOPN)
  Input/Output:
     VALUES(N)  D    array to receive numeric values or defaults,
                     each value corresponds to a KEY.
     VALCHR(N)  C*8  array to receive character values or defaults,
                     each value corresponds to a KEY.
  Outputs:
     N          I    (MODE=3 only) number of values found
     IERR       I    error code, 0=>OK, 1=>EOF found, 2=>Error
```

6.7.11 MAPSIZ

MAPSIZ computes the correct number of bytes to request from ZCREAT for a file using map I/O methods.

```
   MAPSIZ (NAX, NP, ISIZE)
   Inputs:
      NAX    I        # axes
      NP     I(NAX)   Number of points on each axis
   Output:
      ISIZE  I        File size in AIPS bytes
```

6.7.12 MAPCLS

closes a cataloged file, updates header on disk, clears catalog status.

```
   MAPCLS (OP, IVOL, CNO, LUN, IND, CATBLK, CATUP, WBUFF, IERR)
   Inputs:
      OP      C*4     OPcode used by MAPOPN to open this file
      IVOL    I       Disk volume containing map file
      CNO     I       Catalog slot number of file
      LUN     I       Logical unit # used for file
      IND     I       FTAB pointer for LUN
```

```
   CATBLK  I(256)  New catalog header which can optionally
                   be written into header if OP=WRIT or INIT
                   Dummy arguement if OP=READ
   CATUP   L       If TRUE, write CATBLK into catalog,
                   ignored if OP = READ
Outputs:
   IERR    I       0 = O.K.
                   1 = CATDIR couldn't access catalog
                   5 = illegal OP code
```

6.7.13 MAPOPN

MAPOPN opens a map file marking the catalog entry for the desired type of operation.

```
MAPOPN (OP, IVOL, NAMEIN, CLASIN, SEQIN, TYPIN, USID,
  *   LUN, IND, CNO, CATBLK, WBUFF, IERR)
Inputs:
   OP      C*4     Operation: READ, WRIT, or INIT where INIT is for
                   known creation processes (it ignores current file
                   status & leaves it unchanged).  Also: HDWR for
                   use when the header is being changed, but the
                   data are to be read only.
   LUN     I       Logical unit # to use
In/out:
   NAMEIN  C*12    Image name (name)
   CLASIN  C*6     Image name (class)
   SEQIN   I       Image name (seq.#)
   USID    I       User identification #
   IVOL    I       Input disk unit
   TYPIN   C*2     Physical type of file
Outputs:
   IND     I       FTAB pointer
   CNO     I       Catalog slot containing map
   CATBLK  I(256)  Buffer containing current catalog block
   WBUFF   I(256)  Working buffer for CATIO and CATDIR
   IERR    I       Error output: 0 = OK
                      2 = Can't open WRIT because file busy
                          or can't READ because file marked WRITE
                      3 = File not found
                      4 = Catalog i/o error
                      5 = Illegal OP code
                      6 = Can't open file
```

6.7.14 MCREAT

Subroutine to create a map file using the parameters in a CATBLK. The file will be cataloged and marked with WRITE status. The image name parameters incl. physical type must be filled in. A blank physical type is converted to 'MA'. The OUTSEQ default is applied (0 = highest matching+1). The name must be unique ignoring the physical type. The extension file areas of the CATBLK are cleared and the "DATE-MAP" string is filled in.

```
MCREAT (IVOL, CNO, WBUFF, IERR)
In/Outs:
   IVOL      I        Volume # on which to put file: 0 => ALL
```

```
                    on output has volume used
   WBUFF     I(256)   Working buffer
Outputs:
   CNO       I        Catalog slot number
   IERR      I        Error code; 0 => o.k.
                                  1 => couldnt create,no room
                                  2 => no create, duplicate name
                                  3 => no room in catalog
                                  4 => i/o problem on catalog
                                  5 => Other Create errors
                                  6 => no catalog file
Common: (in/out)
   CATBLK    I(256)   Catalog block (via common MAPHDR)
   CATB4     R(256)   Catalog block (equivalenced to CATBLK)
```

6.7.15 MDESTR

MDESTR will delete a catalog entry for a file, delete all extension files for that file, and then delete the file. The file must be in the REST state.

```
MDESTR (IVOL, ISLOT, CATBLK, IWBLK, INDEST, IERR)
Inputs:
   IVOL    I       disk volume number of the file.
   ISLOT   I       catalog slot number.
In/out:
   INDEST  I       number of extension files destroyed.
                   (if = -32000 on in, suppress normal msg)
Output:
   CATBLK  I(256)  the header block for this file.
   IWBLK   I(256)  work buffer.
   IERR    I       error code: 0 no error
                        1 = disk error
                        2 = map too busy
                        3 = destroy failed somehow
```

6.7.16 MDISK

MDISK reads or writes a row of an image. MDISK is called only after a call to MINIT and you should read the precursor remarks of MINIT. MDISK actually sets an array index (BIND) to the start of the next line wanted. Actual IO is done only when needed and a row is written not on “its” call to MDISK but on some subsequent call (or the FINI call).

```
MDISK (OP, LUN, FIND, BUFF, BIND, IERR)
Inputs:
   OP    C*4   Op code 'WRIT', 'READ', 'FINI' (flush write buffers)
   LUN   I     logical unit number
   FIND  I     Pointer to FTAB returned by ZOPEN
Input and output:
   BUFF  R(*)  Buffer holding data
Output:
   BIND  I     Pointer to position in buffer of first pixel in
               window in the present line
   IERR  I     Error return: 0 => ok
                     1 => file not open
```

```
                    2 => input error
                    3 => I/O error
                    4 => end of file
                    5 => beginning of medium
                    6 => end of medium
```

6.7.17 MINIT

MINIT sets up a special section of FTAB for quick-return, double buffered I/O. N.B. This routine is designed to read/write images one plane at a time.

```
MINIT (OP, LUN, IND, LX, LY, WIN, BUFF, BFSZ, BLKOF, IERR)
Inputs:
   OP    C*4   Operation code character string: 'READ','WRIT','UPDT'
   LUN   I     logical unit number
   IND   I     pointer to FTAB, returned by ZOPEN
   LX    I     Number of pixels per line in X-direction for whole
               map
   LY    I     Number of lines in whole plane
   WIN   I(4)  Xmin,Ymin,Xmax,Ymax defining desired subrectangle in
               the current plane
   BFSZ  I     Size of total available buffer in AIPS bytes
   BLKOF I     block number, 1 relative, of first map pixel in this
               plane of the image
Outputs:
   IERR  I     Error return: 0 => ok
                          1 -> file not open
                          2 => input error
                          7 => Buffer too small
                          3 => i/o error on initialize
                          4 => end of file
                          5 => beginning of medium
                          6 => end of medium
Usage notes: For map i/o the first 16 words in each FTAB entry
contain a user table to handle double buffer i/o,  the rest
contain system-dependent IO tables.  A "major line" is 1 row or
1 sector if more than 1 line fits in a sector.  FTAB user table
entries, with offsets from the FIND pointer are:
   FTAB + 0 =>  LUN using this entry
          1 =>  No. of major lines transfered per i/o op
          2 =>  No. of major times a buffer has been acessed
          3 =>  No. of major lines remaining on disk
          4 =>  Output index for first pixel in window
          5 =>  No. pixels to increment for next major line
          6 =>  Which buffer to use for i/o; -1 => single buffer
          7 =>  Block offset in file for next operation
          8 =>
          9 =>  Block increment in file for each operation
         10 =>  No. of bytes transferred
         11 =>  I/O op code
         12 =>  sum of any buffer numbers needing to be waited upon
         13 =>  # rows / major line (>= 1)
         14 =>  # times this major line has been accessed
         15 =>  # pixels to increment for next row (= LX)
```

6.7.18 MINSK

MINSK initializes the use of MSKIP to read noncontigious but evenly spaced rows in a map. Read is double buffered if possible; in which case MINSK initiates the first read. Single buffering is used if the desired data cannot be double buffered. If more data is required than will fit in the buffer, multiple (NBUF) equally filled buffers are obtained by NBUF calls to MSKIP.

```
MINSK (LUN, FIND, LROW, NROW, ISTRT, NSKIP, BUFF, BUFSZ, BO, NBUF,
  *   IERR)
Inputs:
   LUN    I     Logical unit number.
   FIND   I     pointer to FTAB returned by ZOPEN.
   LROW   I     Length of a row in pixels.
   NROW   I     Total number of rows this plane.
   ISTRT  I     First row for read.
   NSKIP  I     Number of rows to skip.
   BUFF   R(*)  Output buffer.
   BUFSZ  I     Buffer size in AIPS bytes.
   BO     I     Block offset
   NBUF   I     factor times which LROW is multiplied normally = 1.
OUTPUT:
  NBUF    I     number of buffer fulls to complete read of row.
                MSKIP must be called this number of times to
                complete the read.
  IERR    I     Error code: 0 = OK
                     1 = file not open
                     2 = input error
                     4 = tried to read past end of map.
                   10+ = 10 + ZMIO or ZWAIT error.
FTAB assigments:
           0 = LUN
           1 =
           2 = BO block offset
           3 =
           4 = length of row / [5] in bytes
           5 = multiplier of [4]
           6 = next record number.
           7 = record increment+1 (total increment)
           8 = # calls per record.
           9 = record call # (when MSKIP is called)
          10 = bytes / call
          11 = buffer flag, -1= single, 1=>current buffer is 1
               2=>current buffer=2 (buffer already read)
          12 = buffer size in pixels (1/2 for double buffering)
          13 = NROW (the number of rows to read)
          14 = BTYOFF the byte offset when double buffering.
```

6.7.19 MSKIP

MSKIP reads rows in a map file which are evenly spaced. The reads are double, single buffered or partial buffers if the row size 1) is $\leq$ BUFSZ/2, 2) between BUFSZ/2 and BUFSZ or 3).GT.BUFSZ. For case 3) multiple calls (NBUF from MINSK) are required to read each row. Each call returns LROW*2/NBUF bytes and I/O is single buffered. IFIN = 0 indicates a row is completed. See MINSK for more details.

```
MSKIP (LUN, FIND, BUFF, BIND, IFIN, IERR)
```

```
Input:
   LUN    I      Logical unit number.
   FIND   I      pointer for FTAB
   BUFF   R(*)   Buffer
Output:
   BIND   I      Pointer for BUFF
   IFIN   I      0 if row complete, 1 otherwise.
   IERR   I      error code: 0 = OK
                             1 = file not open
                             2 = attempt to read past end of map.
                            10+= I/O error = 10 +  ZWAIT error.
```

6.7.20 PLNGET

PLNGET reads a selected portion of a selected plane parallel to the front and writes it into a specified scratch file. The output file will be zero padded and a shift of the center may be specified. If the input window is unspecified (0's) and the output file is smaller than the input file, the NX x NY region about position (MX/2+1-OFFX, MY/2+1-OFFY) in the input map will be used where MX,MY is the size of the input map. NOTE: If both XOFF and/or YOFF and a window (JWIN) which does not contain the whole map, XOFF and YOFF will still be used to end-around rotate the region inside the window. The image header is taken from the disk catalog AND explicitly will not handle blanked images.

```
PLNGET (IDISK, ICNO, CORN, JWIN, XOFF, YOFF, NOSCR,
  *   NX, NY, BUFF1, BUFF2, BUFSZ1, BUFSZ2, LUN1, LUN2, IRET)
Inputs:
   IDISK    I      Input image disk number.
   ICNO     I      Input image catalog slot number.
   CORN     I(7)   BLC in input image (1 & 2 ignored)
   JWIN     I(4)   Window in plane.
   XOFF     I      offset in cells in first dimension of the center
                   from MX/2+1 (MX 1st dim. of input win.)
   YOFF     I      offset in cells in second dimension of the center
                   from MY/2+1 (MY 2nd dim. of input win.)
   NOSCR    I      Scratch file number in common /CFILES/ for outpu.
   NX       I      Dimension of output file in X
   NY       I      Dimension of output file in Y
   BUFF1    R(*)   Work buffer
   BUFF2    R(*)   Work buffer.
   BUFSZ1   I      Size in AIPS bytes of BUFF1
   BUFSZ2   I      Size in AIPS bytes of BUFF2
   LUN1     I      Logical unit number for input file
   LUN2     I      Logical unit number to use for output
Output:
   IRET     I      Return error code, 0 => OK,
                    1 = couldn't copy input CATBLK
                    2 = wrong number of bits/pixel in input map.
                    3 = input map has inhibit bits.
                    4 = couldn't open output map file.
                    5 = couldn't init input map.
                    6 = couldn't init output map.
                    7 = read error input map.
                    8 = write error output map.
                    9 = error computing block offset
                    10 = output file too small.
```

```
Common:
   DCAT.INC CATBLK  is set to the input file CATBLK.
```

6.7.21 PLNPUT

PLNPUT writes a subregion of a scratch file image into a cataloged image.

```
PLNPUT (IDISK, ICNO, CORN, JWIN, NOSCR, NX, NY, BUFF1,
  *   BUFF2, BUFSZ1, BUFSZ2, LUN1, LUN2, IRET)
Input:
   IDISK    I      Output image disk number.
   ICNO     I      Output image catalog slot number.
   CORN     I(7)   BLC in Output image (1 & 2 ignored)
   JWIN     I(4)   Window in plane in input image.
   NOSCR    I      Scratch file number in common /CFILES/ for
                   input scratch file.
   NX       I      X-dimension of input file.
   NY       I      Y-dimension of input file.
   BUFF1    R(*)   Work buffer
   BUFF2    R(*)   Work buffer.
   BUFSZ1   I      Size in bytes of BUFF1.
   BUFSZ2   I      Size in bytes of BUFF2
   LUN1     I      Logical unit number to use.
   LUN2     I      Second loical unit number to use.
Output:
   IRET     I      Return error code: 0 => OK
                      1 = couldn't read output CATBLK.
                      2 = Output bits/pixel not allowed.
                      3 = Output and input windows not same.
                      4 = couldn't open input map file.
                      5 = couldn't init output map.
                      6 = couldn't init input map.
                      7 = read error input map.
                      8 = write error output map.
                      9 = error writing header to catalog
                     10 = error computing block offset.
Commons:
   CATBLK in /MAPHDR/ is used as the map header.
          Of particular importance is the data max/min values
          which must apply to the map.  As this is read from the
          catalog it must be updated by a call to CATIO etc.
          before calling this routine.
```

6.7.22 SCREAT

SCREAT creates scratch files. It uses the Common included via the DFIL.INC INCLUDE and returns the scratch file disk and catalog number in variables SCRVOL(NSCR) and SCRCNO(NSCR), where NSCR is updated on successful creation. It attempts to avoid the disk used for the previously created scratch file. All files have physical name SCvccc01 where v is the revision code and ccc is the catalog slot number. Their logical names are determined from the routine BLDSNM.

```
SCREAT (SIZE, WBUFF, IERR)
Input:
   SIZE    I          Desired size in AIPS bytes
```

```
Output:
   WBUFF   I(512)     Scratch buffer    (NOTE 512 integers)
   IERR    I          error: 0 => ok
                         1 => catalog error in setting name
                         2 => catalog error on open
                         3 => CATIO error writing header to catlg
                         4 => No allowed disk with room
Commons:
   /MAPHDR/   in      scratch file image header - contents mostly
                      ignored
   /CFILES/   in/out  file info
Note: this common uses IBAD to specify BADDISKs which are avoided.
```

6.7.23 SDGET

Subroutine to obtain data from a single dish data base with optional application of flaging and/or calibration and/or pointing information. Reads data with a large variety of selection criteria and will reformat the data as necessary. Does many of the startup operations, finds Single dish uv like data file etc., reads CATBLK and updates the /UVHDR/ commons (INCLUDE DUVH.INC) to reflect the output rather than input data.

```
SDGET (OPCODE, RPARM, VIS, IERR)
Input:
   OPCODE   C*4      Opcode -
                     'INIT' => Open files Initialize I/O.
                     'READ' => Read next specified record.
                     'CLOS' => Close files.
Inputs via common /SELCAL/  (Include DSEL.INC)
   UNAME    C*12     AIPS name of input file.
   UCLAS    C*6      AIPS class of input file.
   UDISK    R        AIPS disk of input file.
   USEQ     R        AIPS sequence of input file.
   SOURCS   C(30)*16 Names of up to 30 sources, '*' => all
                     First character of name '-' => all except those
                     specified.
   TIMRNG   R(8)     Start day, hour, min, sec, end day, hour,
                     min, sec.   0's => all.
   UVRA     R(2)     Range of RA (1) and dec (2) in degrees about
                     the value in CATBLK at time of READ call to
                     SDGET.  0=>all.
   STOKES   C*4      Stokes types wanted.
                     'I','Q','U','V','R','L','IQU','IQUV'
                     '    '=> Leave data in same form as in input.
   BCHAN    I        First channel number selected, 1 rel. to first
                     channel in data base.    0 => all
   ECHAN    I        Last channel selected.   0 => all
   BIF      I        First IF number selected, 1 rel. to first
                     IF in data base.    0 => all
   EIF      I        Last IF selected.   0 => all
   DOCAL    L        If true apply calibration, else not.
   SUBARR   I        Subarray desired, 0 => all
   FGVER    I        FLAG file version number, if < 0 then
                     NO flagging is applied. 0 => use highest
                     numbered table.
   CLUSE    I        Cal (CS) file version number to apply.
```

```
Output:
   RPARM    R(*)     Random parameter array of datum.
   VIS      R(3,*)   Regular portion of data array.
   IERR     I        Error code: 0 => OK,
                        -1 => end of data
                        >0 => failed, abort process.
Output in common /SELCAL/: The default values will be filled in
if null values were specified.
   CATBLK   I(256)   Catalog header block, describes the output
                     data rather than input.
   NPRMIN   I        Number or random parameters in the input data.
   TRANSL   L        If true translate data to requested Stokes'
   CNTREC   I(2,3)   Record counts:
                     (1&2,1) Previously flagged (partly, fully)
                     (1&2,2) Flagged due to gains (part, full)
                     (1&2,3) Good selected (part, full)
Usage notes:
 1) Include DSEL.INC should be declared in the main
    program or at a level that they will not be overlaid while
    SDGET is in use (ie. between the 'INIT' and 'CLOS' calls)
 2) If no sorting is done SDGET uses AIPS luns 25, 28, 29 and 30
   (1 map, 3 non map files).  If sorting is done (usually possible)
   then 8 map and 3 non map files are used (mostly on OPCODE='INIT')
   and LUNs 16,17,18,19,20,21,22,23,24,25, 28,29,30.
 3) OPCODE = 'INIT' does the following:
   - The cataloged data file is located and the catalog header
     record is read.
   - The index file (if any) is initialized.
   - The flag file (if any) is initialized and sorted if necessary
     (Must be in time order).
   - The CS table (if any) is initialized.
   - I/O to the input file is initialized.
         The following LUNs may be used but will be closed on
     return: 16, 17, 18, 19, 20, 21, 22, 23, 24
         The following LUNs may be used but will be open on
     return: 25 (uv data), 28 (NX table), 29 (CS table),
             30 (FG table).
         NO data are returned from this call.
 4) OPCODE = 'READ' reads one  record properly selected,
    transformed (e.g. I pol.), calibrated and edited as requested
    in the call with OPCODE = 'INIT'
 5) OPCODE = 'CLOS' closes all files used by SDGET which are still
    open.  No data are returned.
 6) If DOCAL is true then the common array CNTREC will contain the
    counts of records which are good or fully or partly flagged
    both previously and due to flagged gain solutions.
```

6.7.24 SELINI

Subroutine to initialize the control values for UVGET in commons in DSEL.INC.

```
    SELINI
Outputs via common /SELCAL/  (Include DSEL.INC)
   UNAME    C*12      AIPS name of input file. (blank)
   UCLAS    C*6       AIPS class of input file. (blank)
```

```
   UDISK    R          AIPS disk of input file. (0.0)
   USEQ     R          AIPS sequence of input file. (0.0)
   SOURCS   C(30)*16   Names of up to 30 sources. (blank)
   SELQUA   I          Qualifier wanted (-1 => all)
   SELCOD   C*4        Cal code ('    ')
   TIMRNG   R(8)       Timerange (0s => all)
   UVRNG    R(2)       Baseline range (0s => all)
   STOKES   C*4        Stokes types wanted. (blank)
   BCHAN    I          First channel number selected, (1)
   ECHAN    I          Last channel selected. (0=>all)
   BIF      I          First IF number selected. (1)
   EIF      I          Last IF selected. (0=>all)
   DOCAL    L          If true apply calibration. (false)
   DOPOL    L          If true then correct polarization (false)
   DOACOR   L          True if autocorrelations wanted (false)
   DOXCOR   L          True if cross-correlations wanted (true)
   DOWTCL   L          True if weight calibration wanted. (false)
   DOFQSL   L          True if FREQSEL random parm present (false)
   FRQSEL   I          Default FQ table entry to select (-1)
   SELBAN   R          Bandwidth (Hz) to select (-1.0)
   SELFRQ   D          Frequency (Hz) to select (-1.0)
   DOBAND   I          >0 if bandpass calibration. (-1)
   BPNAME   C*48       Name of scratch file set up for BP's.
   DOSMTH   L          True if smoothing requested. (false)
   SMOOTH   R(3)       Smoothing parameters (0.0s)
   DXTIME   R          Integration time (days). (1 sec)
   ANTENS   I(50)      List of antennas selected. (0=>all)
   SUBARR   I          Subarray desired. (0=>all)
   FGVER    I          FLAG file version number. (0)
   CLUSE    I          Cal (CL or SN) file version number (0)
   BLVER    I          BL Table to apply (-1)
   BPVER    I          BP table to apply (-1)
```

6.7.25 SETVIS

SETVIS setup the arrays JADR, SFACT and the flag ALLWT for reformatting uv data as specified by MODE. There is also a check to make sure the desired data is available. Calls to GETVIS will reformat the data. Needs values set by UVPGET and VHDRIN. Only 1 IF will be processed.

```
SETVIS (MODE, NCH, IFNUM, MVIS, JADR, SFACT, ALLWT, IERR)
Inputs:
   MODE    I        Desired output data format:
                     1 => I
                     2 => IQU
                     3 => IQUV
                     4 => IV
                     5 => R (right hand circular)
                     6 => L
                     7 => RL
                     8 => straight correlators (used in UVFND)
                     10+n => n I pol. line maps. (n .le. 8)
                     20+n => n R pol. line maps.
                     30+n => n L pol. line maps.
   NCH     I        First line channel desired.
```

```
   IFNUM   I        IF number wanted.
Output:
   MVIS    I        Number of visibilities in requested output
                    format.
   JADR    I(2,*)   Pointers to the first and second visibility
                    input records to be used in the output record.
   SFACT   R(2,*)   Factors to be multiplied by the first and
                    second input vis's to make the output vis.
   ALLWT   L        Flag, = .TRUE. if all visibilities must have
                    positive weight.
   IERR    I        Error flag. 0 =>OK, otherwise data unavailable.
Common (input):
   DCAT.INC  must have uv header
   DUVH.INC  must be initialized by UVPGET
```

6.7.26 SET1VS

SET1VS setup the arrays JADR, SFACT and the flag ALLWT for reformatting uv data as specified by MODE. One visibility per frequency channel will be returned by GET1VS. There is also a check to make sure the desired data is available. Calls to GET1VS will reformat the data. Needs values set by UVPGET.

```
SET1VS (MODE, NCH, JADR, SFACT, ALLWT, JINC, IRET)
Inputs:
   MODE         I    Desired output data format:
                     1 => I
                     2 => Q
                     3 => U
                     4 => V
                     5 => RCP
                     6 => LCP
   NCH          I    First line channel desired.
Output:
   JADR(2)    I    Pointers to the first and second visibility
                     input records to be used in the output record.
   SFACT(2)     R    Factors to be multiplied by the first and
                     second input vis's to make the output vis.
   ALLWT        L    If true no flagged data is allowed.
   JINC         I    Visibility increment.
   IRET         I    Error flag. 0 =>OK, otherwise data unavailable.
```

6.7.27 TABINI

TABINI creates/opens a table extension file. If a file is created, it is cataloged by a call to CATIO which saves the updated CATBLK.

```
TABINI (OPCODE, PTYP, VOL, CNO, VER, CATBLK, LUN, NKEY,
  *   NREC, NCOL, DATP, NBUF, BUFFER, IERR)
Input:
   OPCODE   C*4       Operation code, 'READ' => read only,
                                      'WRIT' => read/write
   PTYP     IC*2      Physical extension type (eg. 'CC')
   VOL      I         Disk volume number
   CNO      I         Catalog slot number
   CATBLK   I(256)    Catalog block of cataloged file.
```

```
   LUN      I         Logical unit number to use.
   NREC     I         Number of logical rec. for create/extend
   NBUF     I         Number I   words in BUFFER
In/out:
   VER      I         Version number: (<= 0 => write a new one,
                      read the latest one), returns one used.
   NKEY     I         Maximum number of keyword/value pairs
                      input: used in create, checked on write old
                      (0 => any); output: actual
   NCOL     I         Number of logical columns (does not include
                      selection column).  Input: used in create,
                      checked on write old (0=>any); output: actual
   DATP     I(128,2)  DATP(*,1) address pointers (output only)
                      DATP(*,2) column data type codes. Input:
                      used in create only; output: actual.
   BUFFER   I(*)      Work buffer, at least 1024 bytes in size,
                      more if logical record longer than 512 bytes
                      Output: control info, lookup table, ...
Output:
   IERR     I         Return error code. 0 => OK
                                    -1 => OK, created new file
                                     1 => bad input.
                                     2 => could not find or open
                                     3 => I/O problem.
                                     4 => create problem.
                                     5 => not a table file
Usage notes:
For sequential access, TABINI leaves pointers for TABIO such that,
if IRNO <= 0, reads will begin at the start of the file and writes
will begin after the last previous record.  Cataloged file should
be marked 'WRIT' if the file is to be created.

Header record:
Each extension file using this system must have the first physical
(512 bytes) record containing necessary information. The full table
file format is described in Going AIPS.  The user must read this
section to understand fully how to use such files.  The header
record contains the following:

 I   word(s)         Description
 1              Number 512-byte records now in file
 2
 3              Max number rows allowed in current file
 4
 5              Number rows (logical records) now in file
 6
 7              Number of bytes/value (2 for TA files)
 8              Number values / logical (# Is / row for TA)
 9              > 0 => number rows / physical record
                < 0 => number physical records / row
10              Number logical columns / row
11 - 16         Creation date: ZDATE(11), ZTIME(14)
17 - 28     H   Physical file name (set on each TABINI call)
29 - 30     H   Creation task name
```

```
 31
 32             Disk number
 33 - 38        Last access date: ZDATE(33), ZTIME(36)
 39 - 40    H   Last access task name
 42             Number logical records to extend file if needed
 43             Sort order: logical column # of primary sorting
 44             Sort order: logical column # of secondary sorting
                     0 => unknown, < 0 => descending order
 45             Disk record number for column data pointers (2)
 46             Disk record number for row selection strings (3)
 47             Disk record number for 1st record of titles (5)
 48             Disk record number for 1st record of units
 49             Disk record number for 1st record of keywords
 50             Disk record number for 1st record of table data
 51             DATPTR (row selection column)
 52             Maximum number of keyword/value pairs allowed
 53             Current number of keyword/value pairs in file
 54 - 56        "*AIPS TABLE*" packed string to verify that table.
 57 - 59
 60             If 1 then then table cannot be written as FITS ASCII
 61             Number of selection strings now in file
 62             Next available R   address for a selection string
 63             First R   address of selection string 1
 64             First R   address of selection string 2
 65             First R   address of selection string 3
 66             First R   address of selection string 4
 67             First R   address of selection string 5
 68             First R   address of selection string 6
 69             First R   address of selection string 7
 70             First R   address of selection string 8
********** for TABIO / TABINI use only **********
 71             IOP : 1 => read, 2 => writ
 72             Number I   words per logical record
 73             Current table row physical record in BUFFER
 74
 75             Current table row logical record in BUFFER
 76
 77             Type of current record in BUFFER
 78             Current control physical record number in BUFFER
 79             Current control logical record number in BUFFER
 80             Type of current control record in BUFFER
 81             LUN
 82             FTAB pointer of open file
***********
 83 -100        Reserved
***********
101 -128    H   Table title
129 -256        lookup table as COLPTR(logical column) = phys column
```

6.7.28 TABIO

TABIO does random access I/O to Tables extension files. Mixed reads and writes are allowed if TABINI was called 'WRIT'. Files opened for WRITe are updated and compressed on CLOS.

```
TABIO (OPCODE, IRCODE, IRNO, RECORD, BUFFER, IERR)
Inputs:
   OPCODE   C*4      Opcode 'READ','CLOS'
                        'WRIT' : write data as selected
                        'FLAG' : write data as de-selected
   IRCODE   I        Type of information
                        0 => Table row
                        1 => DATPTR/DATYPE record
                        2 => data selection string
                        3 => title
                        4 => units
                        5 => keyword/value pair
   IRNO     I        Logical record number. 0 => next (can work
                     with row data and latest IRCODE > 0 only)
                        IRNO is row number (IRCODE = 0)
                        IRNO is ignored (IRCODE = 1)
                        IRNO is string number (IRCODE = 2)
                        IRNO is column number (IRCODE = 3)
                        IRNO is column number (IRCODE = 4)
                        IRNO is keyword number (IRCODE = 5)
   RECORD   I(*)     Array containing record to be written
   BUFFER   I(*)     Work buffer = 512 bytes + enough 512 byte
                     blocks for at least one full logical record.
                     Must be the same one given TABINI.
Output:
   RECORD   I(*)     Array containing record read.
   BUFFER   I(*)     buffer.
   IERR     I        Return error code 0 => OK
                         -1 => on READ: row read is flagged
                          1 => file not open
                          2 => input error
                          3 => I/O error
                          4 => attempt to read past end of data
                               or write past end of data + 1
                          5 => error on expanding the file
IMPORTANT NOTE:  the contents of BUFFER should not be changed
except by TABIO between the time TABINI is called until the file
is closed.  The exception is that the user portion of the header
record is available.
```

6.7.29 UVCREA

Subroutine to create a uv file using the parameters in a CATBLK. The file will be cataloged and marked with WRITE status. The image name parameters must be filled in except that the physical type is converted to 'UV'. The OUTSEQ default is applied (0 = highest matching+1). The name must be unique ignoring the physical type. The extension file areas of the CATBLK are cleared and the "DATE-MAP" string is filled in.

```
UVCREA (IVOL, CNO, WBUFF, IERR)
In/Outs:
   IVOL      I        Volume # on which to put file.  0 => any
                      on output is volume used (IERR = 0)
Outputs:
   WBUFF     I(256)   Working buffer
   CNO       I        Catalog slot number
```

```
   IERR      I        Error code; 0 => o.k.
                                  1 => couldnt create,no room
                                  2 => no create, duplicate name
                                  3 => no room in catalog
                                  4 => i/o problem on catalog
                                  5 => Other Create errors
                                  6 => No catalog file on disk
COMMON: /MAPHDR/ catalog block used a lot, final seq # on output
```

6.7.30 UVDISK

UVDISK reads and writes records of arbitrary length especially UV visibility data. Operation is faster if blocks of data are integral numbers of disk blocks. There are three operations which can be invoked: READ, WRITE and FLUSH (OPcodes 'READ', 'WRIT' and 'FLSH').

'READ' reads the next sequential block of data as specified to UVINIT and returns the number of visibilities in NIO and sets the pointer in BUFFER to the first word of this data.

'WRIT' arranges data in a buffer until it is full. Then as many full blocks as possible are written to the disk with the remainder left for the next disk write. For writes, left-over data is transfered to the beginning of buffer 1 if that is the next buffer to be filled. The value of NIO in the call is the number of vis. rec. to be added to the buffer and may be fewer than the number specified to UVINIT. On return NIO is the maximum number which may be sent next time. On return BIND is the pointer in BUFFER to begin filling new data.

'FLSH' writes integral numbers of blocks and moves any data left over to the beginning of buffer 1. One exception to this is when NIO ≤ 0, in which case the entire remaining data in the buffer is written (if NIO > 0 then ABS (NIO) visibilities are to be written). After the call BIND is the pointer in BUFFER for new data. The principal difference between 'FLSH' and 'WRIT' is that FLSH always forces an I/O transfer. This may cause trouble if a transfer of less than 1 block is requested. A call with a nonpositive value of NIO should be the last call and corresponds to a call to MDISK with opcode 'FINI'.

NOTE: A call to UVINIT is REQUIRED prior to calling UVDISK.

```
UVDISK (OP, LUN, FIND, BUFFER, NIO, BIND, IERR)
Inputs:
   OP       C*4   Opcode 'READ','WRIT','FLSH' are legal
   LUN      I     Logical unit number
   FIND     I     FTAB pointer returned by ZOPEN
   BUFFER   I(*)  Buffer for I/O
   NIO      I     No. additional visibilities to write.
Output:
   NIO      I     No. visibilities read.
                  Max. no. vis. for next write.
   BIND     I     Pointer to start of data in buffer
   IERR     I     Return error code: 0 => OK
                     1 => file not open in FTAB
                     2 => input error
                     3 => I/O error
                     4 => end of file
                     7 => attempt to write more vis than specified
                          to UVINIT or will fit in buffer.
```

6.7.31 UVGET

Subroutine to obtain data from a data base with optional application of flaging and/or calibration information. Reads data with a large variety of selection criteria and will reformat the data as necessary. Does many of the startup operations, finds uv data file etc, reads CATBLK and updates the DUVH.INC commons to reflect the output rather than input data. Most of the input to UVGET is through the commons in DSEL.INC; the initial (default) values of these may be set using routine SELINI.

```
UVGET (OPCODE, RPARM, VIS, IERR)
Input:
   OPCODE   C*4       Opcode:
                      'INIT' => Open files Initialize I/O.
                      'READ' => Read next specified record.
                      'CLOS' => Close files.
Inputs via common (Include DSEL.INC)
   UNAME    C*12      AIPS name of input file.
   UCLAS    C*6       AIPS class of input file.
   UDISK    R         AIPS disk of input file.
   USEQ     R         AIPS sequence of input file.
   SOURCS   C(30)*16  Names of up to 30 sources, *=>all
                      First character of name '-' => all except
                      those specified.
   TIMRNG   R(8)      Start day, hour, min, sec, end day, hour,
                      min, sec. 0's => all
   UVRNG    R(2)      Minimum and maximum baseline lengths in
                      1000's wavelengths. 0's => all
   STOKES   C*4       Stokes types wanted.
                      'I','Q','U','V','R','L','IQU','IQUV'
                      '    '=> Leave data in same form as in input.
   BCHAN    I         First channel number selected, 1 rel. to first
                      channel in data base. 0 => all
   ECHAN    I         Last channel selected. 0=>all
   BIF      I         First IF number selected, 1 rel. to first
                      IF in data base. 0 => all
   EIF      I         Last IF selected. 0=>all
   DOCAL    L         If true apply calibration, else not.
   DOPOL    L         If true then correct for feed polarization
                      based on antenna file info.
   DOSMTH   L         True if smoothing requested.
   DOACOR   L         True if autocorrelations are requested.
   DOWTCL   L         True if weight calibration wanted.
   DOFQSL   L         True if FREQSEL random parm present (false)
   FRQSEL   I         Default FQ table entry to select (-1)
   SELBAN   R         Bandwidth (Hz) to select (-1.0)
   SELFRQ   D         Frequency (Hz) to select (-1.0)
   DOBAND   I         >0 if bandpass calibration. (-1)
   BPNAME   C*48      Name of scratch file set up for BP's.
   DOSMTH   L         True if smoothing requested. (false)
   SMOOTH   R(3)      Smoothing parameters (0.0s)
   DXTIME   R         Integration time (days). Used when applying
                      delay corrections to correct for delay error.
   ANTENS   I(50)     List of antennas selected, 0=>all,
                      any negative => all except those specified
   SUBARR   I         Subarray desired, 0=>all
   FGVER    I         FLAG file version number, if < 0 then
                      NO flagging is applied. 0 => use highest
                      numbered table.
   CLUSE    I         Cal (CL or SN) file version number to apply.
   BLVER    I         BL Table to apply .le. 0 => none
```

```
   BPVER    I         BP table to apply .le. 0 => none
Output:
   RPARM    R(*)      Random parameter array of datum.
   VIS      R(3,*)    Regular portion of visibility data.
   IERR     I         Error code: 0 => OK,
                          -1 => end of data
                          >0 => failed, abort process.
Output in commons in DSEL.INC: The default values will be filled in
if null values were specified.
   UVFREQ   D         Frequency corresponding to u,v,w
   CATBLK   I(256)    Catalog header block, describes the output
                      data rather than input.
   NPRMIN   I         Number or random parameters in the input data.
   TRANSL   L         If true translate data to requested Stokes'
   CNTREC   I(2,3)    Record counts:
                      (1&2,1) Previously flagged (partly, fully)
                      (1&2,2) Flagged due to gains (part, full)
                      (1&2,3) Good selected (part, full)
   ISCMP    L         True if input data is compressed.
   KLOCSU   I         0-rel random parm. pointer for source in input
                      file.
   KLOCFQ   I         0-rel random parm. pointer for FQ id in input
                      file.
   KLOCIF   I         0-rel random parm. pointer for IF in input
                      file.
   KLOCFY   I         0-rel random parm. pointer for freq. in input
                      file.
   KLOCWT   I         0-rel random parm. pointer for weight in
                      input file.
   KLOCSC   I         0-rel random parm. pointer for scale in
                      input file.
Usage notes:
 1) Include DSEL.INC should be declared in the main program or at a
    level that they will not be overlaid while UVGET is in use (ie.
    between the 'INIT' and 'CLOS' calls). SELINI can be used to
    initialize the control variables in these commons.
 2) If no sorting is done UVGET uses AIPS luns 25, 28, 29 and 30
   (1 map, 3 non map files).  If sorting is done (usually possible)
   then 8 map and 3 non map files are used (mostly on OPCODE='INIT')
   and LUNs 16,17,18,19,20,21,22,23,24,25, 28,29,30,40,42,43,44,45.
 3) OPCODE = 'INIT' does the following:
   - The catalgue data file is located and the catalog header
     record is read.
   - The source file (if any) is read.
   - The index file (if any) is initialized.
   - The flag file (if any) is initialized and sorted if necessary
     (Must be in time order).
   - The gain table (if any) is initialized.
   - The bandpass table (if any) is initialized
   - The smoothing convolution table (if any) is initialized
   - I/O to the input file is initialized.
         The following LUNs may be used but will be closed on
     return: 16, 17, 18, 19, 20, 21, 22, 23, 24
         The following LUNs may be used but will be open on
```

```
      return: 25 (uv data), 28 (NX table), 29 (CL or SN table),
              30 (FG table), 40 (BL table), 41 (BP table).
          NO data are returned from this call.
   4) OPCODE = 'READ' reads one visibility record properly selected,
      transformed (e.g. I pol.), calibrated and edited as requested
      in the call with OPCODE = 'INIT'
   5) OPCODE = 'CLOS' closes all files used by UVGET which are still
      open.  No data are returned.
   6) If DOCAL is true then the common array CNTREC will contain the
      counts of records which are good or fully or partly flagged
      both previously and due to flagged gain solutions.
   7) Only one subarray can be calibrated at a time if DOPOL is true.
      This is because the polarization information for only one
      subarray is kept at a time.
```

6.7.32 UVINIT

UVINIT sets up bookkeeping for the UV data I/O routine UVDISK. I/O for these routines is double buffered (if possible) quick return I/O. UVDISK will run much more efficiently if on disk LREC*NPIO is an integral number of blocks. Otherwise, partial writes or oversize reads will have to be done. Minimum disk I/O is one block. Smaller calls to UVINIT may be made as long as the buffer is large enough. The buffer size should include an extra NBPS bytes for each buffer for read if NPIO records does not correspond to an integral number of disk sectors (NBPS bytes). 2*NBPS extra bytes required for each buffer for write. NPIO will be adjusted to the maximum allowed for double buffering if the input value is .LE. 0, or the maximum allowed single buffering value if NPIO is too large. If it is positive and useable it is used.

```
UVINIT (OP, LUN, FIND, NVIS, VISOFF, LREC, NPIO, BUFSZ,
  *   BUFFER, BO, BIND, IERR)
Inputs:
   OP       C*4    OP code, 'READ' or 'WRIT' for desired operation.
   LUN      I      Logical unit number of file.
   FIND     I      FTAB pointer for file returned by ZOPEN.
   NVIS     I      Number of visibilities to be transfered.
   VISOFF   I      Offset in vis. rec. of first vis. rec. from BO.
   LREC     I      Number of values in a visibility record.
   NPIO     I      Number of visibilities per call to UVDISK.
                   Determines block size for tape I/O
                   0 => decide (see note above)
   BUFSZ    I      Size in bytes of the buffer.
   BUFFER   R(*)   Buffer
   BO       I      Block offset to begin transfer from (1-relative)
Output:
   NPIO     I      The max. number of visibilities which can be
                   be written or will be read per call.
   BIND     I      Pointer in BUFFER for WRITE operations.
   IERR     I      Return error code:
                   0 => OK
                   1 => file not open in FTAB
                   2 => invalid input parameter.
                   3 => I/O error
                   4 => End of file.
                   7 => buffer too small
Note: VISOFF and BO are additive.
UVINIT sets and UVDISK uses values in the FTAB:
```

```
FTAB(FIND+0) = LUN
          1  = # Bytes per I/O
          2  = # vis. records left to transfer. For double buffer
               read, 1 more I/O will have been done than shown
          3  =
          4  = Block offset for next I/O.
          5  =
          6  = byte offset of next I/O
          7  =
          8  = Current buffer #, -1 => single buffering
          9  = OPcode 1 = read, 2 = write.
         10  = Values per visibility record.
         11  = # vis. records per UVDISK call
         12  = max. # vis. per buffer.
         13  = # vis. processed in this buffer.
         14  = Buffer pointer for start of current buffer
               (in values).  Used for WRIT only; includes any
               data carried over from the last write.
         15  = Buffer pointer for call (values)
```

6.7.33 UVPGET

UVPGET determines pointers and other information from a UV CATBLK. The address relative to the start of a vis record for the real part for a given spectral channel (CHAN) and stokes parameter (ICOR) is given by NRPARM+(CHAN-1)*INCF+ABS(ICOR-ICOR0)*INCS+(IF-1)*INCIF Single dish data, i.e. randomly sampled data in the image plane, is also recognized and ILOCU and ILOCV point to the longitude like and latitude like random parameters. Also a "BEAM" random parameter may be substitued for the "BASELINE" random parameter. The data type present may be determined from the common variable TYPUVD. Two types of single dish data are recognized:
TYPUVD=1 => unprojected RA and Dec and
TYPUVD=2 => projected RA and Dec (ready for GRIDR)

```
UVPGET (IERR)
Inputs: From common /MAPHDR/ (DCAT.INC or DSEL.INC)
   CATBLK   I(256)   Catalog block
   CATH     H(256)   same as CATBLK
   CATR     R(256)   same as CATBLK
   CATD     D(128)   same as CATBLK
Output: In common /UVHDR/ (DUVH.INC)
   SOURCE   C*8      Source name.
   ILOCU    I        Offset from beginning of vis record of U
                     or longitude for single dish format data.
   ILOCV    I        Offset from beginning of vis record of V
                     or longitude for single dish format data.
   ILOCW    I        Offset from beginning of vis record of W.
   ILOCT    I                      "                     Time
   ILOCB    I                      "                   Baseline
                                                       (or beam)
   ILOCSU   I                      "                 Source id.
   ILOCFQ   I                      "                 Freq id.
   JLOCC    I        0-rel. order in data of complex values
   JLOCS    I        Order in data of Stokes' parameters.
   JLOCF    I        Order in data of Frequency.
   JLOCR    I        Order in data of RA
```

```
   JLOCD    I        Order in data of dec.
   JLOCIF   I        Order in data of IF.
   INCS     I        Increment in data for stokes (see above)
   INCF     I        Increment in data for freq. (see above)
   INCIF    I        Increment in data for IF.
   ICOR0    I        Stokes value of first value.
   NRPARM   I        Number of random parameters
   LREC     I        Length in values of a vis record.
   NVIS     I        Number of visibilities
   FREQ     D        Frequency (Hz)
   RA       D        Right ascension (1950) deg.
   DEC      D        Declination (1950) deg.
   NCOR     I        Number of correlators (Stokes' parm.)
   ISORT    C*2      Sort order 1st 2 char meaningful.
   TYPUVD   I        UV data type, 0=interferometer,
                        1=single dish unprojected,
                        2=single dish projected RA and Dec.
   IERR     I        Return error code: 0=>OK,
                        1, 2, 5, 7 : not all normal rand parms
                        2, 3, 6, 7 : not all normal axes
                        4, 5, 6, 7 : wrong bytes/value
```

6.7.34 ZCLOSE

Close the file associated with LUN removing any exclusive use state and clear the FTAB entry for the LUN.

```
ZCLOSE (LUN, FIND, IERR)
Inputs:
   LUN      I           Logical unit number
   FIND     I           Index in FTAB to file control block for LUN
Output:
   IERR     I           Error return code: 0 => no error
                           1 => close error
                           2 => file already closed in FTAB
                           3 => both errors
                           4 => erroneous LUN
```

6.7.35 ZCMPRS

ZCMPRS releases unused disk space from the end of an open disk file. AIPS "Byte" is defined as 1/2 of a integer.

```
ZCMPRS (IVOL, PNAME, LUN, LSIZE, SCRTCH, IERR)
Inputs:
   IVOL    I        volume number
   PNAME   C*48     physical file name
   LUN     I        logical unit number under which file is open.
In/Out:
   LSIZE   I        (In) desired final size in AIPS bytes
                    (Out) actual final size in AIPS bytes
Outputs:
   SCRTCH  I(256)   scratch buffer (not used under UNIX).
   IERR    I        error code: 0 => ok
                                1 => input data error
                                2 => compress error
```

6.7.36 ZCREAT

Create a disk file of a specified name and size reserving the disk space.

```
ZCREAT (IVOL, PNAME, RSIZE, MAP, ASIZE, SCRTCH, IERR)
Inputs:
   IVOL     I        Disk volume containing file
   PNAME    C*48     Physical file name
   RSIZE    I        Requested size of the file in AIPS-bytes (1/2
                     of a local integer)
   MAP      L        Is this a "map" file?
Output:
   ASIZE    I        Actual size of file in AIPS-bytes
   SCRTCH   I(256)   Scratch buffer
   IERR     I        Error return code: 0 => no error
                        1 => file already exists
                        2 => volume not found
                        3 => insufficient space
                        4 => other
                        5 => forbidden (reserved)
```

6.7.37 ZDESTR

Destroy (i.e., delete) a file. The file should already be closed.

```
ZDESTR (IVOL, PNAME, IERR)
Inputs:
   IVOL    I      Disk volume containing file, 1,2,3,...
   PNAME   C*48   Physical file name (left justified)
Output:
   IERR    I      Error return code: 0 => no error
                     1 => file not found (no message)
                     2 => device not found
                     3 => file in use
                     4 => other
```

6.7.38 ZEXPND

Increase the size of a disk file — it must be open.

```
ZEXPND (LUN, IVOL, PNAME, NREC, IERR)
Inputs:
   LUN     I      LUN of file open file
   IVOL    I      Disk volume containing file, 1,2,3,...
   PNAME   C*48   Physical file name
In/Out:
   NREC    I      # 256-integer records requested/received
Output:
   IERR    I      Error return code: 0 => no error
                     1 => input error
                     2 => expansion error
                     3 => ZEXIST error
```

6.7.39 ZFIO

Transfer one logical record between an I/O buffer and device LUN. For disk devices, the record length is always 256 local small integers and NREC is the random access record number. For non-disk devices, NREC is the number of 8-bit bytes.

```
ZFIO (OPER, LUN, FIND, NREC, BUFF, IERR)
Inputs:
   OPER   C*4      Operation code 'READ' or 'WRIT'
   LUN    I        Logical unit number
   FIND   I        Index in FTAB to file control block for LUN
   NREC   I        Random access record number (1-relative) for
                   disk transfers or number of 8-bit bytes for
                   sequential device transfers (e.g., Tektronix
                   terminals)
   BUFF   I(256)   I/O buffer
Output:
   IERR   I        Error return code: 0 => no error
                      1 => file not open
                      2 => input error
                      3 => I/O error
                      4 => end of file
```

6.7.40 ZMIO

Low level random access, large block, double buffered device I/O.

```
ZMIO (OPER, LUN, FIND, BLKNO, NBYTES, BUFF, IBUFF,
  *   IERR)
Inputs:
   OPER     C*4   Operation code 'READ' or 'WRIT'
   LUN      I     Logical unit number
   FIND     I     Index in FTAB to file control block for LUN
   BLKNO    I     Beginning virtual block number (1-relative).
                  Block size is given by NBPS in /DCHCOM/.
   NBYTES   I     Number of AIPS-bytes to transfer (an AIPS-byte is
                  1/2 a local integer).
   IBUFF    I     Buffer number to use (1 or 2)
In/out:
   BUFF     I(*)  I/O buffer
Output:
   IERR     I     Error return code: 0 => no error
                     1 => file not open
                     2 => input error
                     3 => I/O error
                     4 => end of file
```

6.7.41 ZOPEN

Open a binary disk file, line printer or tty. Message files, text files, tape devices, Tektronix devices and TV devices are NOT opened using this routine (see ZMSGOP for message files, ZTOPEN for text files, ZTPOPN for tape devices, ZTKOPN for Tektronix devices and the device specific routine for TV devices, e.g., ZM70OP).

```
ZOPEN (LUN, FIND, IVOL, PNAME, MAP, EXCL, WAIT, IERR)
Inputs:
   LUN     I      Logical unit number
   IVOL    I      Disk volume containing file, 1,2,3,...
   PNAME   C*48   Physical file name (from ZPHFIL)
   MAP     L      Is this a "map" file?
   EXCL    L      Exclusive use requested?
   WAIT    L      Wait for exclusive use?
Output:
   FIND    I      Index in FTAB to file control block for LUN
   IERR    I      Error return code: 0 => no error
                     1 => LUN already in use
                     2 => file not found
                     3 => volume/logical not found
                     4 => exclusive use denied
                     5 => no room for LUN in FTAB
                     6 => other open errors
```

6.7.42 ZPHFIL

Construct a physical file name in PNAM from TYPE, IVOL, NSEQ, and IVER - either for public data files or user-specific files.

```
ZPHFIL (TYPE, IVOL, NSEQ, IVER, PNAM, IERR)
Inputs:
   TYPE    C*2   Type of file: e.g. 'MA' for map file
   IVOL    I     Number of the disk volume to be used (1-15)
   NSEQ    I     Sequence number (000-4095)
   IVER    I     Version number (00-255)
Outputs:
   PNAM    C*48  physical file name, left justified
   IERR    I     Error return code: 0 = good return.   1 = error.

Example: If TYPE='MA', IVOL=7, AIPSVER=C, NSEQ=321, IVER=99,
         NLUSER=762 then
            PNAME='DA07:MAC14163;1'      for public data or
            PNAME='DA07:MAC14163.2FA;1'  for private data
         where 321 = 141 base 16, 99 = 63 base 16, 762 = 2FA base 16

TYPE = 'MT' leads to special name for tapes
TYPE = 'TK' leads to special name for TEK4012 plotter CRT
TYPE = 'TV' leads to special name for TV device
TYPE = 'ME' leads to special logical for POPS memory files
```

6.7.43 ZTCLOS

Close the text file and clear the FTAB entry associated with LUN.

```
ZTCLOS (LUN, FIND, IERR)
Inputs:
   LUN      I   Logical unit number
   FIND     I   Index in FTAB for LUN
Output:
```

```
   IERR     I   Error return code: 0 => no error
                   1 => close error
                   2 => file already closed in FTAB
                   3 => both errors
                   4 => erroneous LUN
```

6.7.44 ZTOPEN

Open a text file - logical area, version, member name as arguments

```
ZTOPEN (LUN, FIND, IVOL, PNAME, MNAME, VERSON, WAIT, IERR)
Inputs:
   LUN      I      Logical unit number
   IVOL     I      Disk volume containing file, (not used)
   PNAME    C*48   Physical file name, only used to determine file
                   type or logical area
   MNAME    C*8    Text file name
   VERSON   C*48   Logical name for directory or version of
                   directory to search (for file-specific
                   directories)
   WAIT     L      T => wait until file is available (not used)
Outputs:
   FIND     I      Index in FTAB for LUN
   IERR     I      Error return code: 0 => no error
                      1 => LUN already in use
                      2 => file not found
                      3 => volume not found
                      4 => file locked
                      5 => no room for LUN in FTAB
                      6 => other open errors
```

6.7.45 ZTREAD

Read the next sequential 80-character card image from a text file.

```
ZTREAD (LUN, FIND, RBUFF, IERR)
Inputs:
   LUN     I      Logical unit number
   FIND    I      Index in FTAB for LUN
Output:
   RBUFF   C*80   I/O buffer for card image
   IERR    I      Error return code: 0 => no error
                     1 => file not open
                     2 => end of file
                     4 => other I/O error
```

6.7.46 ZUVPAK

Routine to pack uv data with magic value blanking. One AIPS logical uv data record is processed at a time.

```
ZUVPAK (NCORR, VISIN, WTSCL, VISOUT)
Inputs:
   NCORR  I       Number of correlator values in data
   VISIN  R(3,*)  Unpacked uv data as real, imag and weight per
```

```
                  correlator.
Output:
   WTSCL  R(2)    "Weight" and "scale" random parameters for the
                  packed record.
   VISOUT R(*)    Packed visibility data with local magic value
                  blanking.
```

6.7.47 ZUVXPN

Routine to expand packed uv data to unpacked form. One AIPS logical uv data record is processed at a time.

```
ZUVXPN (NCORR, VISIN, WTSCL, VISOUT)
 Inputs:
  NCORR  I        Number of correlator values in data
  VISIN  R(*)     Packed visibility data with local magic value
                  blanking.
  WTSCL  R(*)    "Weight" and "scale" random parameters for the
                  packed record.
 Output:
  VISOUT  R(3,*)  Unpacked uv data as real, imag and weight per
                  correlator.
```

6.7.48 ZTXCLS

Close the text file and clear the FTAB entry associated with LUN.

```
ZTXCLS (LUN, FIND, IERR)
Inputs:
   LUN    I   Logical unit number
   FIND   I   Index in FTAB for LUN
Output:
   IERR   I   Error return code: 0 => no error
                 1 => close error
                 2 => file already closed in FTAB
                 3 => both errors
                 4 => inputs error
```

6.7.49 ZTXIO

Read/write the next sequential line from/to a text file.

```
ZTXIO (OPER, LUN, FIND, LINE, IERR)
Inputs:
   OPER   C*4     Operation code ('READ' or 'WRIT')
   LUN    I       Logical unit number
   FIND   I       Index in FTAB for LUN
Input/output:
   LINE   C*(*)   Line of text.  For WRIT, ZTXIO writes the full
                  string including any trailing blanks.  Use ITRIM
                  and substring notation in the call if you desire
                  only up to the last non-blank (which is usually
                  preferable!).  On READ, adequate size must be
                  declared in calling routine.
```

```
Output:
   IERR   I       Error return code: 0 => no error
                     1 => file not open
                     2 => end of file
                     3 => input error
                     4 => other I/O error
```

6.7.50 ZTXOPN

Open a text file.

```
ZTXOPN (OPCODE, LUN, FIND, OUTFIL, APPEND, IERR)
Inputs:
   OPCODE   C*4    Open for 'READ' or 'WRIT'
   LUN      I      Logical unit number
   OUTFIL   C*48   Physical file name
   APPEND   L      If true append new text to end of old file.
                      (OPCODE='WRIT' only).
Outputs:
   FIND     I      Index in FTAB for LUN
   IERR     I      Error return code: 0 => no error
                      1 => error in inputs
                      2 => LUN already in use
                      3 => no room for LUN in FTAB
                      4 => trouble translating logical
                      5 => file already exists
                      6 => open error
```

6.7.51 ZUVPAK

Routine to pack uv data with magic value blanking. One AIPS logical uv data record is processed at a time.

```
ZUVPAK (NCORR, VISIN, WTSCL, VISOUT)
Inputs:
   NCORR  I       Number of correlator values in data
   VISIN  R(3,*)  Unpacked uv data as real, imag and weight per
                  correlator.
Output:
   WTSCL  R(2)    "Weight" and "scale" random parameters for the
                  packed record.
   VISOUT R(*)    Packed visibility data with local magic value
                  blanking.
```

6.7.52 ZUVXPN

Routine to expand packed uv data to unpacked form. One AIPS logical uv data record is processed at a time.

```
ZUVXPN (NCORR, VISIN, WTSCL, VISOUT)
Inputs:
   NCORR  I        Number of correlator values in data
   VISIN  R(*)     Packed visibility data with local magic value
                   blanking.
   WTSCL  R(*)    "Weight" and "scale" random parameters for the
```

```
                   packed record.
Output:
   VISOUT  R(3,*)  Unpacked uv data as real, imag and weight per
                   correlator.
```

6.7.53 ZWAIT

Wait until an asynchronous I/O operation completes.

```
ZWAIT (LUN, FIND, IBUFF, IERR)
Inputs:
   LUN     I   Logical unit number
   FIND    I   Index in FTAB to file control block for LUN
   IBUFF   I   Buffer # to wait for (1 or 2)
Output:
   IERR    I   Error return code: 0 => no error
                  1 => LUN not open in FTAB
                  2 => error in inputs
                  3 => I/O error
                  4 => end of file
                  7 => wait service error
```

THIS PAGE DELIBERATELY BLANK