Going AIPS:
A Programmer's Guide to the NRAO Astronomical Image Processing System

Version 15 April 90

VOLUME 2

ABSTRACT

This manual is designed for persons wishing to write programs using the NRAO Astronomical Image Processing System (AIPS). It should be useful for a wide range of applications, from making minor changes in existing programs to writing major new applications routines. All basic aspects of AIPS programming are dealt with in some detail.

AIPS programmers:

Bill Cotton	— Image construction and applications routines
Phil Diamond	— Spectroscopy
Chris Flatters	— Polarization and VLBI
Eric Greisen	— TVs and AIPS
Kerry Hilldrup	— UNIX and Cray COS implementations
Gareth Hunt	— Data formats
Bill Junor	— VLBI
Glen Langston	— Imaging
Nancy Maddalena	— LaTeX conversion
Dean Schlemmer	— AIPS software administrator

Contents

Chapter 9

Devices

9.1 Overview

Programs in the AIPS system occasionally need to talk to peripheral devices. This chapter discusses such devices, other than disk drives, terminals, TV displays, array processors, and plotters, which are covered elsewhere. Many of these devices have their own I/O routines, but some may also use the same routines as are used for disk I/O. In general, the latter have to branch to special code for each device and we are gradually getting away from such usage. The details of the call sequence for the relevant routines discussed in this chapter are given at the end of the chapter.

9.2 Tape Drives

Tapes are used in AIPS primarily for long term storage of data, images or text files. The principal differences in the AIPS system between use of tape and disk is that tapes, by their physical nature, are sequential access devices and the physical block size of data depends on the program writing the tape. In addition, AIPS batch jobs are forbidden to talk to tape drives.

The usual problems of Fortran I/O apply to tapes, i.e., it is not predictable from one machine and/or operating system to another. For this reason, standard AIPS programs do not use Fortran I/O for tapes. Also, some versions of Fortran cannot read or write some file structures, such as those containing variable length, blocked, unspanned records. All direct operations to tape have their own AIPS Z routines: ZTAPE to position tapes, ZTPOPN to open tape devices, ZTPCLS to close tape devices, ZTPMIO to read/write on tapes, and ZTPWAT to wait for I/O completion and return the number of bytes read/written. In most of AIPS, a "byte" means one-half of a local integer. On some machines, this is 16 bits (e.g., VAX VMS), but on others it can be 32 bits (e.g., Crays). However, with tapes, AIPS usually uses "byte" to mean 8 bits since tapes are for talking to the "outside world." In the tape area, all descriptions of call sequences must specify this distinction clearly.

Since AIPS tasks work directly from I/O buffers, a program using tape must take account of the details of the way data is written on tape. One exception to this is writing variable length, blocked, but unspanned records; such records may be assembled and written using the AIPS utility routine VBOUT. All tape I/O operations or tape-like I/O to the external world should now go through the routine TAPIO.

9.2.1 Opening Tape Files (TAPIO)

Opening of a tape file may be done implicitly by the initial read or write call to TAPIO (or VBOUT). Frequently, however, it is necessary to position the tape after opening the file. In this case, TAPIO may be called with opcode 'OPRD' (open for read) or 'OPWT' (open for write) before using ZTAPE to position the tape. A more detailed description of TAPIO is given later and the call sequence is described at the end of this chapter.

9.2.2 Positioning Tapes

In AIPS, a tape must be mounted before it can be opened or any I/O performed to it. This operation (and dismounts) are done with ZTAPE. However, by convention, mounting and dismounting are done only by the AIPS program itself. Once the file has been opened, the tape may be positioned, or EOFs may be written using ZTAPE. Details of the call sequence to are given at the end of this chapter. The following list gives the opcodes recognized by ZTAPE.

1. 'ADVF' = advance file marks
2. 'ADVR' = advance records
3. 'BAKF' = backspace file marks.
4. 'BAKR' = backspace records.
5. 'DMNT' = dismount tape.
6. 'MONT' = mount tape.
7. 'REWI' = rewind the tape on unit LUN
8. 'WEOF' = write end of file on unit LUN: writes 4 EOFs, positions tape after first one
9. 'MEOF' = write 4 EOF marks on tape, position tape before the first one

9.2.3 I/O to Tape Files (TAPIO)

All tape I/O goes through the routine TAPIO. If the tape file is not open on the first I/O call, then it will be opened automatically. When writing, the buffer will be flushed when TAPIO is called with opcodes of 'FLSH' or 'CLOS'. After tape I/O has begun, should be used for any 'BAKF' (back to the beginning of the current file) operation; all other tape positioning operations should be done by ZTAPE.

Much of the description of the tape file to be read or written is given in the array FDVEC passed to TAPIO. Some bookkeeping information is kept in FDVEC, as well as in the FTAB, so it is important to protect the integrity of this array while the tape file is open. Many parameters passed through this array can be defaulted, so it is best to zero-fill the array before filling in the parameters which are not being defaulted. The contents of the FDVEC are described in the following:

```
FDVEC(40)  I*2  File descriptor vector.

      1 = LUN to use, set before first call.
          31 to 30+NTAPED => tape, else disk.
      2 = Logical record length in 8-bit bytes
      3 = Buffer size in 8-bit bytes
      5 = volume number (disk or tape)
      6 = blocking factor (0=>1), value returned on read
    7-18 = File name for disk files (48 char. hollerith)
     19 = 0 if fixed length, 1 if variable length
     20 = Max. number of logical records to process
          0 => infinity
  21-29   Reserved for future use
 The following are used by TAPIO:
     30 = FTAB pointer
     31 = Number of logical records left to process
          (negative => ignore)
     32 = Number bytes read or written
```

VBOUT

The utility routine VBOUT will collect variable length records and block them, unspanned, into IBM format physical blocks up to 4008 (8-bit) bytes long. The tape may be opened explicitly with TAPIO (OP = 'OPWT') or implicitly with the first call to VBOUT. In either case, the array FDVEC must be properly prepared before opening the tape. The principal use of this routine is to write VLA "EXPORT" format tapes. Details of the call sequence, as well as other important usage notes, are found at the end of this chapter.

Sequential External Files

The AIPS tape reading and writing tasks can read or write FITS format files from or to disk files. These disk files are considered to be byte streams. All I/O to these files is through the routine TAPIO.

9.2.4 Tape Data Structure

In order to make efficient use of tape storage, a number of logical records may be grouped into a single physical record. In general, these logical records may be fixed or variable length and may or may not span physical blocks. In addition, logical records may be formatted (text, usually ASCII) or binary. Such details need to be determined before attempting to read or write such files. Fixed length logical records are packed into physical records as defined by the record size and block length. Since the order and size of these records is well defined there is no need for additional control information. On write, TAPIO must be told the blocking factor (FDVEC(6)) and, on read, TAPIO automatically determines the actual blocking factor. For variable length logical records, control bytes are added to the record to determine the boundaries of logical records. Unfortunately, the details of the of variable length record structure vary from computer to computer and from operating system to operating system. If you wish to read or write one of these tapes, you have to find the details of the formats for the machines in question. The format used by AIPS for Export-format tapes is that defined by IBM. In this format, four bytes are added to the start of each tape record to define the length of the record in bytes (first 16 bits, the other 16 are zero). In addition, four more bytes are added to the start of each logical record to define the length of that record including the control information in bytes (first 16 bits, the other 16 are zero). VBOUT handles these control bytes for the program and TAPIO depends on the first 2 bytes to control the length of the I/O operation.

9.2.5 Tape Data Formats

In AIPS, it is the convention to write all tape data in FITS standard form. This means all characters are unsigned ASCII. All integers are twos-complement in 16 bits or 32 bits with the sign and high order byte first. Unsigned 8-bit integers are also allowed. All floating point binary numbers follow the IEEE conventions.

As a result of this convention, tape reading/writing is somewhat complicated in AIPS. However, there are Z routines to help:

1. ZC8CL converts ASCII characters to local character form.
2. ZI8IL converts 8-bit unsigned integers to local AIPS bytes (half integers).
3. ZI16IL converts 16-bit standard integers to local integers.
4. ZI32IL converts 32-bit standard integers to local long integers.
5. ZR32RL converts 32-bit IEEE to local floating point format.
6. ZR64RL converts 64-bit IEEE to local double precision floating point.
7. ZCLC8 converts local characters to standard ASCII.
8. ZILI16 converts local integers to standard 16-bit integers.
9. ZILI32 converts local long integers to standard 32-bit integers.

10. ZRLR32 converts local floating point to 32-bit IEEE floating.

11. ZRLR64 converts local double precision floating point to 64-bit IEEE floating format.

All of these routines can convert any number of values in one buffer to the output form in another buffer, which can be at the same address. Before FITS tapes were blocked, it was common practice to do the conversion in place even though the lengths of the input array and output array are different (on some machines anyway). However, this practice is now too dangerous to be done inside buffers used for potentially blocked tapes. VBOUT calls ZILI16 for the programmer, which helps as long as everything is in integers. If some datum is not, it must be translated to standard and then back as if it were local integers (see VBOUT precursor remarks below).

9.3 Graphics Displays

The graphics devices currently supported in AIPS fall into three categories: TV display devices, such as the IIS, hardcopy devices, such as the Versatec printer/plotter and QMS Lasergraphix printer, and interactive graphics terminals, such as the Tektronix 4012. This section deals with the Tektronix type graphics terminals. The other devices are discussed in the chapter on plotting. A graphics terminal can be used in two major modes: as a temporary display device, or as an interactive graphics device. When used as a temporary display device, a task will read graphics commands from a plot file, convert these device-independent commands to the form needed by the device, and finally write to the device. The AIPS task that does this is TKPL. A programmer, wishing to write a task to interpret a plot file for another type of graphics terminal, would start with TKPL and convert the routines TKDVEC, TKCHAR, and TKCLR to send the proper commands to the device.

When using a graphics terminal in the interactive mode, the programmer probably will go straight from the data file to the graphics terminal without going through a plot file. In general, an interactive task or verb will open the display device, display the data, activate the cursor, read the cursor position in the absolute device coordinates, convert these coordinates into more useful units, and then perform some useful function with the converted units, such as display them. Current AIPS use of graphics is quite primitive. In the future, we will probably convert to use of the X-windows graphics system, which may invalidate most of the following discussion.

9.3.1 Opening the Graphics Terminal

The graphics terminal is opened as a non-map file using ZTKOPN. AIPS logical unit 7 is reserved for this device type, and should be used in the call to ZTKOPN. On a VAX, each AIPS is assigned a graphics terminal on start up according to a set of logical names. Similar strategies are played in other implementations. Thus, ZTKOPN just opens the user's assigned device to the local program.

9.3.2 Writing to the Graphics Terminal

The include file INCS:DTVC.INC initialized by ZDCHIN contains two useful parameters for graphics. These are MAXXTK(2), which contains the maximum X and Y values in device units (for the Tektronix 4012, these values range from 1 to 1024 for X and 1 to 780 for Y) and CSIZTK(2) , which contains the X,Y character sizes in plot units (14,22 for Tektronics 4012). In include INCS:DTKS.INC, the graphics buffer size, TKSIZE, should be set to 20. The current position in use in the buffer, TKPOS, should be set to zero. Scale factors SCALEX and SCALEY and offsets RX0 and RY0 must be calculated and assigned. If a subroutine is told to scale a value, then the X value in absolute device units will be equal to

```
SCALEX * value_input_for_X + RX0.
```

Usually the first thing a programmer will want to do when writing to the terminal is to clear the screen. This can be done with subroutine TKCLR.

Setting the beginning of the line (sometimes called drawing a dark vector) and drawing lines from the current position to a new position (a bright vector) are done with routine TEKVEC. TEKVEC is given an

X and Y position and a control code which tells it if it should draw a dark vector or a bright vector, and if it should consider X and Y to be in absolute device units or if the values should be scaled. TEKVEC will automatically interpolate vectors that run off the plot and write the buffer when it fills up.

Characters can be written to a Tektronix terminal by calling routine TKCHAR. TKCHAR allows the programmer to write characters either horizontally or vertically. TKCHAR uses the hardware character generator in the Tektronix, so the characters only come in one size. Choosing the starting position of the characters involves a combination of TEKVEC and TKCHAR. First, a vector position on the plot is chosen by calling TEKVEC with the "dark vector" option. Then an offset from the vector position in character sizes is chosen by use of the DCX and DCY parameters in TKCHAR. Programmers who need a character generator that can be adapted to a graphics terminal can find one in task PRTPL.

Before closing the graphics terminal, the programmer should write any remaining buffers to the graphics device by calling TEKFLS.

9.3.3 Activating and Reading the Cursor

Subroutine TKCURS will activate the cursor on the Tektronix 4012 and wait for a response from the 4012 keyboard. After the user positions the cursor and presses any key, the cursor will disappear and TKCURS will return the last coordinate position in absolute Tektronix units. The programmer will probably have to convert this position into plot coordinates by using information in the image catalog.

9.3.4 Updating the Image Catalog

The image catalog should be updated when an image is written to the graphics terminal. This is essential when one task (or verb) writes an image to the device, and another task (or verb) needs information about the plot on the screen. For example, task TKPL can be used to display a contour map on the terminal, and verb TKPOS can be used to print map coordinate values at selected positions on the plot. TKPOS uses information in the image header to convert an absolute Tektronix cursor position into the map axis units such as RA and DEC. The routine TKCATL can be used to set up the image catalog for the graphics terminal. See the chapter on catalogs for a detailed description of the image catalog and the example below for making a minimum image catalog entry.

9.3.5 An Example

This example code shows how to open the graphics terminal, clear the screen, draw a box, and write some text in the center of the box. Opening the map, getting parameters from AIPS, etc., are not shown. In a non-trivial example, calculating the scaling parameters and updating the image catalog would be much more involved.

```
      INTEGER   ITKLUN, ITKIND, IERR, TKSIZE, TKPOS, IPOS,
     *   IDRAW, NCHAR, IHORZ, IPLANE, BUFFER(256), VOL, CNO,
     *   IX, IY
      CHARACTER LINE*80
      REAL      BLCX, BLCY, TRCX, TRCY, CENTER, DCX, DCY
           ...
      INCLUDE 'INCS:DHDR.INC'
      INCLUDE 'INCS:DDCH.INC'
      INCLUDE 'INCS:DTKS.INC'
      INCLUDE 'INCS:DCAT.INC'

      .
      .
      .
C                                       Open the Tektronix device.
      ITKLUN = 7
      CALL ZTKOPN (ITKLUN, ITKIND, IERR)
```

```
      IF (IERR.NE.0) GO TO 900
C                                     Set variables in common.
      TKSIZE = 20
      TKPOS = 0
C                                     Make screen be 100 by 100
C                                     units.
      SCALEX = MAXXTK(1) / 100.0
      SCALEY = MAXXTK(2) / 100.0
      RX0 = 0.0
      RY0 = 0.0
C                                     Clear screen.
      CALL TKCLR (ITKIND, IERR)
      IF (IERR.NE.0) GO TO 900
C                                     Set corners
      BLCX = 25.0
      BLCY = 25.0
      TRCX = 75.0
      TRCY = 75.0
C                                     1 is the code for scale X
C                                     and Y and position vector.
      IPOS = 1
C                                     2 is the code for scale X
C                                     and Y and draw vector.
      IDRAW = 2
C                                     Draw a box.
      CALL TEKVEC (BLCX, BLCY, IPOS, ITKIND, IERR)
      CALL TEKVEC (BLCX, TRCY, IDRAW, ITKIND, IERR)
      CALL TEKVEC (TRCX, TRCY, IDRAW, ITKIND, IERR)
      CALL TEKVEC (TRCX, BLCY, IDRAW, ITKIND, IERR)
      CALL TEKVEC (BLCX, BLCY, IDRAW, ITKIND, IERR)
      IF (IERR.NE.0) GO TO 900
C                                     Write some characters in
C                                     the center of the box.
      NCHAR = 14
      LINE = 'This is a test'
C                                     Position at center.
      CENTER = 50.0
      CALL TEKVEC (CENTER, CENTER, IPOS, ITKIND, IERR)
      IF (IERR.NE.0) GO TO 900
C                                     Compute offset to start
C                                     writing characters.
      DCX = - NCHAR / 2.0
      DCY = - 0.5
      IHORZ = 0
C                                     Write message
      CALL TKCHAR (NCHAR, IHORZ, DCX, DCY, LINE, ITKIND, IERR)
      IF (IERR.NE.0) GO TO 900
C                                     Write any remaining buffer to
C                                     screen.
      CALL TEKFLS (ITKIND, IERR)
C                                     Update image catalog although
C                                     for this example plot has no
C                                     relation to map.
C                                     CATBLK, VOL and CNO were
```

```
C                                       found when map was opened.
      CATBLK(IIVOL) = VOL
      CATBLK(IICNO) = CNO
C                                       Set plot type to MISC
      CATBLK(IIPLT) = 1
      CALL TKCATL ('WRIT', IX, IY, CATBLK, IERR)
C                                       Close graphics terminal.
      CALL ZTKCLS (ITKLUN, ITKIND, IERR)
      .
      .
      .
```

9.4 INCLUDEs

9.4.1 DTKS.INC

```
C                                                             Include DTKS.
      REAL      TKBUFF(20), SCALEX, SCALEY, RX0, RY0, RXL, RYL
      INTEGER   TKPOS, TKSIZE
      COMMON /TKSPCL/ TKBUFF, SCALEX, SCALEY, RX0, RY0, RXL, RYL,
     *   TKPOS, TKSIZE
C                                                             End DTKS.
```

9.4.2 DDCH.INC

```
      INC /DDCH.INC
```

9.5 Routines

9.5.1 TAPIO

TAPIO is primarily for reading and writing tapes but will also work for disks for FITS files etc. Disk files will be created and expanded as necessary. When a disk file is being written it is compressed to its actual size when it is closed. NOTE: TAPIO WORKS IN REAL (8-BIT) BYTES, NOT THE AIPS HALF-INTEGER "BYTES". Usage notes:

1. Zero fill FDVEC before filling in relevant values.
2. Opening the file. If TAPIO determines that the file is not open it will do so. For disk files being opened for write the file will be created. Once the file is open the file descriptor vector FDVEC must be used in each call.
3. Initialization. TAPIO initializes the I/O using the values in FDVEC when it opens the file. If OP='OPxx' the file is created/opened but I/O is not initialized; this allows positioning tapes before the actual I/O starts. 'OPRD' means open for read, 'OPWT' means open for write; 'OPWT' will cause the output file to be created if the output is to disk.
4. Re-initialization. If OP='BAKF' the file is repositioned at the beginning of the current file, I/O will be reinitialized at the next read call. This operation is only relevant to reading files.
5. Closing the file. The file may be closed with a call with opcode 'CLOS'.

6. Writing variable length records. If variable length records are to be read or written FDVEC(19) should be set to 1. On WRITE, the actual record size is assumed to be given by the first I word of the record. TAPIO will not handle blocked, variable length records.

7. Limited number of logical records. If only a fixed number of logical records are to be processed before reinitialization put the number of such records in FDVEC(20). After this number of transfers the I/O will be left in an uninitialized state and on the next call will be reinitialized (e.g. can change logical record length). DO NOT USE THIS WITH BLOCKING HOWEVER, the last block will not be the correct length.

Tapes: $31 \leq \text{LUN} \leq 30 + \text{NTAPED}$

Disks: any other LUN
for FITS format only (no blocking).
TAPIO requires that FDVEC(2) = 2880, FDVEC(31) = 0 on input.
TAPIO sets FDVEC(6) = 1.
The desired file name must be in FDVEC(7-30) as a HOLLERITH string.

```
TAPIO (OP, FDVEC, BUFF, IBIND, IERR)
Inputs:
   OP     C*4     Operation code: 'READ','WRIT','CLOS','BAKF',
                                  'OPRD','OPWT' open for rd,wrt
                                  'FLSH' flush write buffer, Weof
Input/Output:
   FDVEC  I(50)   File descriptor vector.
                  1 = LUN to use, set before first call.
                      31 to 30+NTAPED => tape, else disk.
                  2 = Logical record length in bytes (8-bit)
                  3 = Buffer size in 8-bit bytes
                  5 = volumn number (disk or tape)
                  6 = blocking factor (0=>1). Value returned on read
                7-30 = File name for disk files (48 HOLLERITH char)
                  31 = 0 if fixed length, 1 if variable length
                  32 = Max. number of logical records to process
                       0 => infinity
               33-39   Reserved for future use
              The following are used by TAPIO:
                  40 = FTAB pointer
                  41 = Number of logical records left to process
                       (negative => ignore)
                  42 = LBYTES - number of bytes read or written.
               43-50 = reserved for future use
   BUFF   R(*)    Buffer for I/O must be large enough for the largest
                  transfer rounded up to the next larger number of
                  disk blocks.
Outputs:
   IBIND  I       The location in BUFF of the start of the next
                  record. Before the first write call this should
                  be set to 1 to determine where to start filling
                  BUFF. Note: IBIND points to the address in the
                  I   array irregardless of the actual data type.
```

```
   IERR    I      Error return: 0 => ok
                                1 => error creating file
                                2 => input error
                                3 => i/o error on initialize
                                4 => end of file
                                5 => beginning of medium
                                6 => end of medium
                                7 => Buffer too small
                                8 => error opening file.
                                9 => error expanding
Usage notes: The first 16 words in each FTAB entry
contain a user table to handle double buffer i/o,  the rest
contain system-dependent I/O tables.
FTAB user table   entries, with offsets from the FIND pointer are:
   FTAB + 0 =>  LUN using this entry
          1 =>  Number of blocks to extend the file when full
          2 =>  Number of 8-bit bytes in a logical record
          3 =>  Number of disk logical records in each transfer (1)
          4 =>  Which buffer half currently doing  i/o; -1 =>
                single buffer,  the other buffer half is available.
        5-6 =>  Block offset on disk file for next operation I
        7-8 =>
          9 =>  I/O opcode 0=read, 1=write
         10 =>  1 => tape, 2 => disk
         11 =>  Number of logical records per physical
         12 =>  Number of logical records done for this physical.
         13 =>  1 => I/O active, else inactive (not initialized).
         14 =>  number bytes last read/write to buffer 1
         15 =>  number bytes last read/write to buffer 2
```

9.5.2 TEKFLS

Will write the output buffer TKBUFF to the TEKTRONIX 4012.

```
TEKFLS (FIND, IERR)
Input:
   FIND     I   FTAB position assigned to TEK 4012.
Output:
   IERR     I   error flag. 0=ok,  .GT. 1=write error from ZFIO
```

9.5.3 TEKVEC

This routine will put control characters, and X and Y coordinates into the TEKTRONIX output buffer.

```
TEKVEC (XX, YY, IN, FIND, IERR)
Inputs:
   XX    I   X coordinate value.
   YY    I   Y coordinate value.
   IN    I   control value:
             1 = Scale XX and YY and preceed coordinates
                 by 'write dark vector' control character
             2 = Scale XX and YY, put in buffer.
             3 = XX and YY are not scaled, 'write dark vector'
                 control character is put into the buffer.
```

```
              4 = no scale, write vector
   FIND  I    FTAB position of TEKTRONIX device.
Output:
   IERR  I    error code, 0=ok, 1=write error.
Common:
   DTKS.INC  in/out   TKBUFF, TKPOS, RXL, RYL
```

9.5.4 TKCATL

Read, write, init the Graphics image catalog

```
TKCATL (OPER, IX, IY, CATBLK, IERR)
Inputs:
   OPER     C*4      'INIT' - zero catalog block for current TEK #
                     'READ' - read catalog block for current TEK #
                     'WRIT' - write catalog block for current TEK #
   IX       I        X pixel position (check vs CATBLK on READ)
   IY       I        Y pixel position (check vs CATBLK on READ)
In/out:
   CATBLK   I(256)   Image header converted for image catalog use
Output:
   IERR     I        Error return:  0 => o.k.
                        10 => access not allowed for this POPS #
                        11 => IX, IY not in image
                        > 0 => error return from ZOPEN, ZFIO
```

9.5.5 TKCHAR

Will write characters to a TEKTRONIX 4012.

```
TKCHAR (INCHAR, IANGL, DCX, DCY, TEXT, ITFIND, IERR)
Inputs:
   INCHAR   I        number of characters.
   IANGL    I        0=horizontal, other = vertical.
   DCX      R        X distance in characters from current position.
   DCY      R        Y distance in characters from current position.
   TEXT     C*(*)    packed characters.
   ITFIND   R        FTAB index of open TEK.
Outputs:
   IERR     I        error indicator. 0 = ok.
```

9.5.6 TKCLR

Will clear the screen for a Tektronix 401n.

```
TKCLR (DEVFND, IERR)
Inputs:
   DEVFND   I     FTAB index of an open device.
Output:
   IERR     I     Error code from the last I/O routine. 0=ok.
```

9.5.7 TKCURS

Will activate the cursor on the TEKTRONIX 4012 and wait for a response from the 4012 keyboard. After the response the cursor will disappear and TEKCUR will return the coordinate positions.

```
TKCURS (IFIND, IOBLK, IX, IY, IERR)
Inputs:
   IFIND   I        index into FTAB for open TEKTRONIX device.
   IOBLK   I(256)   I/O block for TEKTRONIX device.
Outputs:
   IX      I        x cursor position.
   IY      I        y cursor position.
   IERR    I        0=ok, 1=TEK write error. 2=TEK read error.
WARNING: This routine assumes a normal interface to a TEK 401n.
Thus it may not work on all CPUs.
```

9.5.8 TKDVEC

Converts TEK4012 vectors to actual commands to the TK buffer Positions are assumed to be in bounds.

```
TKDVEC (IN, X, Y, FIND, IERR)
Inputs:
   IN    I     1 => dark vector, 2 => bright vector
   X     I     X coordinate value.
   Y     I     Y coordinate value.
   FIND  I     FTAB position of TEKTRONIX device.
Output:
   IERR  I     error code, 0=ok, 1=write error.
Common:
   DTKS.INC   in/out   TKBUFF, TKPOS
```

9.5.9 VBOUT

Writes variable blocked records of data to tape. Maximum block size on the tape is 4008 bytes. Tape may be opened by TAPIO (OP='OPWT') before first call. For overlaid programs COMMON /VBCOM/ should be kept in a segment which is core-resident from the first to the last call to VBOUT. A call with N = 0 will cause all data remaining in the buffer to be written. Character data must be in ASCII as integers: i.e. call ZCLC8 followed by ZI16IL on such data before calling VBOUT.

```
VBOUT (N, IDATA, FDVEC, NUM, IERR)
Inputs:
   N      I       Number of words in array IDATA.
                  If N = 0 the buffer is flushed.
   IDATA  I       Array containing data to be written.
   FDVEC  I(50)   Field descriptor vector for TAPIO
   NUM    I       The record number to be written, must be 1 on
                  the first and only the first record in a file.
Output:
   IERR   I       Return error code 0 => OK, else TAPIO error.
```

9.5.10 ZC8CL

Convert 8-bit ASCII standard characters in a buffer to local character form

```
ZC8CL (NCHAR, NP, INBUF, OUTBUF)
Inputs:
   NCHAR    I       Number of characters to convert
   NP       I       Starting position in input buffer in units of
                    8-bit characters (1-relative)
   INBUF    R(*)    Input buffer containing 8-bit ASCII characters
Output:
   OUTBUF   C*(*)   Output buffer containing characters in local
                    form beginning in position 1
```

9.5.11 ZCLC8

Convert local characters in a buffer to standard 8-bit ASCII character form

```
ZCLC8 (NCHAR, INBUF, NP, OUTBUF)
Inputs:
   NCHAR    I       Number of characters to convert
   INBUF    C*(*)   Characters in local form
   NP       I       Starting position in output buffer in units of
                    8-bit characters (1-relative)
Output:
   OUTBUF   R(*)    Buffer containing characters in 8-bit ASCII form
```

9.5.12 ZI8IL

Convert 8-bit unsigned binary numbers to local integers. This must work even when the input and output buffers are the same.

```
ZI8IL (NVAL, NP, INB, OUTB)
Inputs:
   NVAL   I         Number of 8-bit values to convert
   NP     I         Starting position in the input buffer counting
                    from 1 in units of 8-bit values
   INB    I(*)      Input buffer
Output:
   OUTB   I(NVAL)   Output buffer
```

9.5.13 ZI16IL

Extract 16-bit, 2's complement integers from an input buffer and put them into an output buffer in local integer form. This must work even when the address of the input and output buffers is the same.

```
ZI16IL (NVAL, NP, INB, OUTB)
Inputs:
   NVAL   I         Number of 16-bit integers to extract
   NP     I         Starting position in the input buffer counting
                    from 1 in units of 16-bit integers
   INB    I*2(*)    Input buffer
Output:
   OUTB   I(NVAL)   Output buffer
```

9.5.14 ZILI16

Convert a buffer of local integers to a buffer of standard 16-bit, 2's complement integers.

```
ZILI16 (NINT, INB, NP, OUTB)
Inputs:
   NINT   I      Number of integers to convert
   INB    I(*)   Input buffer (start at index 1)
   NP     I      Starting index in the output buffer (1-relative)
                 in units of 16-bit integers
Output:
   OUTB   I(*)   Output buffer
```

9.5.15 ZI32IL

Extract 32-bit, 2's complement integers from an input buffer and put them into an output buffer in local integer form. This must work even when the address of the input and output buffers is the same. The IBM order applies to the input (i.e., the most significant part of the 32-bit integer is in the lower index of the input buffer and the least significant part is in the higher index.

```
ZI32IL (NVAL, NP, INB, OUTB)
Inputs:
   NVAL   I      # values to extract
   NP     I      Starting position in the input buffer (1-relative)
                 in units of 32-bit integers
   INB    I(*)   Input buffer
Output:
   OUTB   I(*)   Output buffer
```

9.5.16 ZILI32

Convert a buffer of local integers to a buffer of standard 32-bit, 2's complement integers. This must work even when the address of the input and output buffers is the same. The IBM order applies to the output (i.e., the most significant part of the 32-bit integer is in the lower index of the output buffer and the least significant part is in the higher index).

```
ZILI32 (NVAL, INB, NP, OUTB)
Inputs:
   NVAL   I         # integers to convert
   INB    I(*)      Input buffer (start at index 1)
   NP     I         Starting position in the output buffer
                    (1-relative) in units of 32-bit integers
Output:
   OUTB   I(NVAL)   Output buffer
```

9.5.17 ZR32RL

Converts from 32 bit IEEE floating format to local single precision.

The IEEE format is:

```
          1         2         3
01234567890123456789012345678901
seeeeeeeemmmmmmmmmmmmmmmmmmmmmm
```

where sign = -1 ** s, exponent = eee..., mantissa = 1.mmmmm..

The value is given by:

value = sign * 2 **(exp-127) * mantissa

Note: these values have a "hidden" bit and must always be normalized The IEEE nan (not a number) values are used to indicate an invalid number; a value with all bits set is recognized as a "nan".
The AIPS internal format for an invalid number is the value which has the same bit pattern as 'INDE'.
The IEEE special values (-0., +/- Infty) are not recognized.
A multiplication by a factor of 4.0 converts between VAX F and IEEE 32 bit formats.

```
ZR32RL (NVAL, NP, INB, OUTB)
Inputs:
   NVAL   I      Number of values to convert
   NP     I      First value in INB to convert
   INB    R(*)   32-bit IEEE format values
Output:
   OUTB   R(*)   Local format values ("nan" values are replaced
                 with AIPS' indefinite value = 'INDE')
```

9.5.18 ZRLR32

Converts from local single precision to 32 bit IEEE floating format.
The AIPS internal format for an invalid number is the value which has the same bit pattern as 'INDE'.
A multiplication by a factor of 4.0 converts between VAX F and IEEE 32 bit formats.

```
ZRLR32 (NVAL, NP, INB, OUTB)
Inputs:
   NVAL   I      Number of values to convert
   NP     I      First value in OUTB for result
   INB    R(*)   Local format values
Output:
   OUTB   R(*)   32-bit IEEE FORMAT values ('INDE' values are
                 replaced with "nan")
```

9.5.19 ZR64RL

Converts from 64 bit IEEE floating format data to local double precision (or corresponding 64 bit precision).
The IEEE format is:

```
          1         2         3         4         5         6
0123456789012345678901234567890123456789012345678901234567890123
seeeeeeeeeeemmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm
```

where sign = -1 ** s, exponent = eee..., mantissa = 1.mmmmm..

The value is given by:

value = sign * 2 **(exp-1023) * mantissa
Note: these values have a "hidden" bit and must always be normalized The IEEE nan (not a number) values are used to indicate an invalid number; a value with all bits set is recognized as a "nan".
The AIPS internal format for an invalid number is the value which has the same bit pattern as 'INDE '.
The IEEE special values (-0., +/- Infty) are not recognized.
A multiplication by a factor of 4.0 converts between VAX G and IEEE 64 bit formats.

```
ZR64RL (NVAL, NP, INB, OUTB)
Inputs:
   NVAL   I      Number of values to convert
   NP     I      First value in INB to convert
   INB    D(*)   64-bit IEEE format values
Output:
   OUTB   D(*)   Local format values ("nan" values are replaced
                 with AIPS' D.P. blank = 'INDE    ')
Generic version  - does IEEE and VAX G formats for 32-bit machines,
is stubbed with STOP for all others.
```

9.5.20 ZRLR64

Converts from local double precision (or corresponding 64 bit precision) to 64 bit IEEE floating format.
The AIPS internal format for an invalid number is the value which has the same bit pattern as 'INDE '.
A multiplication by a factor of 4.0 converts between VAX G and IEEE 64 bit formats.

```
ZRLR64 (NVAL, NP, INB, OUTB)
Inputs:
   NVAL   I      Number of values to convert
   NP     I      First location in OUTB for results
   INB    D(*)   Local format values
Output:
   OUTB   D(*)   64-bit IEEE format values ('INDE    ' values are
                 replaced with "nan")
```

9.5.21 ZTAPE

Performs standard tape manipulating functions.

```
ZTAPE (OP, LUN, FIND, COUNT, IERR)
Inputs:
    OP      C*4   Operation to be performed. 4 characters ASCII.
                  'ADVF' = advance file marks
                  'ADVR' = advance records
                  'BAKF' = backspace file marks.
                  'BAKR' = backspace records.
                  'DMNT' = dismount tape.
                  'MONT' = mount tape.
                  'REWI' = rewind the tape on unit LUN
```

```
                    'WEOF' = write end of file on unit LUN: writes 4
                             EOFs, positions tape after first one
                    'MEOF' = write 4 EOF marks on tape, position tape
                             before the first one
      LUN      I     logical unit number
      FIND     I     FTAB pointer.  Drive number for MOUNT/DISMOUNT.
      COUNT    I     Number of records or file marks to skip. On MOUNT
                     this value is the density.
   Outputs:
      IERR  I        Error return: 0 => ok
                        1 = File not open
                        2 = Input specification error.
                        3 = I/O error.
                        4 = End Of File
                        5 = Beginning Of Medium
                        6 = End Of Medium
```

9.5.22 ZTKCLS

Close a Tektronix device

```
   ZTKCLS (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 => Non-zero ZTKCL2 error
                       2 => Non-zero LSERCH error
                       3 => both 1 and 2
                       4 => invalid LUN
```

9.5.23 ZTKOPN

Open a Tektronix device (calls ZTKOP2 to perform the actual open).

```
   ZTKOPN (LUN, FIND, IERR)
   Inputs:
      LUN     I   Logical unit number
   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 => no such logical device
                     3 => device not found
                     4 => exclusive use denied
                     5 => no room for LUN in FTAB
                     6 => other open errors
```

9.5.24 ZTPCLS

Close the tape drive associated with LUN as well as its disk control file removing any exclusive use state and clear the corresponding FTAB entries. ZTPCL2 actually closes the tape drive and ZDACLS is called to close the disk control file. Also closes sequential type disk files via ZTPCLD.

```
ZTPCLS (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 => non-zero LSERCH error
                 3 => both 1 and 2
                 4 => invalid LUN
```

9.5.25 ZTPMIO

Low level sequential access, large record, double buffered tape device I/O.

```
ZTPMIO (OPER, LUN, FIND, 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
   NBYTES   I      Number of 8-bit bytes to transfer
   BUFF     I(*)   I/O buffer
   IBUFF    I      Buffer number to use (1 or 2)
Output:
   IERR     I      Error return code: 0 => no error
                      1 => file not open
                      2 => input error
                      3 => I/O error
                      4 => end of file (no messages)
```

9.5.26 ZTPOPN

Open a tape drive (as well as its corresponding disk control file) for sequential, "map" (double buffered, asynchronous) I/O or open a pseudo-tape sequential disk file. Exclusive use and wait to open are assumed. Uses a 'TP' disk "lock" file for real tapes.

```
ZTPOPN (LUN, FIND, IVOL, PNAME, OPER, IERR)
Inputs:
   LUN      I       Logical unit number (30 < LUN <= 30 + NTAPED
                       => tape, else disk)
   IVOL     I       Tape drive or disk volume containing file
   PNAME    C*48    tape disk physical file name
   OPER     C*4     'READ' => read only or 'WRIT' => read/write
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 not found
                    4 => exclusive use denied
                    5 => no room for LUN in FTAB
                    6 => other open errors
```

9.5.27 ZTPWAT

Wait until an asynchronous tape or sequential pseudo-tape disk file I/O operation completes.

```
ZTPWAT (LUN, FIND, IBUFF, LBYTES, 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:
   LBYTES   I   Number 8-bit bytes read/written (+1 if tape tape
                record longer than requested)
   IERR     I   Error return code: 0 => no error
                   1 => LUN not open in FTAB
                   3 => I/O error
                   4 => end of file
                   7 => wait service error
```

Chapter 10

Using the TV Display

10.1 Overview

The most useful implementations of the AIPS system include one or more computer peripheral devices capable of displaying images with multiple levels of grey and/or color. We refer to such devices as TV displays since most are implemented via large binary memories and standard television monitors. The main program AIPS and some tasks (e.g., TVFLG) use the TV display as an interactive input, as well as display, device. Other tasks (e.g., UVMAP, MX, APCLN) use the TV display simply to show the stages of the data processing. All use of the TV is optional and the AIPS system will run without such a device. The number of TV displays in the local system is parameterized (under control of the stand alone program SETPAR) and all programs are told which TV display (if any) is assigned to the current user.

10.1.1 Why Use (or not use) the TV Display

There are numerous reasons to use the TV display in writing AIPS routines. Grey scale images provide a realistic view of image data allowing the eye to integrate over noisy regions and to separate closely spaced features. Contour images require much more elaborate software to generate and they make unreasonably definitive assertions about the intensity levels. The TV may be used to display intermediate results which are never stored on disk. And the TV may be used to interact with the user in a very wide variety of ways. Current interactive usages include modification of the black and white transfer function, modification of pseudo coloring, selection of features of interest, selection of subimage corners, dynamic, multi-image displays, and communication to the task of simple information. The last is used primarily to tell iterative tasks that they may stop at the current iteration.

Despite these desirable features, an AIPS programmer should not put the TV in a task unless it is truly useful. A TV option requires some, potentially considerable extra coding effort and, during execution, some significant extra real and CPU time. Many TV devices also require a high rate of I/O in order to load an image and, especially, to interact with the user. If an algorithm is based on the TV display, then it will not be available at those AIPS sites which do not have one. Although TV displays can function as graphics devices, many of them are very slow in that mode. Finally, tasks which use the TV will interfere with the interactive AIPS user's other uses of the display by replacing current images in the TV memory or modifying the zoom, scroll, transfer functions, et al.

10.1.2 The AIPS Model of a TV Display Device

As AIPS was being designed, it was realized that there was already a wide variety of TV display devices on the market and that the market would not hold still. The NRAO initially purchased two International Imaging Systems (IIS) Model 70/E displays. However, that company changed rapidly to Model 70/F and now no longer sells the Model 70. Instead, it now markets, among other things, a Model 75 and a less expensive IVAS model. The NRAO acquired an IVAS in 1986. Support for work stations with imaging capability was added early in 1988 after SUN loaned a SUN 3/110 to the NRAO. Our initial choices undoubtedly color our

image of what a TV display device does and how it does it. Nonetheless, we have attempted to design the code to be very general and to account for the range of options available on individual models of display and for the range of different manufacturers.

We regard the TV display as being a computer peripheral device which accepts the basic I/O operations of open, close, initialize, read, and write. Special Z routines are provided in AIPS since we do not assume that these I/O operations are identical, for all TVs and host operating systems, to those for disks, tapes, or Fortran devices. We assume that the TV display may be subdivided logically into a variety of sub-units which control the various functions of the display. Special libraries of subroutines, subdivided by model of TV display, are provided for communicating to these sub-units. These subroutines are called "Y routines" because all of them have names beginning with the letter Y. The NRAO has, at this time, developed the Y routines for IIS Models 70/E, 70/F, and IVAS and for SUN workstations using the SUNView software system. In addition, we store, distribute, and attempt to maintain sets of Y routines developed by other institutions for other models of displays. At the moment, we have Y routines for DeAnza, developed by Walter Jaffe at the STScI, for IIS Model 75, developed by IIS, and for Comtal Vision 1/20, developed by Andy Lubenow at the University of Illinois. We expect (hope) to receive more sets of Y routines for distribution in the near future. We have developed and support a set of Y routines and the other software needed to operate a TV device "remotely". These so-called "virtual-television" routines connect the computer to the remote TV device operating on some cooperating AIPS computer, presumably located near the user.

AIPS also uses, at both the Y and non-Y programming levels, a TV device parameter common. This common is initialized by a Y routine (YTVCIN) and is maintained via a disk file and a stand alone program (SETTVP). The common contains both fundamental parameters (i.e., number of memories, display size, maximum intensity, maximum zoom, etc.) and parameters describing the current state of the TV (i.e., which planes are on, current zoom and scroll, etc.).

In order to provide the full functionality of the basic AIPS verbs and tasks, a TV display device needs to contain the following sub-units. Note, these subunits are logical devices. They may be implemented as control registers in the device or in numerous other fashions. It is only necessary that the Y routines impose on the device a control that forces it to this general structure.

1. IMAGE MEMORIES: These are one or more memories each n bits deep which hold the grey-scale images to be displayed. All n bits of the image contribute to the display. The memory is assumed to have a fixed number of pixels on each axis and to be addressable at the individual pixel level. The addresses are assumed to be one-relative and to begin at the lower left of the display. The number of bits, the dimensions of each axis, and the number of memories are parameters inside AIPS. It is also assumed that each memory may be turned on and off in each of the three colors individually, although the capability is used solely to display images having separate R, G, and B planes. Beginning with the 15APR90 release, we assume that the image memory is actually viewed through a rectangular "window" whose size and coordinates with respect to the image meory may change with time.

2. GRAPHICS MEMORIES: These are one or more memories each 1 bit deep used to display graphical information, such as axis labels or line drawings on top of the grey-scale images. It is assumed that the overlay planes have the same number of pixels on each axis as the image memories and that each overlay plane may be enabled or disabled individually. It is nice to be able to assign unique colors to each of the overlay planes. AIPS will want to use four overlay planes, but all standard programs will work more or less normally with only one. The number of graphics memories is a parameter.

3. CURSOR AND BUTTONS: The cursor is some form of marker which may be enabled or disabled and which is under the positional control of some mechanical device (e.g., trackball, joy stick, thumb wheels, mouse). The position of the cursor on the TV screen may be read at any time it is enabled. The "buttons" are some device to signal conditions to the programs, such as "this is the desired position" or "time to quit". AIPS assumes that there are four such buttons returning to the program a value between 0 and 15. Simultaneity of more than one button is never used, however.

4. LOOK UP TABLES: These are tables of numbers which convert the stored n-bit image intensities to the desired display intensities. AIPS assumes that there is one n-bit in, m-bit out look up table ("LUT") for each color of each image memory. AIPS also assumes that there is a second set of three look-up tables, called the output function memory ("OFM"), which converts the sums of all enabled

memories to the final displayed intensities. In practise, AIPS uses the individual channel LUTs for black and white enhancement (most of the time) and the OFM for pseudo-color enhancement. There are algorithms, such as TVHUEINT, which utilize the full capability of the two sets of look-up tables. Most other routines work well with a single LUT per image memory. Arrays inside AIPS are likely to be dimensioned for 11-bit image planes and a 10-bit OFM. (These assumptions will be increased as the need arises.)

5. SCROLL: It is assumed that each image memory may be displayed on the TV screen shifted along both axes by varying amounts. AIPS assumes that each memory may be scrolled independently and that the graphics memories may be scrolled together independent of the image memories. The minimum increments of scroll along each axis are parameters. Note that AIPS does not make heavy use of scroll except for the TVROAM display and, of course, TVSCROLL. TVROAM does not require the image memories to scroll independently.

6. SPLIT SCREEN: It is assumed that the screen may be divided into quadrants and different image channels enabled in each quadrant. There is a control parameter specifying the degree to which the local TV display has this capability. AIPS currently uses split screen primarily in the TVROAM display, but also uses it during image enhancement in the channel blink routines.

7. ZOOM: AIPS assumes that the display of an image may be blown up about any pixel by automatic pixel replication by simple integer factors (or by integer powers of two) without affect on the images stored in the image memories. The highest factor (or power of two) available is a parameter. Zoom is important to the TVMOVIE algorithm and is used in many of the image enhancement routines.

The most important TV operations of AIPS could be implemented on a TV device having one image memory, one LUT with three OFMs (or three LUTs), and a cursor with buttons. Additional image memories, graphics memories, an OFM, scroll, split screen, and zoom are needed primarily for less important aspects of the basic operations and for some interesting, but esoteric operations.

There are several other sub-units in the IIS Model 70 which are supported by the Y routines in that sub-library. They include an input function memory (translates input to the TV from the host and from the ALU), a histogram generator, a feedback arithmetic logic unit, shift and min/max registers, and the like. Although there are no standard routines in AIPS which use these units, there are two nonstandard tasks for histogram equalization which make some use of them. The special Y routines used by these two tasks will be described below, but they should not (yet) be required for other kinds of TV devices — if they are even possible on them. All functions which are not possible on the current TV device should be represented by "stubbed" versions of the appropriate Y routines and these stubs should return an error condition.

10.2 Fundamentals of the Coding

10.2.1 The Parameter Commons and their Maintenance

All application routines must open the TV device via a call to TVOPEN and close it via a call to TVCLOS. TVOPEN opens a disk file called ID10000n with exclusive use requested, where n is the number of the assigned TV device. From the first record of this file, it reads a 256-word record containing parameters which describe the structure and current status of the assigned TV device. The parameters are stored in a common called /TVCHAR/ which is obtained by including DTVC.INC. TVCLOS puts back to the disk the time variable portions of this common and then closes the file. In this way, several users/programs may share the TV in sequence and all will know the current status information. The disk file may be initialized and the individual parameters set by using the stand-alone program SETTVP. TVOPEN automatically reads the current window coordinates from the device after it is opened and puts them in this common. The parameters are important to the correct functioning of the local TV device and must be set and maintained carefully.

The fixed portion of /TVCHAR/, namely that portion not written by TVCLOS, includes the parameters:

```
NGRAY     The number of n-bit image memories.
NGRAPH    The number of 1-bit graphics overlay memories.
```

```
     NIMAGE      The number of images which may be stored
                 simultaneously in a grey-scale image plane (affects
                 the image catalog mostly).
     MAXXTV(2)   The number of pixels in the X and Y directions.
     MAXINT      The highest grey-scale intensity = 2 ** n - 1.
     LUTOUT      Peak intensity out of LUTs.
     OFMINP      Peak intensity into OFMs.
     OFMOUT      Peak intensity out of OFMs.
     SCXINC      The minimum increment in scroll in the X direction.
     SCYINC      The minimum increment in scroll in the Y direction.
     MXZOOM      If > 0, the highest power of two for zooming.
                 If < 0, highest linear zoom factor = 1 - MXZOOM
                 in simple integer steps.
     CSIZTV(2)   The size of characters in pixels in the X, Y
                 directions.
     TYPSPL      Type of split screen: 0 none, 1 vertical division
                 only, 2 horizontal division only, 3 either, 4 both.
     TVALUS      Number of TV arithmetic logic units.
     TVXMOD      Mode for loading TV in X direction: 0 none, 1 ok in
                 AIPS order (to right), 2 ok in reverse direction.
     TVYMOD      Mode for loading TV in Y direction: 0 none, 1 ok in
                 AIPS order (to top), 2 ok in reverse direction.
     ISUNUM      Number of image storage units.
     TVDUMS(10)  Spare room

The time variable portion of the /TVCHAR/ common is:
     TVZOOM(3)   Current zoom: power of two, X, Y zoom center
     TVSCRX(16)  Current X scroll for 15 image planes and graphics.
     TVSCRY(16)  Current Y scroll for 15 image planes and graphics.
     TVLIMG(4)   Bit pattern for which images are on by quadrant:
                 quadrants are numbered CCW from top right and the
                 lsb is for grey plane one and NGRAY+NGRAPH bits are
                 used.
     TVSPLT(2)   Current split screen position in X, Y.
     TVSPLM      10 * (number planes in X) + (number planes in Y) in
                 Roam mode.
     TVSPLC      Roam mode: digits imply which channels in which
                 order.
     TYPMOV(16)  Movie loop code: 4 * (magnification factor - 1) +
                 64 * (number frames remaining). Add 2 if this is the
                 first plane of the movie.  Add 1 if the frames are
                 in "display" rather than movie order.
     WINDTV(4)   BLC x,y, TRC x,y of the current window visible from
                 the TV memory.  (Not affected by zoom, scroll!)
     TVDUM2(4)   Spare room
     YBUFF(160)  Machine-dependent parameters.
```

There is a second TV include which controls I/O, but is little used elsewhere. It is obtained by including and contains:

```
     TVLUN       LUN of open TV device.
     TVIND       Position of TV device in FTAB for I/O.
     TVLUN2      LUN of open TV parameter disk.
     TVIND2      Position of parameter disk in FTAB.
     TVBFNO      Not used (map style I/O no longer supported).
```

```
TVMAP       Not used.
```

10.2.2 The I/O Routines

Four basic I/O operations for TV devices are supported: open, close, I/O reset ("master clear"), and data transfer (read/write). The actual Z subroutines which perform these operations are both TV device and host operating system specific. The subroutines are stored in the subdirectory appropriate for the host operating system with names reflecting the TV device type. To insure that the correct Z routines are link edited, a layer of Y routines is interposed between these Z routines and all other non-Y AIPS routines. No non-Y subroutine or program should call these Z routines. These Z subroutines have names of the form ZMMMOO, where MMM is the TV model (i.e., M70 for IIS Models 70 and 75, DEA for DeAnza) and OO is the type of operation (OP for open, CL for close, MC for I/O reset, and XF for data transfer).

Note that the four Z routines may have TV-device specific call sequences and that not all devices require four Z routines. The current implementations are

```
Z...OP  :
    ZARGOP (LUN, IND, IERR)          ZARGO2 (FCB, PNAME, IERR)
    ZDEAOP (LUN, IND, IERR)          ZDEAO2 (FCB, PNAME, IERR)
    ZIVSOP (LUN, IND, IERR)
    ZM70OP (LUN, IND, IERR)          ZM70O2 (FCB, PNAME, IERR)
    ZSSSOP (LUN, IND, IERR)          ZSSSO2 (FCB, PNAME, IERR)
    ZV20OP (LUN, IND, IERR)          ZV20O2 (FCB, PNAME, IERR)
    ZVTVOP (LUN, IND, IERR)          ZVTVO2 (FCB, PNAME, IERR)
    ZVTVRO (LUN, IND, IERR)          ZVTVO3 (FCB, PNAME, IERR)
```

Performs the needed channel assignment and opens a non-map entry in the FTAB. The DeAnza version also calls ZDEAXF ('DAT ',...) to initialize the I/O. The Z...OP routines are generic routines which do machine-dependent things by calls to other Z routines including the Z...O2 versions listed. ZSSSOP exists only in APLSUN:.

```
Z...CL  :
    ZARGCL (LUN, IND, IERR)          ZARGC2 (FCB, IERR)
    ZDEACL (LUN, IND, IERR)          ZDEAC2 (FCB, IERR)
    ZM70CL (LUN, IND, IERR)          ZM70C2 (FCB, IERR)
    ZSSSCL (LUN, IND, IERR)          ZSSSC2 (FCB, IERR)
    ZV20CL (LUN, IND, IERR)          ZV20C2 (FCB, IERR)
    ZVTVCL (LUN, IND, IERR)          ZVTVC2 (FCB, IERR)
    ZVTVRL (LUN, IND, IERR)          ZVTVC3 (FCB, IERR)
```

Performs a close (deassign) and clears the FTAB entry. The DeAnza version calls ZDEAXF ('DET ',...) to perform a deallocation before closing. The M70 version flushes the IO buffer before the close.

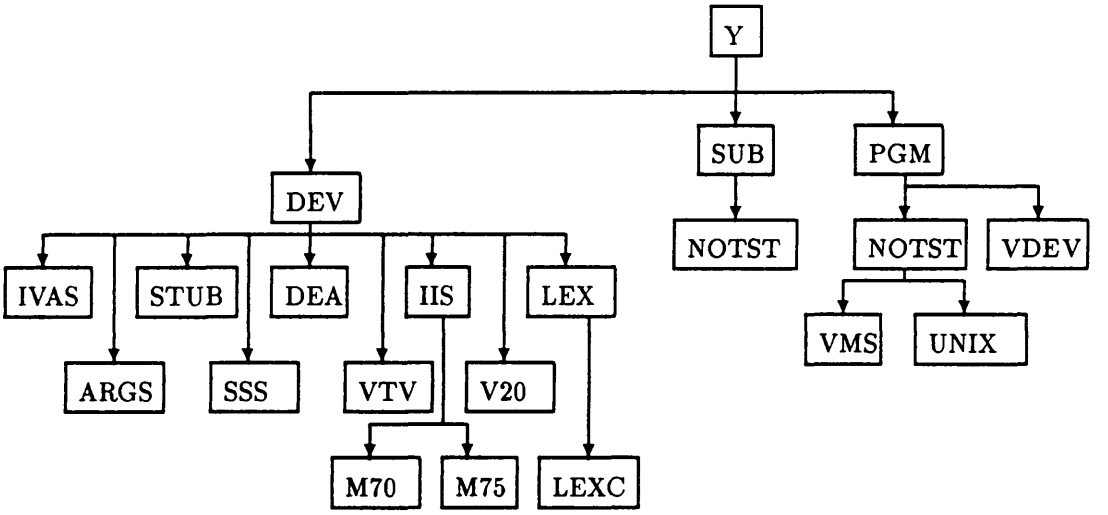
```
Z...MC  :
    ZM70MC (FCB)                     ZM70M2 (FCB, IERR)
       Performs a "rewind" QIO operation causing the IIS to
       reset its I/O status.
    ZARGMC (FCB)
       Invokes ZARGS
    ZDEAMC
    ZSSSMC
    ZV20MC
       Null subroutines.


Z...XF  :
    ZARGXF (OP, NBYTES, HEADER, BUFFER, IERR)
       translates IIS Model 70 like commands for ARGS and then
```

```
   calls a selected portion of ZARGS
ZDEAXF (OPER, BUFFER, NBYTE, PA, PB, WAIT, IERR)
ZDEAX2 (FCB, IOP, OPER, BUFFER, NBYTE, PA, PB, WAIT, IERR)
   "Calls to ZDEAXF map one to one to calls to IP8 routines
   in the DeAnza IP8500 level 0 software package." Does
   requested I/O operation using opcode definitions
   contained in IP8IOF.MAR (supplied by DeAnza, not NRAO).
ZM70XF (OPER, HBYTES, HEADER, BUFFER, IERR)
ZM70X2 (OPER, FCB, BUFF, NBYTES, IERR)
   writes an eight-word command HEADER to the IIS after
   preparing the checksum word of the header. Then reads
   from or writes to the IIS NBYTES of BUFFER. Actual
   writing to the IIS is deferred until a large buffer is
   filled or a read or close occurs. Issues a master
   clear on error.
ZSSSXF (OP, DAT, NWORDS, BUFFER, NSWORD, IERR)
ZSSSX2 (FCB, OP, DAT, NWORDS, BUFFER, NSWORD, ISTAT)
   reads/writes from the stand-alone SSS program which
   creates and operates the workstation ''TV'' device.
ZV20XF (OPER, NBYTES, COMST, COMDB, BUFFER, IERR)
ZV20X2 (OPER, FCB, NBYTES, COMST, COMDB, BUFFER, IERR)
   performs the actual QIOW operation to transmit data
   to/from the Vision 1/20. See ZB.DOC provided with the
   Comtal device driver ZBDRIVER.
ZVTVXF (BUFSW, BUFSR, HBUF, IERR)
ZVTVX2 (FCB, BUFSW, BUFSR, BUFFER, IERR)
   reads/writes from/to server (real TV computer) from the
   client (virtual TV). Thus, is used by AIPS et al.
ZVTVRX (FIND, BUFSW, HBUF, IERR)
ZVTVX3 (FCB, BUFSW, HBUF, IERR)
   reads/writes from/to client (i.e., the virtual TV) from the
   server (actual TV). Thus, is used by program TVMON.
```

10.2.3 The Y Routines

The directory structure of AIPS was designed with the need to support multiple TV devices in mind. See the Appendix to Volume I of Going AIPS for details. The highest level of the Y-routine device-dependent directory tree, called Y/DEV with logical name YGEN, contains a "generic" version of the Y subroutine library. In some cases, there really can be a generic version of a routine, i.e., vectors can be drawn on the TV screen by a sequence of calls to the image writing routine. In many cases, however, the Y routines in YGEN are "stubs" which issue an error message and return error code 2. Below this is a set of directories for the various supported devices. The diagram below summarizes the full Y directory tree:



Y Directories

The STUB area is used to hold stubbed versions of those Y routines which have device-independent implementations. On those systems with no TV display, link edits of AIPS and other "TV" programs will use this area. Each of the end nodes of this tree has a corresponding link library. If a routine is present in the end node, then the compiled version of that routine will be in the corresponding library. If the routine is not present, then the compiled version of the routine from the next higher node having that routine will be placed in the library. Thus, one needs to write for the IVAS, for example, only those Y routines for which the DEV version will not do. In particular, since the IVAS does not have a feed back arithmetic logic unit (for example), we can use the generic (stubbed) version of YALUCT and have no version of YALUCT in the IVAS directory.

In the chart below, the names of all routines in the Y directories are given along with indications of which directories have versions of them. The codes are "G" for generic, "S" for stubbed, "D" for device dependent, and "O" for operating system dependent. The routines are broken into four "levels" depending on how they may be used. Levels 0 and 1 may be called from non-Y routines and level 3 may not. Level 2 was originally designed to be called from non-Y code, but experience has shown that the device capabilities are available only on IIS Models 70 and 75. and, hence, they are not recommended.

Level 0: required, but a generic is available

Routine	DEV	IIS	M70	M75	IVAS	DEA	V20	STUB	VTV	SSS
YCHRW	G				D			S	D	
YCINIT	G								D	
YCNECT	G				D			S	D	
YCOVER	G								D	
YCREAD	G								D	
YCUCOR	G		D					S	D	
YCURSE	G		D					S	D	
YCWRIT	G								D	
YFILL	G				D				D	
YFIND	G								D	
YLOCAT	G								D	
YLOWON	G									
YSLECT	G							S	D	
YTCOMP	G									
YTVCLS	G								D	

YTVOPN	G						S	D		
YWINDO	G							D	D	

Level 1: required, device-dependent is needed

Routine	DEV	IIS	M70	M75	IVAS	DEA	V20	STUB	VTV	SSS
YCRCTL	S		D	D	D	D	D		D	D
YGRAPH	S		D	D	D	D	D		D	D
YIMGIO	S	D			D	D	D		D	D
YINIT	S		D	D	D	D	D		D	D
YLUT	S	D			D	D	D		D	D
YOFM	S		D	D	D	D	D		D	D
YSCROL	S		D	D	D	D	D		D	D
YSPLIT	S	D			D	D	D		D	D
YTVCIN	S		D	D	D	D	D			D
YTVCL2	S	D			D	D	D		D	D
YTVMC	S	D			D	D	D		D	D
YTVOP2	S	D			D	D	D		D	D
YZERO	S	D			D	D	D		D	D
YZOOMC	S		D	D	D	D	D		D	D

Level 2: IIS dependent routines - not available to other TVs

Routine	DEV	IIS	M70	M75	IVAS	DEA	V20	STUB	VTV	SSS
YALUCT	S		D	D					D	
YCONST	S		D	D					D	
YFDBCK	S		D	D					D	
YIFM	S		D	D					D	
YMNMAX	S		D	D					D	
YRHIST	S		D	D					D	
YSHIFT	S		D	D					D	

Level 3: routines only called by Y routines - not available to non-Y

Routine	DEV	IIS	M70	M75	IVAS	DEA	V20	STUB	VTV	SSS
BYTE2I							O			
I2BYTE							O			
YBUTON					D					
YCHACT							D			
YCMND							D			
YCMSET							D			
YDOERR					D					
YGGRAM	S		D	D		D				
YGRAFE	S		D	D						
YGYHDR	S		D	D						
YISDRM	S		D							
YISDSC	S		D							
YISJMP	S		D							
YISLOD	G									
YISMPM	S		D							
YLINTV							D			

YMAGIC	S		D			
YMKCUR	S				D	
YMKHDR	S	D	D			
YSTCUR	S	D	D			D
YVRTR				D		

The following sections provide a brief overview of the current Y routines. The precursor comments of most of the Y routines are reproduced near the end of this chapter.

Level 0

1. YCHRW writes characters into an image or graphics plane. The DEV version is TV independent and uses a 7 x 9 pixel area per character. The background intensity is set to 1 for multi-bit channels and 0 for graphics. Uses YIMGIO.
2. YCINIT initializes the TV image catalog. It is a Y routine to allow for remote TV devices.
3. YCNECT writes a line segment in an image or graphics plane at a specified intensity. It is a Y routine to allow for TV devices with hardware vector generators. Uses YIMGIO.
4. YCOVER checks the TV image catalog to see if there are overlapping images visible. It is a Y routine to allow for remote TV devices.
5. YCREAD uses the TV image catalog to determine which image is associated with a particular pixel. It is a Y routine to allow for remote TV devices.
6. YCUCOR converts cursor positions and obtains the corresponding image header. It is a specialized version of YCURSE to avoid any TV I/O and to do the image catalog work.
7. YCURSE enables/disables cursor and cursor blink and reads cursor position and buttons value. The main complications come from corrections for zoom and scroll. The generic version uses YCRCTL and assumes that zoom/scroll is done by specifying the pixel to be visible in the upper left corner.
8. YCWRIT updates the TV image catalog to add an image to the display list. It is a Y routine to allow for remote TV devices.
9. YFILL fills a rectangular region of the display with an image of constant intensity. Uses YIMGIO and is a Y routine to allow for devices with hardware polygon fill.
10. YFIND uses the TV image catalog to determine if only one image is visible and, if so, to return the image header. It is a Y routine to allow for remote TV devices.
11. YLOCAT uses the TV image catalog to convert a list of image pixel positions to a list of TV positions. It is a Y routine to allow for remote TV devices.
12. YLOWON selects the lowest numbered channel from a bit mask. It is a Y routine for no good reason.
13. YSLECT enables/disables grey and graphics channels setting the proper values into TVLIMG. Uses YSPLIT for image planes and YGRAPH for graphics planes.
14. YTCOMP performs logical tests on parameter values to see if they have changed. It is a Y routine only because it is now used only to minimize I/O to the DeAnza control registers.
15. YTVCLS updates and closes the TV parameter disk file and closes the TV device via YTVCL2.
16. YTVOPN opens and reads the TV parameter disk file and opens the TV device via YTVOP2.
17. YWINDO reads current viewing rectangle on the image memory, can force it on workstations.

Level 1

1. YCRCTL reads/writes the cursor/trackball control register including position, enable/disable on each axis, blink control.
2. YGRAPH enables/disables graphics overlay planes by altering the graphics color look up tables. A non-essential nicety is the use of complimentary colors when two or more graphics planes are enabled at the same pixel.
3. YIMGIO reads/writes a line of image data from/to a grey-scale or graphics plane. It will perform buffer swaps if needed to get the desired angle and bit-level corrections when graphics planes are read. This is the most heavily used Y routine, in part because of the generic versions of YCHRW and YCNECT.
4. YINIT initializes all subunits of the TV, clears the TV memories, resets the image catalog, and resets status parameters in common.
5. YLUT reads/writes the full channel-level lookup table for one or more image channels and colors.
6. YOFM reads/writes the full OFM lookup table for one or more colors.
7. YSCROL writes the scroll control registers for one or more channels.
8. YSPLIT reads/writes the split screen control registers. This is the actual control of the split screen center and of which channel(s) are enabled/disabled in each quadrant.
9. YTVCIN provides initial values for the TV characteristics commons.
10. YTVCL2 closes the TV device. Actually it is usually just an interface to the appropriate Z...CL subroutine.
11. YTVMC resets the TV I/O status. Actually it is usually just an interface to the appropriate Z...MC subroutine.
12. YTVOP2 opens the TV device. Actually it is usually just an interface to the appropriate Z...OP subroutine.
13. YZERO clears a full grey or graphics memory by the fastest possible method.
14. YZOOMC writes the zoom control registers giving magnification and zoom center.

Level 2

1. YALUCT reads/writes the IIS arithmetic logic unit control registers. No actual function is performed until a feedback operation is done via YFDBCK. This routine is very IIS specific and we doubt that its functions can be implemented on other TVs.
2. YCONST reads/writes the constant "biases" which are added to the sums of the individual enabled channels before the signals are sent to the OFM.
3. YFDBCK causes a feedback operation to occur. The ALU does its thing with one or more channels and returns an 8 or 16 bit result to one or two channels. A magic bit causes the function to be a simple zeroing of a channel.
4. YIFM reads/writes a portion of the input function memory. This lookup table can be used in writing data to the TV memory and in the feedback operation. AIPS does not do the former and only one non-standard task does the latter.
5. YMNMAX reads the min and max output from the sum of all enabled grey-scale planes for each color.
6. YRHIST reads a portion of the histogram of the output of the OFM for a selected color. The IIS can do this on the fly if properly equipped.
7. YSHIFT reads/writes the shift registers which shift the 13-bit output of the sum of all enabled channels before the data get to the OFM.

Level 3

1. BYTE2I converts VMS BYTE data to integer forcing the BYTE variable to be in the range 0-255.
2. I2BYTE converts integer data in range 0-255 to VMS BYTE data.
3. YBUTON reads the state of the button buffer of the TV. IVAS uses this for converting two different buttons into "button D".
4. YCHACT activates a specified Comtal channel.
5. YCMND sends a command string to the Comtal via ZV20XF.
6. YCMSET sets up the Comtal driver for I/O.
7. YDEA.INC Include file giving parameter definitions to specify positions in YBUFF which correspond to the various registers in a DeAnza TV device.
8. YDOERR handles error conditions, reporting, and resetting for the IVAS.
9. YGGRAM reads/writes the lookup table used for graphics planes.
10. YGRAFE reads/writes the graphics control register which assigns a graphics plane as the "blotch" plane and another as the "status" plane. No use is made of this.
11. YGYHDR prepares a basic I/O control header for writing/reading image data to/from the IIS.
12. YISDRM reads/writes data memory of the NRAO Image Storage Unit.
13. YISDSC reads/writes micro-processor memory of the NRAO Image Storage Unit from/to the ISU disks.
14. YISJMP causes the micro-processor of the NRAO Image Storage Unit to jump to a specified address.
15. YISLOD reads/writes program memory of the NRAO Image Storage Unit from/to disk. Uses YISMPM.
16. YISMPM reads/writes micro-processor memory of the NRAO Image Storage Unit.
17. YLINTV sends a line to/from Comtal image or graphics planes.
18. YMAGIC (Model 75 only) initializes graphics, zoom, and scroll subunits (called by YINIT only).
19. YMKCUR creates and loads the cursor pattern memory with a specified shape. Only the AIPS plus sign is implemented.
20. YMKHDR prepares a basic I/O control header for the IIS.
21. YSTCUR reads/writes the IIS cursor array. This 64 x 64 bit array provides a wide choice of patterns for the display "cursor". AIPS uses only a simple plus sign with a blank pixel at the center.
22. YVRTR checks and switches as needed the transfer restrictions on the IVAS.

10.3 Current Applications

This section is devoted to a generally brief overview of the current application code. Primarily it will be used simply to point out which routines do what, with some comment on the methods. This should suffice as an introductory guide to the code for applications programmers wishing to include the TV display in their programs. In a couple of cases, some of the actual code will be reproduced in order to clarify the use of the various service routines. The precursor remarks for some of the most commonly used, non-Y service routines are reproduced at the end of this chapter.

10.3.1 Status Setting

By "status setting", we mean initializing the TV device, clearing memory channels, enabling and disabling portions of the display, and the like. Many of the applications which involve loading images to the TV display will zero the relevant memories (via YZERO) and clear the corresponding portions of the image catalog (via YCINIT) before carrying out their primary functions. However, the simplest examples of status setting are those performed by various AIPS verbs. The subroutine AU5 performs the verbs TVINIT (via YINIT), TVCLEAR (as follows), GRCLEAR (like TVCLEAR without the MOVIST call), TVON, TVOFF, GRON, GROFF (via calls to YSLECT), TV3COLOR (use YSLECT to turn off all channels, then YSLECT to turn on channels 1 through 3 in red, green, blue, respectively), and CURBLINK (via YCURSE). Other verbs in AU5 are described later.

The verb TVCLEAR is coded as follows. The channel number is picked up as an integer, the decimal code is converted to a bit pattern (via DECBIT), the movie status parameters are reset (via MOVIST), and then a loop over all selected grey planes is done to zero the memory (via YZERO) and clear the image catalog (via YCINIT).

```
C                                       Open TV device
      CALL TVOPEN (CATBLK, JERR)
      IF (JERR.NE.0) THEN
         POTERR = 101
         GO TO 980
         END IF
....
 200  ICHAN = ABS (TVCHAN) + EPS
C                                       convert to channel bit mask
      CALL DECBIT (NGRAY, ICHAN, ICHAN, ITEMP)
C                                       clear movie parameters
      CALL MOVIST ('OFFF', ICHAN, 0, 0, 0, IERR)
      DO 210 IP = 1,NGRAY
C                                       is plane requested
         ITEMP = 2 ** (IP-1)
         IF (IAND (ICHAN,ITEMP).NE.0) THEN

C                                       clear image catalog
            CALL YCINIT (IP, INBUF)
C                                       clear TV memory
            CALL YZERO (IP, JERR)
            IF (JERR.NE.0) GO TO 975
            TDEL = TDEL + 1.0
            END IF
 210     CONTINUE
      GO TO 900
....
C                                       normal TV close
 900  CALL TVCLOS (CATBLK, JERR)
      IF (TDEL.GT.0.0) THEN
         TDEL = TDEL + 1.0
         CALL ZDELAY (TDEL, IERR)
         END IF
      GO TO 999
```

10.3.2 Load Images, Label

Images are loaded to the TV by a wide variety of tasks (e.g., APCLN, TVPL, BLANK) and by several verbs (TVLOD, TVROAM, TVMOVIE). TVLOD will be illustrated in this subsection and the others mentioned in later subsections.

The full code from subroutine AU5A for TVLOD, except declarations, formats, error branches, and the like, is reproduced below. It begins by opening the TV control file and device (via TVOPEN). It moves the user adverbs to local variables to avoid changing their (global) values and opens the map file (via MAPOPN). It converts the user's PIXRANGE adverb using standard defaults (via RNGSET) and fills in some of the image catalog parameters in the header. It sets the window parameters (via TVWIND), selects a single grey scale memory plane (via DECBIT), and clears the movie parameters (via MOVIST). Finally, it finishes up the image catalog parameters, puts the header in the image catalog, and reads, scales, and loads the image to the TV memory (all via TVLOAD). Afterwards, it closes the map file (via MAPCLS) and the TV device and disk file (via TVCLOS).

```
      INCLUDE 'INCS:DHDR.INC'
C                                       open TV
      CALL TVOPEN (INBUF, IERR)
      IF (IERR.NE.0) GO TO 980
      IBSIZ = 2 * 4096
C                                       Map open junk: TVLOD, TVROAM
      IF (BRANCH.LE.2) THEN
C                                       adverbs -> local variables
C                                       Adverbs used:
C                                       TVCHAN = tv channel
C                                       INNAM  = File name
C                                       INCLS  = File class
C                                       INSEQ  = File sequence number
C                                       INDSK  = Disk number
C                                       USERID = User ID number
C                                       TVBLCO = TV bottom left corner
C                                       TVTRCO = TV top right corner
C                                       TVXINC = TV x pixel increment
C                                       TVYINX = TV y pixel increment
C                                       PXRANG = Range of pixel values
C                                       TVCORN = BLC on TV screen for
C                                                image
         ICHAN = IROUND (TVCHAN)
         IVOL = INDSK + EPS
         USID = ABS(USERID) + EPS
         SEQNO = INSEQ + EPS
         IF (USID.EQ.0) USID = NLUSER
         IF (USID.EQ.MAGIC) USID = 0
         CALL H2CHR (12, 1, INNAM, NAME)
         CALL H2CHR (6, 1, INCLS, CLASS)
         INC(1) = IROUND (TVXINC)
         INC(2) = IROUND (TVYINC)
         PTYPE = 'MA'
                                        open map file
         CALL MAPOPN ('READ', IVOL, NAME, CLASS, SEQNO, PTYPE, USID,
            DLUN, DIND, CNO, CATBLK, INBUF, IERR)
         POTERR = 33
         IF (IERR.GT.1) GO TO 975
         CALL RCOPY (7, TVBLCO, LBLC)
         CALL RCOPY (7, TVTRCO, LTRC)
```

```
C                                       Image cat fill in some
        CALL RNGSET (PXRANG, CATR(KRDMX), CATR(KRDMN), CATR(IRRAN))
        CATBLK(IIVOL) = IVOL
        CATBLK(IICNO) = CNO
        CATN(IITRA) = FUNTYP
        ITVC(1) = TVCORN(1) + EPS
        ITVC(2) = TVCORN(2) + EPS
        POTERR = 49
        END IF
C                                       TVLOD
C                                       load one image plane
C                                       set windows
 100  TYPE = -1
      CALL DECBIT (NGRAY, ICHAN, ICHAN, I)
      CALL TVWIND (TYPE, INC, LBLC, LTRC, I, ITVC, IWIN, IERR)
      IF (IERR.NE.0) GO TO 970
C                                       convert channel number
      ICHAN = I
      CALL DECBIT (NGRAY, ICHAN, ICHAN, I)
C                                       clear movie parameters
C                                       load it
      CALL MOVIST ('OFFF', ICHAN, 0, 0, 0, IERR)
C                                       do the TV load
C                                       image catalog
      CALL TVLOAD (DLUN, DIND, I, INC, ITVC, IWIN, IBSIZ, RINBUF, IERR)
      IF (IERR.EQ.0) POTERR = 0
      GO TO 970

C                                       Close down ops
 970  CALL MAPCLS ('READ', IVOL, CNO, DLUN, DIND, CATBLK, F, INBUF,
     *   IERR)
C
 975  CALL TVCLOS (INBUF, IERR)
```

The verbs TVWEDGE, IMWEDGE, IMERASE, and WEDERASE load step wedge or pure zero images to the TV. They occur in subroutine AU5C. This routine calls TVFIND and possibly TVWHER to determine which image is desired. It then computes a buffer of appropriate values calling ISCALE (as TVLOAD does). AU5C then does a lot to set an appropriate image catalog header and writes that to the catalog via YCWRIT. Finally it loads the TV rows via calls to YIMGIO.

The image labeling verbs TVLABEL and TVWLABEL are implemented from subroutine AU5B. This routine calls TVFIND to determine which image is to be labeled and IAXIS1 to do the labeling. Subroutines IAXIS1 and ITICS are very similar to the standard axis labeling routines used to make plot files and to write directly to the TEK graphics device. Characters are written to a graphics memory with a black background by calls to IMANOT and lines are written to the graphics memory by calls to IMVECT. (See the precursor comments of these routines at the end of this chapter.) The verb TVANOT, which adds a user-provided string to the display, is also implemented in AU5B and uses TVWHER, IMANOT, and IMCHAR.

10.3.3 UVMAP

UVMAP uses the TV display for a fairly simple purpose — to show the pattern of sampled uv cells (after convolution of the data to the grid). In principle, the algorithm is simple: associate uv cells with TV pixels and display 0 on the TV when the uv cell is unsampled (0.00) and display MAXINT on the TV when the cell is sampled (not 0.0). Unfortunately, the uv grid may be larger than the TV display and the disk file contains the grid in transposed, quadrant-swapped order. The first problem is solved by decimation (examine only every n'th cell in X and m'th cell in Y). The quadrant swapping is solved by addressing the TV beginning

in the middle and by starting in the middle of the buffer which is written to the TV. The transposition is solved by writing the rows of the file as columns on the TV. The subroutine in UVMAP which does this (UVDISP) uses the image writing mode parameters (TVYMOD and TVXMOD) to handle this correctly when possible, and to leave the display in transposed order when not (i.e., TVYMOD = 0).

10.3.4 APCLN, VTESS, MX, et al.

Iterative map analysis programs can make good use of the TV display. The user may, for example, request that APCLN display the residual map after each major cycle. APCLN does this, then turns on the cursor and waits up to 15 seconds for the user to push Button D to signify that sufficient iterations have been performed. Several tasks (currently MX, APGS, SDCLN, VTESS, UTESS, APVC) use code similar to that in APCLN for loading the image to the TV and requesting the user input. Given below is the TV subroutine from APCLN. Note that it uses the array processor to scale the data for YIMGIO. This is reasonable, but only for tasks which are already using the array processor for more important computations. The costs of opening and closing the AP device and performing the I/O to it make any improvement in computational speed marginal for computations such as these. Note also the scaling parameters used here. The lowest displayed intensity gets TV value 1.01 and the highest gets MAXINT+0.99 (after the 0.5 for rounding is added and before the integers are truncated by routine QVVFIX). This scaling is assumed (primarily by CURVALUE) for all linear transfer functions. TV value zero is reserved for "blanked" (indefinite) pixels and should always be given zero intensity on the display (by the LUTs and OFMs).

```
      SUBROUTINE DISPTV (TVPASS)
C-----------------------------------------------------------------------
C   DISPTV displays the current residual map on the TV, showing inner
C   portion only if that's all that will fit.
C   Inputs:
C      TVPASS  I     code: 0,1 => clear screen, else don't
C                          0,3 => don't question the user about
C                                 quitting
C   Output:
C      TVPASS  I     code: 32700 => user wants to quit cleaning
C-----------------------------------------------------------------------
      INTEGER   TVPASS
C
      CHARACTER PREFIX*5
      INTEGER   JROW(1), WIN(4), MY, FIND, BIND, IERR, ICH, CATII(256),
     *   IQ, IB, I, MX2, MX, IWIN(4), IY
      REAL      XN(4), XBUFF(1), TD, RPOS(2), XFLUX, TVLMAX, TVLMIN,
     *   ARG, CATIR(256)
      LOGICAL   MAP, EXCL, WAIT, LERR, F
      INCLUDE 'APCLN.INC'
      INCLUDE 'INCS:DFIL.INC'
      INCLUDE 'INCS:DTVC.INC'
      INCLUDE 'INCS:DMSG.INC'
      INCLUDE 'INCS:DHDR.INC'
      INCLUDE 'INCS:DTVD.INC'
      INCLUDE 'INCS:DCAT.INC'
      EQUIVALENCE (JROW, BUFF2),    (BUFF1, XBUFF)
      EQUIVALENCE (CATII, CATIR, BUFF1(513))
      DATA MAP, EXCL, WAIT, F / 3*.TRUE., .FALSE./
C-----------------------------------------------------------------------
      ICH = 1
      CALL TVOPEN (BUFF1, IERR)
      IF (IERR.NE.0) THEN
```

```
         WRITE (MSGTXT,1000) IERR
         CALL MSGWRT (6)
         GO TO 999
         END IF
      IF (TVPASS.LE.1) THEN
         CALL YZERO (ICH, IERR)
         IF (IERR.NE.0) THEN
            WRITE (MSGTXT,1010) IERR
            CALL MSGWRT (6)
            GO TO 998
            END IF
         CALL YCINIT (ICH, XBUFF)
         END IF
C                                        Set max/min for display
      IF (TVFMAX.LE.TVFMIN)
         TVFMAX = TVREMX
         TVFMIN = TVREMN
         END IF
      IF (TVREMX.GT.TVFMAX) TVFMAX = TVREMX
      IF (TVREMN.LT.TVFMIN) TVFMIN = TVREMN
      TVLMAX = TVFMAX - TVFMIN
      IF (0.1*TVLMAX.GT.TVREMX-TVREMN) THEN
         ARG = 0.1 * TVFMIN
         TVFMIN = MIN (ARG, TVREMN)
         ARG = TVFMIN + 0.1 * TVLMAX
         TVFMAX = MAX (ARG, TVREMX)
         TVLMAX = TVFMAX - TVFMIN
         END IF
      XN(1) = TVFMIN
      XN(2) = TVFMAX
      XN(3) = (MAXINT - 0.02) / TVLMAX
      XN(4) = 0.51 - TVFMIN * XN(3)
      CALL QPUT (XN, 0, 4, 2)
C                                        Write scaling factor
      XFLUX = TVLMAX
      CALL METSCA (XFLUX, PREFIX, LERR)
      TVLMIN = TVFMIN * XFLUX / TVLMAX
      TVLMAX = TVFMAX * XFLUX / TVLMAX
      WRITE (MSGTXT,1020) TVLMIN, TVLMAX, PREFIX
      CALL MSGWRT (1)
C                                        Set window to display
      WIN(1) = (WINM(3,1) + WINM(1,1)) / 2 - MAXXTV(1) / 2 + 1
      WIN(1) = MAX (1, WIN(1))
      WIN(2) = (WINM(4,1) + WINM(2,1)) / 2 - MAXXTV(2) / 2 + 1
      WIN(2) = MAX (1, WIN(2))
      WIN(3) = (WINM(3,1) + WINM(1,1)) / 2 + MAXXTV(1) / 2
      WIN(3) = MIN (NX, WIN(3))
      WIN(4) = (WINM(3,1) + WINM(1,1)) / 2 + MAXXTV(2) / 2
      WIN(4) = MIN (NY, WIN(4))
      DO 70 I = 1,2
         IWIN(I) = (MAXXTV(I) - WIN(I+2) + WIN(I) + 1)/2
         IF (IWIN(I).GE.1) GO TO 50
            IWIN(I) = 1
            WIN(I) = (WIN(I+2) + WIN(I) - MAXXTV(I) + 1)/2
```

```
            GO TO 60
 50       IWIN(I+2) = IWIN(I) + WIN(I+2) - WIN(I)
          IF (IWIN(I+2).LE.MAXXTV(I)) GO TO 70
 60       IWIN(I+2) = MAXXTV(I)
          WIN(I+2) = WIN(I) + IWIN(I+2) - IWIN(I)
 70    CONTINUE
C                                       Prepare to read map.
      CALL ZOPEN (LUNRES, FIND, RESVOL, RESFIL, MAP, EXCL, WAIT, IERR)
      CALL MINIT ('READ', LUNRES, FIND, NX, NY, WIN, XBUFF, BUFSZ1,
     *   BORES, IERR)
      IF (IERR.NE.0) GO TO 110
      MX = WIN(3) - WIN(1) + 1
      MY =  WIN(4) - WIN(2)+ 1
C                                       loop, passing map to TV.
      DO 100 I = 1,MY
         IY = I + IWIN(2) - 1
         CALL MDISK ('READ', LUNRES, FIND, XBUFF, BIND, IERR)
         IF (IERR.NE.0) GO TO 110
c                                       clip, sacle, and fix in Q routines
         CALL QPUT (XBUFF(BIND), 4, MX, 2)
         CALL QWD
         CALL QVCLIP (4, 1, 0, 1, 4, 1, MX)
         CALL QVSMSA (4, 1, 2, 3, 4, 1, MX)
         CALL QVFIX (4, 1, 4, 1, MX)
         CALL QWR
         CALL QGET (JROW, 4, MX, 1)
         CALL QWD
C                                       Send row to TV.
         MX2 = MX
         CALL YIMGIO ('WRIT', ICH, IWIN, IY, 0, MX2, JROW, IERR)
         IF (IERR.NE.0) GO TO 110
 100     CONTINUE
 110  CALL ZCLOSE (LUNRES, FIND, IERR)
C                                       Release the AP
      CALL QRLSE
C                                       Update image catalog
      CALL COPY (256, CATBLK, CATII)
      CATII(IIVOL) = 0
      CATII(IICNO) = 0
      CALL FILL (5, 1, CATII(IIDEP))
      CALL COPY (4, IWIN, CATII(IICOR))
      CALL COPY (4, WIN, CATII(IIWIN))
      CALL CHR2H (2, '  ', 1, CATII(IITRA))
      CATIR(IRRAN) = TVFMIN
      CATIR(IRRAN+1) = TVFMAX
      CATIR(KRDMN) = TVREMN
      CATIR(KRDMX) = TVREMX
      CALL YCWRIT (ICH, IWIN, CATII, XBUFF, IERR)
      IF (IERR.EQ.0) GO TO 120
         WRITE (MSGTXT,1110)
         CALL MSGWRT (6)
C                                       Ask user to quit?
 120  IF ((TVPASS.NE.1) .AND. (TVPASS.NE.2)) GO TO 998
         WRITE (MSGTXT,1120)
```

```
         CALL MSGWRT (1)
         WRITE (MSGTXT,1121)
         CALL MSGWRT (1)
         RPOS(1) = (WINDTV(1) + WINDTV(3)) / 2.0
         RPOS(2) = (WINDTV(2) + WINDTV(4)) / 2.0
         TD = 0.2
         CALL YCURSE ('ONNN', F, F, RPOS, IQ, IB, IERR)
         IF (IERR.NE.0) GO TO 998
         DO 130 I = 1,75
            CALL ZDELAY (TD, IERR)
            CALL YCURSE ('READ', F, F, RPOS, IQ, IB, IERR)
            IF (IB.GT.7) GO TO 140
            IF (IB.GT.0) GO TO 135
            IF (IERR.NE.0) GO TO 135
 130        CONTINUE
 135     WRITE (MSGTXT,1135)
         CALL MSGWRT (1)
         GO TO 150
C                                    Wants to quit
 140     TVPASS = 32700
         WRITE (MSGTXT,1140)
         CALL MSGWRT (3)
C                                    Off cursor
 150     CALL YCURSE ('OFFF', F, F, RPOS, IQ, IB, IERR)
 998  CALL TVCLOS (BUFF1, IERR)
C
 999  RETURN
C-------------------------------------------------------------------------
 1000 FORMAT ('CAN''T OPEN TV IER=',I6)
 1010 FORMAT ('IMCLEAR ERROR =',I6)
 1020 FORMAT ('TVDISP: Display range =',2F8.3,1X,A5,'Jy')
 1110 FORMAT ('CAN''T UPDATE IMAGE CATALOG  IER=',I6)
 1120 FORMAT ('Hit button D within 15 seconds to stop cleaning now')
 1121 FORMAT ('Hit buttons A, B, or C to continue sooner')
 1135 FORMAT ('Continuing')
 1140 FORMAT ('TV Button D hit: have done enough I guess')
      END
```

10.3.5 Plot Files (TVPL)

Plots in AIPS are usually produced as device-independent plot files (see the chapter on plotting). The task which interprets such files and writes on the TV display is called TVPL. It will scale line drawings to fill the TV screen or, at the user's option, plot them at the original pixel scaling (converted to TV pixels). Grey-scale plot files are always done at pixel scaling. The character and vector portions of the plot are written to one of the graphics planes (chosen by the user) via subroutines IMVECT and IMCHAR. Grey-scale records, if any, are written via YIMGIO to the user-specified grey-scale memory. TVPL also updates the image catalog as needed.

10.3.6 Transfer Function Modification, Zooming

Subroutine AU6A carries out the verbs OFFTRAN, TVTRANSF, TVLUT, and TVMLUT which perform modifications on the black and white (or single color) LUTs of the specified grey-scale memories. OFFTRAN simply writes a linear, 0 through MAXINT array to the LUTs via YLUT. TVTRAN is implemented by the subroutine IENHNS which is also used by other verbs and tasks (e.g., TVFIDDLE, BLANK, TVMOVIE,

TVBLINK). IENHNS allows a linear LUT with the cursor position controlling the slope and intercept and buttons allowing a switch in the sign of the slope and a continually updated plot of the LUT. TVLUT and TVMLUT allow the user to plot his own LUT function on a graphics plane with the cursor and the buttons. They both use the subroutine GRLUTS.

Subroutine AU6 implements the verbs OFFPSEUD, OFFZOOM, and OFFSCROL to clear the OFM, the zoom setting, and the scroll(s). It also implements interactive setting of the zoom factor and center (verb TVZOOM), of individual channel scrolls (TVSCROL), and of the pseudo-color OFM (TVPSEUDO). OFFPSEUD simply sends a linear OFM to all colors via YOFM; OFFZOOM sends a 0 zoom factor via YZOOMC; and OFFSCROL sends a 0 scroll via YSCROL. TVZOOM makes considerable use of YCURSE and YZOOMC, while TVPSEUDO uses YCURSE and alternately IMLCLR (RGB color triangle), IMPCLR (circle in hue), and IMCCLR (color contours). AU6 also implements a much more complicated enhancement algorithm in which one grey-scale channel is used to set the intensity and another to set the hue. This algorithm requires the TV to have an LUT for each color and channel and an OFM for the sum of the enabled channels for each color. A log function is put in the LUTs and an exponential in the OFM to carry out the required multiplication of the two signals. Subroutines HIENH and HILUT actually do most of the algorithm including interactive enhancements (via an algorithm similar to IENHNS) and switching of the roles of the two channels.

One of the most commonly used image enhancement routines is TVFIDL. It is called by the verb TVFIDDLE via subroutine AU6C and several tasks including BLANK and TVFLG. It is a deliberately limited interactive routine designed to provide easy-to-use enhancement in black and white (via IENHNS) or pseudo-color (via IMCCLR with a single type of color contour). A simple zoom procedure is also provided. During image enhancement, the cursor position controls slope and intercept and, during zoom, the cursor position controls zoom center. Button A (value 1) alternately selects color and black and white enhancement, button B/C increments/decrements the zoom and selects zoom mode. As in all interactive algorithms, button D (values ≥ 8) terminates the function.

The algorithm for TVSCROL is a good example to present in detail since the action required when the cursor moves is quite simple. The most important thing to notice below is the routine DLINTR. This routine tests the output of YCURSE to see if anything has changed. If not, it delays the program by some period of time which increases slowly as the time since the last change increases. Without this algorithm, the tight loop on reading the TV cursor is capable of jamming the CPU and I/O channels, especially when the user does not move the cursor. DLINTR also keeps the cursor from moving off the screen (wrapping around to the other side).

```
C                                        general parameters
      QUAD = -1
      RPOS(1) = 0.0
      RPOS(2) = 0.0
C                                        open TV device
      CALL TVOPEN (BUFFER, IERR)
C                                        get start time
      CALL ZTIME (ITW)
      IF (IERR.NE.0) THEN
         POTERR = 101
         GO TO 980
         END IF
......
C                                        TVSCROL
C                                        user instructions
 500  WRITE (MSGTXT,1500)
      CALL MSGWRT (1)
      WRITE (MSGTXT,1505)
      CALL MSGWRT (1)
C                                        find channel(s) to scroll
C                                        scroll graphics too ?
```

```
      IC = ABS(TVCHAN) + EPS
      CALL DECBIT (NGRAY, IC, IC, J)
      ITEMP = 2 ** NGRAY
      IF (ABS(GRCHAN).GT.EPS) IC = IOR (IC, ITEMP)
      IF (IC.EQ.0) THEN
         IC = MOD (TVLIMG(1), ITEMP)
         IF (IC.NE.TVLIMG(1)) IC = IOR (IC, ITEMP)
         END IF
      IX = 0
      IY = 0
      RPOS(1) = (WINDTV(1) + WINDTV(3)) / 2
      RPOS(2) = (WINDTV(2) + WINDTV(4)) / 2
C                                        turn on cursor
      CALL YCURSE ('ONNN', F, F, RPOS, QUAD, IBUT, IERR)
      IF (IERR.NE.0) GO TO 900
C                                        force scroll
 510  CALL YSCROL (IC, IX, IY, T, IERR)
      IF (IERR.NE.0) GO TO 900
         PPOS(1) = RPOS(1)
         PPOS(2) = RPOS(2)
C                                        read until cursor moves
 520     CALL YCURSE ('READ', F, F, RPOS, QUAD, IBUT, IERR)
         IF (IERR.NE.0) GO TO 900
C                                        test for change
         CALL DLINTR (RPOS, IBUT, F, QUAD, PPOS, ITW, DOIT)
         IF (.NOT.DOIT) GO TO 520
C                                        cursor moved, change scroll
            IX = RPOS(1) - (WINDTV(1) + WINDTV(3)) / 2
            IY = RPOS(2) - (WINDTV(2) + WINDTV(4)) / 2
C                                        any button => done
            IF (IBUT.EQ.0) GO TO 510
               POTERR = 0
               GO TO 900
......
C                                        close down
C                                        cursor off, TV closed
 900  IF (BRANCH.GE.4) CALL YCURSE ('OFFF', F, F, RPOS, QUAD, IBUT,
     *   JERR)
 910  CALL TVCLOS (BUFFER, JERR)
```

10.3.7 Object location, window setting

Subroutine AU5 performs the verbs TVPOS, IMXY, IMPOS (see below), and TVNAME (via TVFIND) as well as a variety of status setting verbs. IMPOS is implemented as follows. It calls TVWHER to find the cursor position indicated by the user. Then it checks all enabled memories via YCREAD to see if there is an image displayed at that pixel position. Finally, it calls MP2SKY to set up the coordinate commons and get the primary positions and goes through some other messy stuff to display the results to the user.

```
      CALL TVOPEN (CATBLK, JERR)
      IF (JERR.NE.0) THEN
         POTERR = 101
         GO TO 980
         END IF
......
```

```
C                                       IMPOS
C                                       read cursor to get position
 600  CALL TVWHER (IPL, RPOS, IBUT, JERR)
      IF (JERR.NE.0) GO TO 975
C                                       Set output adverb = button #
      TVBUTT = IBUT
......
C                                       image pix -> map pixel pos
 625  IX = RPOS(1) + EPS
      IY = RPOS(2) + EPS
C                                       Find lowest plane with x,y
      IN2 = NGRAY + NGRAPH
      DO 630 IP = 1,IN2
C                                       skip off channels
         ITEMP = 2 ** (IP-1)
         IF (IAND (TVLIMG(IQUAD), ITEMP).EQ.0) GO TO 630
C                                       get image cat block
         CALL YCREAD (IP, IX, IY, CATBLK, IERR)
C                                       loop if x,y not in image
         IF (IERR.EQ.1) GO TO 630
         IF (IERR.EQ.0)  GO TO 650
         GO TO 975
 630     CONTINUE
C                                       x,y not in on image
      WRITE (MSGTXT,1630) IX, IY
      CALL MSGWRT (6)
      GO TO 900
C                                       image -> map positions
 650  CALL IMA2MP (RPOS, RPOS)
      WRITE (MSGTXT,1650) RPOS
      CALL MSGWRT (5)
......
C                                       map -> sky positions
 660  CONTINUE
         CALL MP2SKY (RPOS, SKYPOS, IERR)
         IF (IERR.NE.0) THEN
            WRITE (MSGTXT,1660) IERR
            CALL MSGWRT (6)
            GO TO 900
            END IF
C                                       3rd axis pairs w 1st or 2nd
         IF ((AXTYP.EQ.2) .OR. (AXTYP.EQ.3)) CALL AXSTRN (CTYP(3),
     *      SKYPOS(3), KLOCA, NCHLAB(1), SAXLAB(1))
C                                       Primary axes
C                                       Tell user results via MSGWRT.
         WRITE (MSGTXT,1661)
         ICH = 8
         DO 665 I = 1,2
            ITEMP1 = I-1
            CALL AXSTRN (CTYP(1,I), SKYPOS(I), ITEMP1, ILEN, RSTR)
            MSGTXT(ICH:ICH+ILEN-1) = RSTR(1:ILEN)
            ICH = ICH + ILEN + 2
 665        CONTINUE
         CALL MSGWRT (5)
```

```
C                                          Secondary axes values
      IF ((NCHLAB(1).LE.0) .AND. (NCHLAB(2).LE.0)) GO TO 900
         ICH = 8
         MSGTXT(ICH:) = ' '
         DO 670 I = 1,2
            IF (NCHLAB(I).LE.0) GO TO 670
               MSGTXT(ICH:ICH+NCHLAB(I)-1) = SAXLAB(I)(1:NCHLAB(I))
               ICH = ICH + NCHLAB(I) + 2
 670        CONTINUE
         CALL MSGWRT (5)
         GO TO 900
......
C                                          normal TV close
 900  CALL TVCLOS (CATBLK, JERR)
```

The interactive window setting verbs TVWIN, TVBOX, TVSLICE, and REBOX are initiated from subroutine AU5C and performed primarily by subroutine GRBOXS. This routine is another instance of interactivity via YCURSE and line drawing via IMVECT. It uses YCUCOR at the end to obtain the image catalog header and thence, to correct the cursor positions to map pixel locations. CURVALUE is an interactive verb which displays on a TV graphics channel the position and image value of the pixel currently under the TV cursor. It is implemented by subroutine AU6B. The image values are read from the original map files on disk, if possible, using MAPOPN, MINIT, and MDISK. However, the intensities of step wedges and temporary images (i.e., intermediate residual maps displayed by APCLN) are read from the TV memory via YIMGIO. The routine makes extensive use of IMCHAR and, although too long to reproduce here, is an interesting example of AIPS image plus TV coding.

10.3.8 Blotch Setting, Use

A "blotch" is a region within an image over which some action is to be performed. Pixels outside the blotch are ignored or have some alternative action performed on them. At present, AIPS has three functions which generate and use blotches: the verb TVSTAT which returns image statistics within the blotch area, the task BLANK which blanks out all pixels outside the blotch, and the task BLSUM which sums an image and every plane of a second image over the blotch areas. In all three, the user uses the TV cursor to set the vertices of one or more polygonal areas and the routines draw lines on a graphics plane between the vertices. When the user is done, the routines fill in the blotch areas on the TV graphics and then read and act on the map file. Subroutine AU6D implements TVSTAT for whatever image is visible on the TV, obtaining the polygons through subroutine GRPOLY. AU6D itself does the data reading, determination of whether a pixel is inside or outside the blotch, and the computation and display of the image statistics. Task BLANK uses internal subroutines BLNKTV and BLKTVF to display the image (via TVLOAD), allow transfer modification (via TVFIDL), to obtain the polygons (BLKTVF), and to use them to blank the output image (BLNKTV). BLSUM is similar to BLANK in the creation and handling of the blotch regions. The subroutine BLTFIL does the filling of the polygons on the TV graphics screen for TVSTAT, BLANK and BLSUM.

10.3.9 Roam

Roam is a mode of display which requires multiple grey-scale memories and the capability to do split screen and scroll. Adjacent portions of the image are loaded into separate image memories. Then the screen is split horizontally and/or vertically and the appropriate memories are enabled in each quadrant, each with the same scroll. This allows the user to view a screen-size portion of a rather larger image. By shifting the scroll and split point interactively, the user may select which portion is viewed. Roam is implemented in AIPS from the subroutine AU5A. This routine loads the image to the TV memories in a manner similar to TVLOD (above). However, it uses TVWIND to determine a much more complicated window and must itself play with windows further before calling TVLOAD. The interactive portion of the Roam is carried out by AU5A calling subroutine TVROAM. That routine can handle images of up to 1 x 4, 4 x 1, or 2 x 2 planes

and uses YCURSE for interactive input, YSCROL to set the scroll (identical for all planes), and YSPLIT to set the split point and enable the appropriate channels. A zoom option is also available.

10.3.10 Movie, Blink

The verbs TVMOVIE and TVCUBE use a very interesting algorithm implemented via subroutines AU5D and TVMOVI. A movie is a method of displaying a 3-dimensional image as a time sequence of 2-dimensional planes. Each grey-scale TV memory is subdivided into an n x n matrix of images of consecutive planes of the cube (where the allowed n are the available zoom factors). During the display phase, the zoom factor is set to n, so that only one plane is visible at a time. The zoom center is moved from frame to frame at a user controlled rate to simulate a movie. Subroutine AU5D determines which zoom factor and windows to use, zeros the grey-scale memories, loads the planes to the TV (via TVLOAD), transfers the LUT of the first TV memory to the other TV memories, draws border lines around each plane (via IMVECT), annotates each plane with the 3rd coordinate axis value, and puts a small pointer in the image as well. TVMOVI executes an interactive algorithm in which the cursor controls the frame rate and the buttons allow a single frame at a time mode and interactive enhancement of the LUTs (via IENHNS) or the OFM (via IMCCLR). The verb REMOVIE is also done by AU5D and TVMOVI using the stored parameters which describe how the movie was loaded to the TV memories (parameter TYPMOV in the /DTVC.INC/ common). TVCUBE and TVMOVIE differ in the placement of the sub-images on the TV screen. The former orders the images from left to right, top to bottom, an order which is good for viewing all together, without zoom. The latter adopts a funny order designed to minimize changes between frames in movie mode, making a smoother switch between frames.

The subroutines AU6A and TVBLNK implement the verbs TVBLINK and TVMBLINK. Blinking is simply enabling one grey-scale memory for a while, then disabling it and enabling another for a second period of time, then disabling the second channel and re-enabling the first, and so on. These two verbs allow manual as well as timed switching between the two planes and transfer function modification via the subroutine IENHNS (see above). Image comparison may also be done by comparing images from different image memories in halves or quarters of the screen. The verb TVSPLIT implements an interactive algorithm in subroutine AU6A.

10.3.11 Tasks

There are three tasks which use a wide range of television functions in order to edit visibility data under interactive control of the user. All of these implement a menu written to a TV graphics screen to allow the user to select among many separate functions. All of the standard TVFIDLLE and TVTRANS enhancements are present. For window setting, an especially friendly version of GRBOXS is coded into the tasks. These tasks are:

1. TVFLG grids the data in baseline-time order displaying the visibility amplitude, phase, or amplitude rms as grey levels, with baseline on the horizontal axis and time in a semi-regular grid on the vertical axis. The user may select which IF, Stokes, or channel to display. The cursor is used to select the data to be flagged, as well as to make selections of functions to perform and enhancements.

2. SPFLG is similar to TVFLG, but grids the data with spectral channel on the X axis instead of baseline number. The user may select which baseline is displayed.

3. IBLED is also an interactive editor for baseline-oriented data. It uses the TV mostly as a graphics device to plot the data from a user-selected baseline. Time is on the horizontal axis, and amplitude is on the vertical axis. The cursor is used to select the data to be flagged and to do the selection of operations.

10.3.12 Non-Standard Tasks

There are a number of tasks in AIPS which are seriously non-standard in their coding and in their use of various devices. Among these are several which use the TV display. We will list them here briefly.

Programmers should not use these tasks as models of how to code in AIPS and should not assume that they can even be made to run on non-VMS, non-IIS systems.

1. IMLHS uses up to 3 maps to create a false color image on the TV. It uses the first map to modulate the brightness of the image, the 2nd to modulate the hue and the 3rd to modulate the saturation. If any of the images are omitted the corresponding parameter is set to a constant. (Note: verb TVHUEINT is standard and does a similar function with two images.)

2. TVHLD loads up to 13-bit image to two TV memories and performs an interactive histogram equalization of the display. Can feed the result back to a 3rd TV memory. This task uses YRHIST, YALUCT, YFDBCK, YIFM, and the dual-channel mode of the IIS and will be hard to implement on TV display devices other than the IIS.

3. TVHXF does an interactive histogram equalization of the image which is currently displayed. This task uses YRHIST which is currently IIS specific. However, a TV-independent (but SLOW) YRHIST can be coded if someone wishes to do the work.

10.4 Includes

10.4.1 DTVC.INC

```
C                                                               Include DTVC.
      INTEGER   NGRAY, NGRAPH, NIMAGE, MAXXTV(2), MAXINT, LUTOUT,
     *   OFMINP, OFMOUT, SCXINC, SCYINC, MXZOOM, CSIZTV(2), TYPSPL,
     *   TVALUS, TVXMOD, TVYMOD, ISUNUM,
     *   TVDUMS(10),
     *   TVZOOM(3), TVSCRX(16), TVSCRY(16), TVLIMG(4), TVSPLT(2),
     *   TVSPLM, TVSPLC, TYPMOV(16), WINDTV(4), TVDUM2(4), YBUFF(160)
      COMMON /TVCHAR/ NGRAY, NGRAPH, NIMAGE, MAXXTV, MAXINT, LUTOUT,
     *   OFMINP, OFMOUT, SCXINC, SCYINC, MXZOOM, CSIZTV, TYPSPL,
     *   TVALUS, TVXMOD, TVYMOD, ISUNUM,                 TVDUMS,
     *   TVZOOM, TVSCRX, TVSCRY, TVLIMG, TVSPLT, TVSPLM, TVSPLC,
     *   TYPMOV, WINDTV, TVDUM2, YBUFF
C                                                               End DTVC.
```

10.4.2 DTVD.INC

```
C                                                               Include DTVD.
      INTEGER   TVLUN, TVIND, TVLUN2, TVIND2, TVBFNO
      LOGICAL   TVMAP
      COMMON /TVDEV/ TVMAP, TVLUN, TVIND, TVLUN2, TVIND2, TVBFNO
C                                                               End DTVD.
```

10.4.3 YDEA.INC

```
C                                               Begin YTPARM.INC
C               Parameter definitions specifying positions
C               in YBUFF (c.f. DTVC.INC) which correspond
C               to various registers in a DeAnza TV device.
      INTEGER   INTREG, IRBYTE, MEMREG, MRBYTE, VOCREG, VCBYTE,
     *          FCR,    FCBYTE, CURREG, CRBYTE, CHANOW, CHBYTE
      PARAMETER (INTREG = 1,          IRBYTE = 48,
     *           MEMREG = INTREG+24,  MRBYTE = 8,
     *           VOCREG = MEMREG+64 , VCBYTE = 8,
     *           FCR    = VOCREG+4 ,  FCBYTE = 16,
```

```
     *           CURREG = FCR   +8 ,  CRBYTE = 14,
     *           CHANOW = CURREG+7 ,  CHBYTE = 24)
C                                       Positions in INTERFACE REGISTERS
C                                       Relative to CONTROL REGIST (#10)
      INTEGER   CONTRL, RES, FG, BG, XR, XT, XMIN, XMAX,
     *          XAMIN, XAMAX, DX, XTEMP, YR, YT,   YMIN,
     *          YMAX, YAMIN, YAMAX, DY, YTEMP, CMRO, CMROA
      PARAMETER (CONTRL=INTREG,   RES=INTREG+1,    FG=INTREG+2,
     *           BG=INTREG+3  ,   XR= INTREG+6,    XT=INTREG+7,
     *           XMIN=INTREG+8,   XMAX=INTREG+9,   XAMIN=INTREG+10,
     *           XAMAX=INTREG+11, DX=INTREG+12,    XTEMP=INTREG+13,
     *           YR=INTREG+14,    YT=INTREG+15,    YMIN=INTREG+16,
     *           YMAX=INTREG+17,  YAMIN=INTREG+18, YAMAX=INTREG+19,
     *           DY=INTREG+20,    YTEMP=INTREG+21, CMRO=INTREG+22,
     *           CMROA=INTREG+23)
C                                       Positions in MEMORY REGISTERS
      INTEGER   XSCZ, YSCZ, BITPL, MEMLUT
      PARAMETER (XSCZ=MEMREG,    YSCZ = MEMREG+1,
     *           BITPL=MEMREG+2, MEMLUT=MEMREG+3)
C                                       Positions in VOC REGISTERS
      INTEGER   XVSP, YVSP, VOCCON, VOCLUT
      PARAMETER (XVSP=VOCREG,     YVSP=VOCREG+1,
     *           VOCCON=VOCREG+2, VOCLUT=VOCREG+3)
C                                       Positions in Cursor registers
      INTEGER   CURX1, CURY1, CURX2, CURY2, CURCON, CURLUT, CURBLI
      PARAMETER (CURX1=CURREG,   CURY1=CURREG+1,   CURX2=CURREG+2,
     *           CURY2=CURREG+3, CURCON=CURREG+4,  CURLUT=CURREG+5,
     *           CURBLI=CURREG+6)
```

10.5 Y-Routine Precursor Remarks

10.5.1 Level 0

YCHRW

writes characters into image planes of the TV. The format is 5 x 7 or a multiple thereof with one or more blanks all around. The net is set to match CSIZTV if possible. We recommend CSIZTV = 7, 9 for TVs of size around 512 square and CSIZTV = 14, 22 or so for TVs of size around 1024 square. This version will work on all TVs which allow horizontal writing to the right. It is a Y routine to allow for hardware character generators on the TV.

```
YCHRW (CHAN, X, Y, STRING, SCRTCH, IERR)
Inputs:
   CHAN    I       channel select  (1 to NGRAY + NGRAPH)
   X       I       X position lower left corner first char.
   Y       I       Y position lower left corner first char.
   STRING  C*(*)   character string - length passed from Fortran
Output:
   SCRTCH  I(>)    scratch buffer (dim = 14*count+8 < MAXXTV(1))
   IERR    I       error code of Z...XF:0 - ok
                                          2 - input error
```

YCINIT

Initialize image catalog for plane IPLANE - TK now done with TKCATL

```
YCINIT (IPLANE, BUFF)
Input:
   IPLANE   I        Image plane to initialize
Output:
   BUFF     I(256)   Working buffer
```

YCNECT

Writes a line segment on the TV. This version will work on all TVs. It is called a Y routine to allow the use of hardware vector generators on those TVs equiped with them.

```
YCNECT (X1, Y1, X2, Y2, IC, BUFFER, IERR)
Inputs:
   X1       I       start X position
   Y1       I       start Y position
   X2       I       end X position
   Y2       I       end Y position
   IC       I       Channel (1 to NGRAY+NGRAPH)
   BUFFER   I(*)    Buffer contains desired intensity (size > max
                    horizontal or vertical line, e.g., 1280)
Output:
   IERR     I       error code : 0 => ok
```

YCOVER

Checks to see if there are partially replaced images in any of the TV planes currently visible by quadrant

```
YCOVER (OVER, BUF, IERR)
Outputs:
   OVER   L(4)      T => there are in quadr. I
   BUF    I(512)    scratch
   IERR   I         Error code: 0 => ok, other catlg IO error
```

YCREAD

Read image catalog block into CATBLK - TV only (TK in TKCATL)

```
YCREAD (IPLANE, IX, IY, CATBLK ,IERR)
Inputs:
   IPLANE  I        plane containing image whose block is wanted
   IX      I        X pixel coordinate of a point within image
   IY      I        Y pixel coordinate of point within image
Outputs:
   CATBLK  I(256)   Image catalog block
   IERR    I        error codes: 0 => ok
                       1 => IX, IY lies outside image
                       2 => Catalog i/o errors
                       3 => refers to TK device
```

YCUCOR

Takes a cursor position (corrected for zoom, but not scroll) corrects it for scroll, determines the quadrant of the TV, and gets the corresponding image header in common /MAPHDR/ and returns the image coordinates. NOTE WELL: RPOS ON INPUT MUST BE CORRECTED FOR ZOOM AND NOT SCROLL. To get this from a raw TV position call YCURSE with OPCODE 'FXIT' (or, of course, 'READ') and quadrant set to -1. **********

```
YCUCOR (RPOS, QUAD, CORN, IERR)
Inputs:
   RPOS    R(2)    X,Y screen pos before  zoom & scroll
Output:
   QUAD    I       TV quadrant to use for scrolls
                   Out: if in=-1, no scroll, else find quadrant
                        (needs real TV pos)
   CORN    R(7)    Image coordinates (pixels)
   IERR    I       error code of Z...XF : 0 - ok
                                          2 - input error
```

YCURSE

Reads cursor positions and controls the blink and visibility of the TV cursor.

```
YCURSE (OP, WAIT, CORR, RPOS, QUAD, BUTTON, IERR)
Inputs:
   OP      C*4     'READ' read cursor position
                   'ONNN' place cursor at RPOS & leave on
                   'OFFF' turn cursor off
                   'BLNK' reverse sense of cursor blink
                   'FXIT' fix RPOS for zoom scroll, no IO
   WAIT    L       wait for event; then return RPOS & BUTTON
                   (done on all OPs)
   CORR    L       T => correct RPOS for zoom & scroll
In/Out:
   RPOS    R(2)    X,Y screen pos before  zoom & scroll
   QUAD    I       TV quadrant to use for scrolls
                      In: if <1 >4, no scroll
                      Out: if in=-1, no scroll, else find
                           quadrant (needs real TV pos)
Output:
   BUTTON  I       event # (0 none, 1-7 low buttons,
                   8-15 the "quit" button)
   IERR    I       error code of Z...XF : 0 - ok
                                          2 - input error
```

YCWRIT

Write image catalog block in CATBLK into image catalog

```
YCWRIT (IPLANE, IMAWIN, CATBLK, BUFF, IERR)
Inputs:
   IPLANE   I        image plane involved
   IMAWIN   I(4)     Corners of image on screen
```

```
    CATBLK   I(256)   Image catalog block
Outputs:
    BUFF     I(256)   working buffer
    IERR     I        error code: 0 => ok
                         1 => no room in catalog
                         2 => IO problems
```

YFILL

Will write a constant in a given rectangle in a given graphics or image plane. It will use fast methods if full screen requested with IVAL 0.

```
YFILL (CHAN, IX0, IY0, IXT, IYT, IVAL, IBLK, IERR)
Inputs:
   CHAN     I   Channel (1 to NGRAY+NGRAPH)
   IX0      I   lower left X pixel (1 relative) of rectangle
   IY0      I   lower left Y pixel of rectangle.
   IXT      I   top right X pixel of rectangle.
   IYT      I   top right Y pixel of rectangle.
   IVAL     I   desired value: for graphics = 0 or 1
                               for grey scale = 0 - MAXINT
In/out:
   IBLK     I(IXT-IX0+1)   work buffer.
Output:
   IERR     I   error code of Z...XF: 0 ok, 2 input error
```

YFIND

Determines which of the visible TV images the user wishes to select. The TV must already be open.

```
YFIND (MAXPL, TYPE, IPL, UNIQUE, CATBLK, SCRTCH, IERR)
Inputs:
   MAXPL   I        Highest plane number allowed (i.e. do graphics
                    count?)
   TYPE    C*2      2-char image type to restrict search
Output:
   IPL     I        Plane number found
   UNIQUE  L        T => only one image visible now
                        (all types except zeroed ones ('ZZ'))
   CATBLK  I(256)   Image catalog block found
   SCRTCH  I(256)   Scratch buffer
   IERR    I        Error code: 0 => ok
                       1 => no image
                       2 => IO error in image catalog
                       3 => TV error
                      10 => > 1 image of requested type
```

YLOCAT

Locates a set of image pixel positions on the TV for a specified image using only those grey planes that are turned on.

```
YLOCAT (NP, XP, YP, NAME, CLASS, SEQ, DISK, PTYP, IX,
  *   IY, IQ, SCRTCH, IERR)
Inputs:
   NP       I          Number of pixel positions
   XP       R(NP)      X image pixel positions
   YP       R(NP)      Y image pixel positions
   NAME     C*12       image name (name) '     ' => any
   CLASS    C*6        image name (class) '     ' => any
   SEQ      I          Image name (sequence #)  0 => any
   DISK     I          Image file disk 0 => any
   PTYP     C*2        Image type  '  ' => any
Output:
   IX       I(NP)      TV x positions
   IY       I(NP)      TV y positions
   IQ       I(NP)      TV channels  0 => none this position
   IERR     I          Error code: 0 -> ok
                          2 -> input error
                          3 -> IO error
                         11 -> some positions bad
                         12 -> no positions found
Uses common /MAPHDR/ results unpredictable except on IERR = 0
then = image catlg header of last position found.
```

YLOWON

Returns a zero-relative least bit on number up to NGRAY. If MASK=0, it returns 0. If MASK $> 2 *$ $(NGRAY - 1)$ it returns NGRAY. Since some TVs can't have more than one channel on at a time in each color, determine the lowest channel that is flagged ON in MASK, and return it (zero relative) in ICHAN. If MASK is clear, set ICHAN to be a 0.

```
YLOWON (MASK, ICHAN)
Inputs:
   MASK     I   IIS type channel mask
Output:
   ICHAN    I   # of lowest bit set in MASK
```

YSLECT

Enables and disables gray and graphics planes

```
YSLECT (OP, CHAN, COLOR, BUFFER, IERR)
Inputs:
   OP       C*4       'ONNN' or 'OFFF'
   CHAN     I         channel number ( 1 to NGRAY+NGRAPH)
   COLOR    I         0 - all,   1,2,3 = R,G,B, resp.
Output:
   BUFFER   I(256)    scratch buffer (for graphics only)
   IERR     I         error code of Z...XF: 0 - ok, 2 - input error
YSLECT sets TVLIMG in the TV device parms common /TVDEV/
```

YTCOMP

Check whether a value in a soft register has to be changed. If so, change it and set a flag indicating that the hard register must be updated too.

```
YTCOMP (OLD, NEW, UPDATE)
Inputs:
   OLD      I   Current value in soft register
   NEW      I   New value
Output:
   OLD      I   Put new value here too if necessary
   UPDATE   L   Set true if update is needed, else leave alone
```

YTVCLS

Closes the TV device and the TV status disk file, updating the information on the disk. Actual device call done by YTVCL2.

```
YTVCLS (BUF, IERR)
Outputs:
   BUF   I(256)     Scratch buffer
   IERR  I          Error code : 0 => ok
                       else as returned by ZFIO
                       11 => close disk error
                       12 => close device error
```

YWINDO

Reads the current viewport into the TV memory. It is hoped that someday we will also offer writing to force the size.

```
YWINDO (OPER, WIND, IERR)
Inputs:
   OPER   C*4    'READ', 'WRIT'
In/out:
   WIND   I(4)   BLC x,y, TRC x,y of window in TV pixels
                 In: desired window ('WRIT' only)
                 Out: actual window given
Output:
   IERR    I     error code of Z...XF: 0 -> ok, 2 -> input error
                 -1 => on WRIT, device not windowing type
Generic version - returns full memory size always and IERR = -1
on WRIT.
```

YTVOPN

Opens the TV including TV lock/parameter file and reads the parameters, placing them in commons.

```
YTVOPN (BUF, IERR)
Outputs:
   BUF   I(256)   Scratch buffer
   IERR  I        Error return from YTVOP2
                     = 10 TV unavailable to this version
```

10.5.2 Level 1

YCRCTL

Reads/writes the cursor/trackball control register of TV

```
YCRCTL (OP, ON, X, Y, LINKX, LINKY, RBLINK, BUTTON,
  *   VRTRTC, IERR)
Inputs:
   OP       C*4    'READ' from TV or 'WRIT' to TV
   VRTRTC   L      T => do on vertical retrace only
In/out:
   ON       L      T => cursor visible, F => off
   X        I      X position cursor center (1-512, 1 => LHS)
   Y        I      Y position cursor center (1-512, 1 => bot)
   LINKX    L      T => trackball moves cursor in X
   LINKY    L      T => trackball moves cursor in Y
   RBLINK   I      rate of cursor blink: 0-3 no-fast blink
Output:
   BUTTON   I      button value (0 - 15)
   IERR     I      error code of Z...XF : 0 => ok
                                          2 => input error
```

YGRAPH

Is used to turn graphics overlay planes on and off by altering the graphics color look up table. The color pattern is:

```
CHAN = 1    insert   yellow            drawing plots
       2    insert   green+.05 red     axis labels
       3    insert   blue + 0.6 green  blotch
                     + red
       4    insert   black             label backgrounds
       5-7  add      nothing           null channels
       8    insert   purple            cursor

YGRAPH (OP, CHAN, SCRTCH, IERR)
Inputs:
   OP       C*4        'OHNN' or 'OFFF'
   CHAN     I          channel number (1 - 8)
Output:
   SCRTCH   I(256)     scratch buffer
   IERR     I          error code of Z...XF: 0 => ok
                                             2 => input error
```

YIMGIO

Reads/writes a line of image data to the TV screen. For graphics overlay planes, the data are solely 0's and 1's in the least significant bit of IMAGE after a READ. For 'WRIT', all bits of each word should be equal (i.e. all 1's or all 0's for graphics). NOTE***** on 'WRIT', the buffer may be altered by this routine for some IANGLs.

```
YIMGIO (OP, CHAN, X, Y, IANGL, NPIX, IMAGE, IERR)
Inputs:
```

```
   OP      C*4       'READ' from TV or 'WRIT' to TV
   CHAN    I         channel number (1 to NGRAY+NGRAPH)
   X       I         start pixel position
   Y       I         end pixel position
   IANGL   I         = 0 => horizontal (to right)
                     = 1 => vertical (up the screen)
                     = 2 => horizontal (to left)
                     = 3 => vertical (down the screen)
   NPIX    I         number of pixels
In/Out:
   IMAGE   I(NPIX)   data (only no header)
Output:
   IERR    I         error code of Z...XF - 0 => ok
                                            2 => input err
```

YINIT

Initializes the TV subunits: doing everything

```
YINIT (SCRTCH, IERR)
Output:
   SCRTCH  I(1024)    scratch buffer
   IERR    I          error code of Z...XF - 0 => ok
                                             2 => input error
```

YLUT

Reads/writes full channel look up tables to TV.

```
YLUT (OP, CHANNL, COLOR, VRTRTC, LUT, IERR)
Inputs:
   OP      C*4    'READ' from TV, 'WRIT' to TV
   CHANNL  I      channel select bit mask
   COLOR   I      color select bit mask (RGB <-> 421)
   VRTRTC  L      T => do it only during vertical retrace
In/Out:
   LUT     I(*)   look up table (dimension = MAXINT+1, values to
                  LUTOUT are used)
Out:
   IERR    I      error code of Z...XF : 0 => ok, 2 => input error
```

YOFM

Reads/writes full OFM look up tables to TV.

```
YOFM (OP, COLOR, VRTRTC, OFM, IERR)
Inputs:
   OP      C*4    'READ' from TV, 'WRIT' to TV
   COLOR   I      color select bit mask (RGB <-> 421)
   VRTRTC  L      T => do it only during vertical retrace
In/Out:
```

```
   OFM      I(*)   look up table (dimension = OFMINP+1, values to
                   OFMOUT used)
Output:
   IERR     I      error code of Z...XF : 0 => ok, 2 => input error
```

YSCROL

Writes the scroll registers on the TV.

```
YSCROL (CHANNL, SCROLX, SCROLY, VRTRTC, IERR)
Inputs:
   CHANNL   I   bit map channel select: bits 1-NGRAY gray channels,
                bit NGRAY+1 => all graphics
   VRTRTC   L   T => do it on vertical retrace only
In/Out:
   SCROLX   I   amount of X scroll (>0 to right)
   SCROLY   I   amount of Y scroll (>0 upwards)
Output:
   IERR     I   error from Z...XF : 0 => ok, 2 => input error
YSCROL updates the scroll variables in /TVDEV/ common
```

YSPLIT

Reads/writes the look up table/ split screen control registers of the TV - turns channels on/off by quadrant

```
YSPLIT (OP, XSPLT , YSPLT , RCHANS, GCHANS, BCHANS,
  *   VRTRTC, IERR)
Inputs:
   OP       C*4    'READ' from TV, 'WRIT' to TV
   VRTRTC   L      T => do on vertical retrace only
In/Out:
   XSPLT    I      X position of split (1-512, 1 => LHS)
   YSPLT    I      Y position of split (1-512, 1 => bot)
   RCHANS   I(4)   chan select bit mask 4 quadrants : red
   GCHANS   I(4)   chan select bit mask 4 quadrants : green
   BCHANS   I(4)   chan select bit mask 4 quadrants : blue
Output:
   IERR     I      error code of Z...XF: 0 => ok, 2 => input error
Quadrants are numbered CCW from top right!!!!
```

YTVCIN

Initializes the common which describes the characteristics of the interactive display devices and the common which has the current status parameters of the TV.
NOTE: These are default values only. They are reset to the current true values by a call to TVOPEN.
NOTE: YTVCIN resets the common values of TVZOOM and TVscroll, but does not call the TV routines to force these to be true. A separate call to YINIT or YZOOMC and YSCROL is needed.

```
   YTVCIN
(no arguments)
```

YTVCL2

Closes TV DEVICE associated with LUN removing any EXCLusive use state and clears up the FTAB.

```
YTVCL2 (LUN, IND, IERR)
Inputs:
   LUN    I   logical unit number
   IND    I   pointer into FTAB
Output:
   IERR   I   error code: 0 -> no error
                  1 -> Deaccess or Deassign error
                  2 -> file already closed in FTAB
                  3 -> both errors
                  4 -> erroneous LUN
```

YTVMC

Issues a "master clear" to the TV. This resets the TV I/O system (if necessary) to expect a command record next. YTVMC gets all needed parameters from the TV device common. The TV must already be open.

```
   YTVMC
(no arguments)
```

YTVOP2

Performs a system "OPEN" on the TV device. It is a Y routine in order to call the appropriate Z routine only.

```
YTVOP2 (LUN, IND, IERR)
Inputs:
   LUN    I   Logical unit number to use
Output:
   IND    I   Pointer to FTAB entry for open device
   IERR   I   Error code: 0 => ok
                  1 = LUN already in use
                  2 = file not found
                  3 = volume not found
                  4 = excl requested but not available
                  5 = no room for lun
                  6 = other open errors
```

YZERO

Fills an TV TV memory plane with zeros the fast way.

```
YZERO (CHAN, IERR)
Inputs:
   CHAN   I     channel # ( 1 - NGRAY+NGRAPH), 0 => all
Outputs:
   IERR   I     error code of Z...XF: 0 - ok, 2 - input error
```

YZOOMC

Writes (ONLY!!!!) the zoom control registers of the TV

```
YZOOMC (MAG, XZOOM, YZOOM, VRTRTC, IERR)
Inputs:
   MAG      I      0-3 for magnification 1,2,4,8 times, resp.
   XZOOM    I      X center of expansion (1-512, 1 => LHS)
   YZOOM    I      Y center of expansion (1-512, 1 => bot)
   VRTRTC   L      Do on vertical retrace only?
Output:
   IERR     I      error code of Z...XF: 0 -> ok, 2 -> input error
YZOOMC updates the /TVDEV/ common TVZOOM parameter
```

10.5.3 Level 2 (Used as level 1 in non-standard tasks)

YALUCT

Reads / writes the TV arithmetic logic unit control registers. The actual feedback-ALU computation is performed only upon a call to YFDBCK.

```
YALUCT (OP, ARMODE, BFUNC, NFUNC, CONSTS, OUTSEL,
  *   EXTOFM, ESHIFT, SNIFT, CARYIN, CARRY, EQUAL, IERR)
Inputs:
   OP       C*4      'READ' from TV or 'WRIT' to TV
In/Out:
   ARMODE   L        T => arithmetic mode F => logic mode
   BFUNC    I        function number (1-16) in blotch
   NFUNC    I        function number (1-16) outside blotch
   CONSTS   I(8)     constant array (may select as ALU output)
   OUTSEL   I(8)     lookup table selects output based on carry
                     (lsb), equal, ROI (msb) input. values -
                     0 - 7 : constants 1 - 8
                     8     : accumulator channel pair
                     9     : selected OFM
                     10    : ALU
                     11    : external
   EXTOFM   L        T => extend sign of OFM on input to ALU
   ESHIFT   L        T => extend sign of ALU output if SHIFT
   SHIFT    L        T => right shift ALU output
   CARYIN   L        T => add one to arithmetic results
Output:
   CARRY    L        T => carry condition occurred in frame
   EQUAL    L        T => equal condition occurred in frame
   IERR     I        error code of Z...XF : 0 - ok
                                           2 - input error
```

YCONST

Reads/writes the constants which are added to the 3 sum channels on the TV.

```
YCONST (OP, RCONST, GCONST, BCONST, VRTRTC, IERR)
Inputs:
   OP       C*4    'READ' from TV, 'WRIT' to TV
```

```
In/out:
   RCONST  I     constant for red channel
   GCONST  I     constant for green channel
   BCONST  I     constant for blue channel
   VRTRTC  L     Vertical retrace - do only on ?
Output:
   IERR    I     error code of Z...XF : 0 => ok
```

YFDBCK

Sends a feedback command to the TV

```
YFDBCK (COLOR, CHANNL, BITPL, PIXOFF, BYPIFM, EXTERN,
  *   ZERO, ACCUM, ADDWRT, IERR)
Inputs:
   COLOR   I    bit map of color to be fedback (RGB = 4,2,1)
   CHANNL  I    bit map of channels to receive feedback
   BITPL   I    bit map of bit planes to receive feedback
   PIXOFF  I    offset fedback image to left by 0 - 1 pixels
   BYPIFM  L    F => image goes thru IFM lookup before store
   EXTERN  L    T => image from external input (iedigitizer)
   ZERO    L    T => feed back all zeros
   ACCUM   L    T => use 16-bit accumulator mode then CHANNL must
                     give even-odd pair lsbyte goes to even (lower)
                     # channel
   ADDWRT  L    T => additive write  F => replace old data
Outputs:
   IERR    I    error code of Z...XF: 0 -> ok
                                      2 -> input error
```

YIFM

Reads/writes a section of TV input function memory. This look up table takes 13 bits in and gives 8 bits out.

```
YIFM (OP, START, COUNT, PACK, VRTRTC, IFM, IERR)
Inputs:
   OP      C*4    'READ' from TV or 'WRIT' to TV
   START   I      start address of IFM (1 - 8192)
   COUNT   I      # elements of IFM to transfer (1-8192)
   PACK    L      T => 2 values/word, F => 1 value/word
   VRTRTC  L      T => do it only on vertical retrace
In/Out:
   IFM     I(*)   function values (0-255)
Output:
   IERR    I      error code of Z...XF: 0 - ok
                                        2 - input error
```

YMNMAX

Reads the min and max values put out by the 3 summers (before application of constants, shifts and OFM) from the TV

```
YMNMAX (RMIN, RMAX, GMIN, GMAX, BMIN, BMAX, VRTRTC,
  *   IERR)
Inputs:
   VRTRTC  L    do it on vertical retrace only
Output:
   RMIN    I    red minimum
   RMAX    I    red maximum
   GMIN    I    green minimum
   GMAX    I    green maximum
   BMIN    I    blue minimum
   BMAX    I    blue maximum
   IERR    I    error code of Z...XF : 0 => ok, 2 => input error
```

YRHIST

Reads the histogram of the output of a selected OFM of the TV. Intensity values 0 through OFMOUT may be read.

```
YRHIST (MODE, COLOR, INITI, NINT, HISTOG, IERR)
Inputs:
   MODE    I          selects area to histogram: 0 blotch,
                      1 not blotch, 2 all, 3 external blotch
   COLOR   I          bit map of single color (RGB - 4,2,1)
   INITI   I          first intensity to histo (1 - 1024)
   NINT    I          # values to get
Output:
   HISTOG  I(NINT)    histogram
   IERR    I          error code of Z...XF : 0 => ok, 2 => input err
```

YSHIFT

Reads/writes the TV shift registers - which shift the 13-bit output of the adders before entry into the 10-bit OFM

```
YSHIFT (OP, SHIFTR, SHIFTG, SHIFTB, VRTRTC, IERR)
Inputs:
   OP      C*4    'READ' from TV or 'WRIT' to TV
   VRTRTC  L      T => do on vertical retrace only
In/Out:
   SHIFTR  I      # bits to shift (right) red channel
   SHIFTG  I      # bits to shift green channel
   SHIFTB  I      # bits to shift blue channel
Output:
   IERR    I      error code of Z...XF : 0 - ok, 2 - input error
```

10.5.4 Level 3 (selected ones of some general interest)

YBUTON

Reads the state of the Button Buffer of the TV display.

```
YBUTON (LBUT, BUTTON, IERR)
Input:
   LBUT     I   Button value returned from an initial status read
                (Send -1 if none done).
Output:
   BUTTON   I   Button value: 0, 1, 2, 4, or 8
   IERR     I   Error: 0 ok, 1, bad IO
```

YGGRAM

Reads/writes the TV graphics color assignment RAM. The data are packed in this look up table for colors as:

```
   bit 15 (msb)   :   1 - graphics replace 0 - add to image
   bits 10-14     :   red value for this graphics value
   bits  5- 9     :   green
   bits  0- 4     :   blue

YGGRAM (OP, VRTRTC, RGBBUF, IERR)
Inputs:
   OP       C*4        'READ' from TV or 'WRIT' to TV
   VRTRTC   L          T => do it only on vertical retrace
In/Out:
   RGBBUF   I(256)     data array
Out:
   IERR     I          error code of Z...XF : 0 => ok
```

YGRAFE

Reads/writes the TV graphics control register. This version does not support all the TV options: it forces the black and white status monitor on and refuses to allow the disable video, graphics, and cursor options.

```
YGRAFE (OP, BLOTCH, STATUS, VRTRTC, IERR)
Inputs:
   OP       C*4    'READ' from TV or 'WRIT' to TV
   VRTRTC   L      T => do it only during vertical retrace
In/Out:
   BLOTCH   I      graphics plane used as blotch (1 - 7)
   STATUS   I      graphics plane used as status (1 - 7)
Output:
   IERR     I      error code of Z...XF : 0 => ok
                                          2 => input error
```

YSTCUR

Reads/writes the TV cursor array which has the pattern exhibited when the cursor is visible. NOTE: if more than one row is read/written at a time, then the Y value decreases!!!

```
YSTCUR (OP, X, Y, NPOINT, PACK, VRTRTC, BUFFER, IERR)
Inputs:
   OP       C*4     'READ' from TV or 'WRIT' to TV
```

```
   X       I       initial X position (1-64, 1 => LHS)
   Y       I       initial Y position (1-64, 1 => bot)
   NPOINT  I       # pixel values in BUFFER
   PACK    L       T => 2 values/word, F => 1 value/word
   VRTRTC  L       T => do it on vertical retrace only
In/Out:
   BUFFER  I(*)    data array (lsb's used only)
Output:
   IERR    I       error code of Z...XF : 0 => ok, 2 => input error
```

10.6 Selected Applications Subroutines

10.6.1 Basic TV I/O Operations

DLINTR

Is called by interactive routines to delay the task when nothing is happening (i.e. the user is thinking or out to lunch.) It also prevents cursor wrap around.

```
DLINTR (RP, IEV, DOCOR, QUAD, PP, IT, DOIT)
Inputs:
   IEV     I       not = 0  => event has occurred
   DOCOR   L       Scroll correction parameter for YCURSE
   QUAD    I       quadrant parameter for YCURSE
In/out:
   RP      R(2)    cursor position read (fixed on wraps)
   PP      R(2)    previous cursor position
   IT      I(3)    time of last action
Output:
   DOIT    L       T => something has happened.
```

IMANOT

Is used to annotate an image by writing the string into the lettering plane (usually graphics plane 2) and, if possible writing a block of ones NEDGE pixels wider than the string into graphics plane 4 to force a black background: $NEDGE = 2 * MAXXTV/512$

```
IMANOT (OP, X, Y, IANGL, CENTER, STRING, SCRTCH, IERR)
Inputs:
   OP      C*4     'ONNN' enables the 2 graphics planes
                   'OFFF' disables the 2 planes
                   'INIT' zeros and enables the 2 planes
                   'WRIT' writes strings to the planes
   X       I       X position of string
   Y       I       Y position of string
   IANGL   I       0 - horizontal, 3 - vertical (DOWN)
   CENTER  I       0 - XY are lower left first character
                   1 - XY are center of string
                   2 - XY are top right of last character
   STRING  C*(*)   character string: length from LEN (STRING)
Output:
   SCRTCH  I(*)    scratch buffer (> 1280)
   IERR    I       error code of ZM70XF : 0 - ok
                                          2 - input error
```

IMVECT

Writes a connected sequence of line segments on a TV channel calling YCNECT

```
IMVECT (OP, CHAN, COUNT, XDATA, YDATA, SCRTCH, IERR)
Inputs:
   OP      C*4      'ONNN' line of ones (max intensity)
                    'OFFF' line of zeros (min intensity)
   CHAN    I        channel number (1 to NGRAY+NGRAPH)
   COUNT   I        number of X,Y pairs ( > 1)
   XDATA   I(COUNT)      X coordinates X1,X2,...
   YDATA   I(COUNT)      Y coordinates Y1,Y2,...
Output:
   SCRTCH  I(*)     scratch buffer (size MAXXTV)
   IERR    I        error code of ZM70XF - 0 => ok;  2 => input error
```

TVCLOS

Closes the TV device and the TV status disk file, updating the information on the disk. Does this all by call to YTVCLS.

```
TVCLOS (BUF, IERR)
Outputs:
   BUF   I(256)     Scratch buffer
   IERR  I          Error code : 0 => ok,   2 => not open in parms,
                    else as returned by YTVCLS
```

TVFIND

Determines which of the visible TV images the user wishes to select. If there is more than one visible image, it requires the user to point at it with the cursor. The TV must already be open. This routine is available only from program AIPS.

```
TVFIND (MAXPL, TYPE, IPL, UNIQUE, CATBLK, SCRTCH,
  *   IERR)
Inputs:
   MAXPL   I        Highest plane number allowed (i.e. do graphics
                    planes count?)
   TYPE    C*2      2-char image type to restrict search
Output:
   IPL     I        Plane number found
   UNIQUE  L        T => only one image visible now
                    (all types except zeroed ones ('ZZ'))
   CATBLK  I(256)   Image catalog block found
   SCRTCH  I(256)   Scratch buffer
   IERR    I        Error code: 0 => ok
                       1 => no image
                       2 => IO error in image catalog
                       3 => TV error
```

TVOPEN

Open the TV, passing pointers through common/DTVC.INC/. Almost all except error checking done by YTVOPN these days.

```
TVOPEN (BUF, IERR)
Outputs:
   BUF   I(256)     Scratch buffer
   IERR  I          Error return from YTVOPN
                    = 10 TV unavailable to this version
```

TVWHER

Is the routine to use if you want the user to point at something on the TV screen. It turns on the cursor, waits for a button push, determines which quadrant the event occurred in, and returns the quadrant number and the TV coordinates of the event corrected for the scroll of the lowest numbered plane which is "on" in that quadrant and for zoom.

```
TVWHER (QUAD, RPOS, IBUT, IERR)
Outputs:
   QUAD  I       Quadrant number
   RPOS  R(2)    TV position corrected for scroll (& zoom)
   IBUT  I       Value of button(s) pushed
   IERR  I       Error code of ZM70XF
```

10.6.2 TV I/O Utilities

BLTFIL

Fills in a series of closed polygons on a TV "blotch" plane

```
BLTFIL (NP, NV, XV, YV, GRCHAN, BUF, IERR)
Inputs:
   NP      I        # of polygons
   NV      I(NP)    # of vertices in each polygon
   XV      I(*)     vertex X-positions list
   YV      I(*)     vertex Y-positions list
   GRCHAN  I        graphics channel number
Output:
   BUF     I(*)     scratch buffer (> 1280)
   IERR    I        ZM70XF error code: 0 ok
```

GRPOLY

Uses a graphics plane to let the user develop a set of closed polygons as a "blotch" region. This routine is available only from program AIPS.

```
GRPOLY (IG, NPY, NV, XV, YV, SCRTCH, IERR)
Inputs:
   IG      I          graphics plane to use
Output:
   NPY     I          Number of polygons set
   NV      I(50)      Number of vertices in each polygon
   XV      I(500)     X-position of vertices in image
   YV      I(500)     Y-position of vertices in image
   SCRTCH  I(*)       Scratch buffer: > 1 line length (> 1280)
   IERR    I          Error code
Common:
   /MAPHDR/ CATBLK image catalog block of blotched image
                   (actually used at lower level in YCUCOR)
```

IENHNS

Performs an interactive linear enhancement of TV LUTs. X cursor => intercept, Y cursor => slope, high button => quit

```
IENHNS (ICHAN, ICOLOR, ITYPE, RPOS, BUFFER, IERR)
Inputs:
   ICHAN   I          channel select bit mask
   ICOLOR  I          color select bit mask
In/Out:
   ITYPE   I          on in: 1 => do plot, A, B switch plot
                              C switch sign of slope
                         2 => no plot, A, B return
                              C switch sign of slope
                         3 => no plot, return any button
                      on out - button value
   RPOS    R(2)       Cursor position: initial -> final
Output:
   BUFFER  I(>3072)   Scratch buffer
   IERR    I          Error code of ZM70XF: 0 => ok
```

ILNCLR

Computes a piecewise linear OFM and writes it to the TV. If NEND(NPOINT) is <= OFMINP/2 or /3 or /4, the table is repeated an approprite number of times.

```
ILNCLR (COLOR, NPOINT, NEND, SLOPE, OFFSET, GAMMA,
  *   BUFFER, IERR)
Inputs:
   COLOR   I            color bit mask: RGB = 421
   NPOINT  I            # of segments
   NEND    I            end points of segments
   SLOPE   R(NPOINT)    slopes of segments
   OFFSET  R(NPOINT)    offsets of segments
   GAMMA   R            power applied to colors (1 /gamma)
Output:
   BUFFER  I(1024)      scratch buffer
   IERR    I            error code of ZM70XF : 0 - ok
Form is C = (i-1)*SLOPE + OFFSET with 0 <= C <= 1.0.
Used to be called YLNCLR.
```

IMCCLR

Writes a color contour OFM using a standard table and sequence of colors.

```
IMCCLR (ITYPE, NLEVS, NSTART, NCONT, GAMMA, BUFFER,
  *   IERR)
Inputs:
   ITYPE   I          Which table: 2 Dutch 10c, 1 Dutch 9c,
                      3 IMPS 8c, 4 IMPS 64c
   NLEVS   I          number of levels (256 or 1024 usually)
   NSTART  I          intensity level of first contour
   NCONT   I          intensity range to contour
   GAMMA   R          gamma power for color correction
Output:
```

```
   BUFFER  I(1024)    scratch buffer
   IERR    I          error code of ZM70XF: 0 => ok
```

IMCHAR

Causes characters to appear on the TV by calling YCHRW.

```
IMCHAR (CHAN, X, Y, IANGL, CENTER, STRING, SCRTCH,
  *   IERR)
Inputs:
   CHAN    I       channel number (1 - NGRAY+NGRAPH)
   X       I       X position of string
   Y       I       Y position of string
   IANGL   I       0 - horizontal (to right), 3 - vertical (down)
                   ONLY ones supported.
   CENTER  I       0 - XY are lower left of first character
                   1 - XY are center of string
                   2 - XY are upper right of last character
   STRING  C*(*)   character string to go to TV - length from LEN
Output:
   SCRTCH  I(*)    scratch buffer (TV size > 1280)
   IERR    I       error code of ZM70XF: 0 - ok;   2 - input error
```

IMLCLR

Creates a continuous coloring from blue thru green to red.

```
IMLCLR (NLEVS, ICOLR, NBRK, GAMMA, BUFFER, IERR)
Inputs:
   NLEVS   I      # of intensities (usually 256 or 1024)
   ICOLR   I      initial color R,G,B = 1,2,3
   NBRK    I      break point between blue & red
   GAMMA   R      gamma correction power
Output:
   BUFFER  I(*)   scratch buffer
   IERR    I      error code of ZM70XF: 0 - ok
```

IMPCLR

Uses an STC algorithm to produce color contouring along a helix in the lightness-hue-saturation space.

```
IMPCLR (NLEVS, STEPS, LOOPS, LITE, SATUR, HUE, GAMMA,
  *   BUFFER, IERR)
Inputs:
   NLEVS   I          number of intensities (256 or 1024)
   STEPS   I          number of output colors (1 - 1024)
   LOOPS   I          number of loops of helix
   LITE    R(2)       min,max lightness (0.-100.)
   SATUR   R(2)       min,max saturation (0.-100.)
   HUE     R          start hue in degrees (0-360.)
   GAMMA   R          gamma correction power (2.7 or 1.8 ok?)
Output:
   BUFFER  I(1024)    scratch buffer
   IERR    I          error code of ZM70XF
```

TVFIDL

Does an interactive run with button A selecting alternately TVTRANSF and TVPSEUDO (color contour type 2 only), button B incrementing the zoom and C decrementing the zoom.

```
TVFIDL (LCHAN, NLEVS, INBUF, IERR)
Inputs:
   LCHAN    I      Selected gray-scale channels: bit mask
   NLEVS    I      Number of gray levels (usually LUTOUT+1)
Output:
   INBUF    I      Scratch buffer >3072
   IERR     I      Error code: 0 -> ok;  else set by ZM70XF
```

TVLOAD

Subroutine to load a map from an already opened map file to one TV memory plane. TVLOAD puts TV and map windows in the image header and writes it in the image catalog. It assumes that the other parts of the image header are already filled in (and uses them) and that the windows are all computed.

```
TVLOAD (LUN, IND, IPL, PXINC, IMAWIN, WIN, BUFSZ, BUFF, IERR)
Inputs:
   LUN      I      Logical unit # of map file
   IND      I      FTAB pointer for map file
   IPL      I      Channel to load
   PXINC    I(2)   Increment in x,y between included pixels
   IMAWIN   I(4)   TV corners: BLC x,y TRC x,y
   WIN      I(4)   Map window: ""
   BUFSZ    I      Buffer size in bytes
Outputs
   BUFF     R(*)   Buffer
   IERR     I      Error code: 0 => ok
                               1 => input errors
                               2 => MINIT errors
                               3 => MDISK errors
Commons: /DCAT.INC/ CATBLK  image header
```

10.6.3 Non-I/O Utilities

DECBIT

translates a decimal based channel number into a binary channel number, e.g., 1453 => $2^0 + 2^3 + 2^4 + 2^0$. A maximum of nine channels are addressable (6 at a time).

```
DECBIT (HMAX, ICHAN, IPL, LOW)
Inputs:
   HMAX       I       Maximum allowed channel number
   ICHAN      I       Input channel decimal number
Outputs:
   IPL        I       Binary channel # pattern
   LOW        I       Lowest of specified channels
```

ISCALE

Rescale a line of map data in preparation for output device

```
ISCALE (TYP, OMAX, RANGE, NPIX, NINC, RBUF, OUTBUF)
Inputs:
   TYP      C*2    Type of scaling function '  ' = linear
   OMAX     I      Maximum value accepted by output device
   RANGE    R(2)   Map range to transfer; RANGE(1) and (2)
                   should be transfered to output 0 and OMAX
                   RANGE(1) < RANGE(2) forced inside routine.
   NPIX     I      Pixels/line
   NINC     I      Increment between included pixels
   RBUF     R(*)   Input buffer
Outputs:
   OUTBUF   I(*)   output buffer with scaled data
```

MOVIST

Sets and resets the movie status parameters in the TV common.

```
MOVIST (OP, ICHAN, NFR, NFRPCH, MAG, IERR)
Inputs:
   OP       C*4   'ONNN' when turning on a movie
                  'OFFF' when clearing channel(s)
   ICHAN    I     Bit pattern of channels involved (OFFF)
                  Actual first channel number (1-NGRAY, ONNN)
   NFR      I     Number of frames in movie total (ONNN)
   NFRPCH   I     Number of frames per TV channel (ONNN)
                  < 0 => in display rather than movie order
   MAG      I     Magnification number (0 - 3, ONNN)
Output:
   IERR     I     Error = 2 => bad input, else ok
The code: bit 1 (lsb) 1 (on) = display order, 0 => movie order
              2       1 => frame starts movie
              3-6     magnification step
              7-15    # frames
```

RNGSET

Calculates range parameters for displaying a map using the IRANGE adverb supplied by POPS plus scaling information derived from the map header.

```
RNGSET (IR, MMAX, MMIN, RANG)
Inputs:
   IR     R(2)   Range values specified by user
   MMAX   R      Map maximum value from header
   MMIN   R      Map minimum value from header
Outputs:
   RANG   R(2)   Output range values calculated using defaults
```

TVWIND

Sets windows for normal and split screen TV loads.

```
TVWIND (TYPE, PXINC, BLC, TRC, ICHAN, ITVC, IWIN,
  *   IERR)
In/out:
```

```
   TYPE   I       In: <0 -> 1 plane, other -> split method
                  Out: 0 -> 1 plane, other = 10 * (#planes in X) +
                                             (# planes in Y)
   PXINC  I(2)    X, Y increments
   BLC    R(7)    User requested bot left corner
   TRC    R(7)    User requested top right corner
   ICHAN  I       User requested TV chan (decimal form)
   ITVC   I(4)    IN: first 2 user req. TVCORN
                  Out: full "pseudo-TV" corners
Output:
   IWIN   I(4)    Window into map
   IERR   I       error code: 0 -> ok, else fatal
Common:
   /DCAT.INC/ CATBLK image header used extensively, the depth array is
                                 set here
```

Chapter 11

Plotting

11.1 Overview

Plotting in AIPS is usually a two-step process. First a task or a verb creates an AIPS "plot file", which consists of plot-device independent "commands" that tell a device how to draw the plot. This file is always an extension file associated with a cataloged file. The second step in obtaining a plot is to run a task to read the plot file and write it to a specific device, such as a TV, or a hardcopy plotter. This two-step method greatly reduces the number of plot programs that must be written and maintained. For instance, if a new graphics device is added to the system, then only one new program, that reads the plot file and writes to the new device, is needed. All the other plotting programs work with no modification. Another advantage is that a plot file may exist for an extended period of time, thus allowing plots to be written to different devices, and copies to be generated at later times, without duplicating the calculations needed in making the plot.

There are exceptions to the two-step process. For example, slices of map files can be plotted directly on the Tektronix 4012. This is done to simplify matters in interactive situations such as gaussian fitting of slices.

AIPS contains some very powerful routines for plotting in a variety of coordinate systems in use in astronomy. The complexity of these routines is commensurate to their power. Fortunately, a set of plot program templates exist to provide a starting point. These routines are described in a later section in this chapter.

11.2 PLOT FILES

11.2.1 General Comments

Plot files are a generalized representation of a graphics display. They contain scaling information and commands for drawing lines, pixels, and characters, and a command for putting miscellaneous information in the image catalog. The image catalog is used by programs that must know details about an image currently displayed on the graphics device in order to allow user interaction with the device. For example, a program may want to read a cursor position and translate it to the coordinate system of the image displayed on the graphics device.

The records in plot files do not include a record length value. This means that it is inconvenient to invent new types of records (i.e., new opcodes) or to add new values on to the end of records of existing types because all of the programs must be changed. On the other hand, the rigid format definitions aided in debugging the code several years ago and continue to assure the integrity of I/O systems (AIPS device plotting programs refuse to proceed if they encounter an unknown opcode). So far, the increased flexibility supplied by length values seems not to have been absolutely required in AIPS.

The character drawing record includes neither a size value nor an angle value. This is because character plotting capabilities are device dependent. Orientations are either vertical or horizontal (and not backwards) and the position offsets for plotting character strings are specified in units of the device character size,

permitting the device plotting program to position strings nicely no matter what size it chooses to use. It also follows that most plots produced by AIPS have only one size of character. One AIPS application program (PROFL) draws its own characters by using the line drawing commands in order to plot characters with arbitrary size, orientation, and even perspective.

11.2.2 Structure of a Plot File

The first physical record (256 words) in the plot file contains information about the task which created the file. It is not logically part of the "plot file", but is there to provide documentation of the file's origins. This record is ignored by the programs that actually do the plotting. The primary use of this information is by the the verb EXTLIST that lists all the plot files associated with a cataloged file. When new types of plots are added to AIPS, an experienced programmer should update the verb EXTLIST (found in subroutine AU8A) to list useful things about the plot. Otherwise the verb will print a line telling the user that he has a plot file of type UNKNOWN. A novice AIPS programmer should leave this code alone.

The contents of the first physical record are task-dependent and have the form:

```
FIELD   TYPE        DESCRIPTION
  1.    H(3)       Task name (6 character hollerith string)
  2.    I(6)       Date/time of file creation YYYY,MM,DD,HH,MM,SS
  3.    I          Number of (integer) words of task parameter data
  4.    R(*)       Task parameter block as transmitted from AIPS
                   (preferably with defaults replaced by the values
                    used).
```

The rest of the plot file contains a generalized representation of a graphics display. This representation is in the form of scaling information and commands for drawing lines, pixels, and characters and a command for putting miscellaneous information in the image catalog.

The lowest level plot file I/O routines read and write 256-word blocks. The applications programmer will be concerned with routines that read and write logical records. The logical records are of 6 types and vary in length. With the exception of the "draw pixels" record, logical records do not cross the block boundaries. Unused space at the end of a block consists of integer zeros. All values in the plot file are I variables or Hollerith characters. This aids in exporting plot files to other computers via tape. Unfortunately, this also limits the values that can be stored in the plot file, thus forcing us to use a scaling factor and offset for some plots to prevent integer overflow. The scaling factor and offset are not in the plot file. If we are not careful, this can cause some problems for interactive tasks that read positions from a graphics device and then try to convert them to the original coordinates. These interactive tasks must make do with information from the map header and data from the "miscellaneous information" record.

Plot files have names of the format PLdsssvv, where d is the format revision letter, sss is the catalog slot number of the associated map, and vv is the version number. Hexadecimal code is used for sss and vv, limiting the number of versions to 255.

11.2.3 Types of Plot File Logical Records

Initialize plot record

The first logical record in a plot file must be of this type. It is written by the basic subroutine GINIT; see the end of this chapter for the details of the calling sequence.

```
FIELD  TYPE     DESCRIPTION
  1.   I       Opcode (equal to 1 for this record type).
  2.   I       User number.
  3.   I(3)    Date:  yyyy, mm, dd
  4.   I        ITYPE: type code 1 => misc.,  2 => CNTR,   3 => GREYS,
                 4 => PROFL,  5 => SL2PL,  6 => PCNTR,  7 => IMEAN,
```

```
 8 => UVPLT,  9 => GNPLT, 10 => VBPLT, 11 => PFPLn,
12 => GAPLT, 13 => PLCUB, 14 => IMVIM, 15 => TAPLT,
16 => POSSM, 17 => SNPLT, 18 => KNTR,  19 => UVHGM,
20 => ISPEC
```

Initialize for line drawing record

This record provides scaling information needed for a plot. The plot consists of a "plot window" in which all lines are drawn and a border (defined in terms of character size) in which labeling may be written. The second record in a plot file must be of this type. This record is written by the subroutine GINITL.

FIELD	TYPE	DESCRIPTION
1.	I	Opcode (equal to 2 for this record type).
2.	I	X Y ratio * 100. The Ratio between units on the X axis and units on the Y axis (X / Y). For example if the decrement between pixels in the X direction on a map is twice the decrement in the Y direction the X Y ratio can be set to 2 to provide proper scaling. Some programs may ignore this field. For example TVPL when writing grey scale plots to the TV.
3.	I	Scale factor (currently 16383 in most applications). This number is used in scaling graph positions before they are written to disk. BLC values in field 4 are represented on disk by zero and TRC values are represented by integers equal to the scale factor).
4.	I(4)	The bottom left hand corner X and Y values and the top right hand X and Y values respectively in the plot window (in pixels).
5.	I(4)	1000 * the fractional part of a pixel allowed to occur outside the (integer) range of BLC and TRC (field 4 above) in line drawing commands
6.	I(4)	10 * the number of character positions outside the plot window on the left, bottom, right, and top respectively
7.	I(5)	Location of the X Y plane on axes 3,4,5,6,7. This field is valid only for plots associated with a map.

Initialize for grey scale record

This record, if needed, must follow the "init for line drawing" record. This record allows proper interpretation of pixels for raster type display devices. Programs that write to line drawing type devices (e.g., the Tektronix 4012) ignore this record. Subroutine GINITG is used to write this record into the plot file.

FIELD	TYPE	DESCRIPTION
1.	I	Opcode (equals 3 for this record type).
2.	I	Lowest allowed pixel intensity.
3.	I	Highest allowed pixel intensity.
4.	I	Number of pixels on the X axis.
5.	I	Number of pixels on the Y axis.

Position record

This record, written by subroutine GPOS, tells a device where to start drawing a line, row/column of pixels, or character string.

```
FIELD  TYPE     DESCRIPTION
  1.    I       Opcode (equals 4 for this record type).
  2.    I       scaled x position, i.e., a value of 0 represents the
                BLC values defined in the "init for line drawing"
                record, and a value equal to the scale factor
                represents the TRC value.
  3.    I       Scaled Y position.
```

Draw vector record

Subroutine GVEC writes this record to tell a device to draw a line from the current position to the final position specified by this record.

```
FIELD  TYPE     DESCRIPTION
  1.    I       Opcode (equals 5 for this record type).
  2.    I       Scaled final X position.
  3.    I       Scaled final Y position.
```

Write character string record

This record tells a device to write a character string starting at the current position. Subroutine GCHAR writes this logical record.

```
FIELD  TYPE     DESCRIPTION
  1.    I       Opcode (equals 6 for this record type).
  2.    I       Number of characters.
  3.    I       Angle code: 0 = write characters horizontally.
                            1 = write characters vertically.
  4.    I       X offset from current position in characters * 100
  5.    I       Y offset from current position in characters * 100
                (net position refers to lower left corner of 1st char)
  6.    I(n)    Hollerith characters (n = INT((field2 + 3) / 4))
```

Write pixels record

This record tells a raster type device to write an n-tuple of pixel values starting at the current position. Programs that write to line drawing type devices ignore records of this type. This type of record may span disk-record boundaries and is written by subroutine GRAYPX.

```
FIELD  TYPE     DESCRIPTION
  1.    I       Opcode (equals 7 for this record type).
  2.    I       Number of pixel values.
  3.    I       Angle code: 0 = write pixels horizontally.
                            1 = write pixels vertically (up).
  4.    I       X offset in characters * 100.
  5.    I       Y offset in characters * 100.
  6.    I(n)    n (equal to field 2) pixel values.
```

Write misc. info to image catalog record

This record tells the programs that write to interactive devices (TKPL, TVPL) to put up to 20 words of miscellaneous information in the image catalog starting at word IITRA + 2. This information is interpreted by routines such as AU9A (TKPOS, TKVAL, etc.) and is used, at the moment, only for slices by task SL2PL. Routines that write to non-interactive graphics devices (PRTPL) ignore this record. Subroutine GMCAT handles this record.

```
FIELD  TYPE      DESCRIPTION
  1.   I         Opcode (equals 8 for this record type).
  2.   I         Number of words of information.
  3.   I(n)      Miscellaneous info (n=value of field 2).
```

End of plot record

This record marks the end of a plot file and is written by subroutine GFINIS.

```
FIELD  TYPE     DESCRIPTION
  1.   I      Opcode (equals 32767 for this record type).
```

11.2.4 Other Plotting Customs

Over the years, certain plotting options have come to be expected by AIPS users. Among these are (1) recording the plot in the main file's history extension file, (2) offering to plot "star" positions on appropriate plots (i.e., those with the image's X-Y coordinates for axes), (3) offering to draw tick marks all the way across the plot, and (4) offering a variety of axis labeling options. The subroutines mentioned in the discussion below are shown with their precursor remarks at the end of this chapter.

1. HIPLOT adds a line to the main file's history extension file giving the time, date, task name, and plot file version number.

2. STARPL plots "star" positions on plots normally, but not necessarily, with axes which are a celestial-coordinate pair. The positions are entered as an extension file by the user with task STARS. The user then specifies which star extension file with INVERS and whether the positions are plotted and, if so, how large to plot them with adverb STFACTOR.

3. LABINI is an initialize routine which sets up the basic position common and initializes a variety of other standard plotting parameters. Calls CHNTIC. The user specifies the axis labeling type with adverb LTYPE (see the precursor remarks for supported values). See the chapter on Catalogs for details of the position common.

4. COMLAB is a plot labeling routine designed for contour maps. It is a good example of the standard conventions for labeling.

5. CLAB1 labels the X and Y axes and calls CTICS.

6. CLAB2 labels the X and Y axes and calls CTICS. It allows the plotting of the X axis at a specified Y value and vice versa.

7. CTICS draws tick marks on the plot, either as short ticks or as tracks following the coordinate across the full plot area. The user specifies this option with the AIPS adverb DOCIRCLE.

8. CHNTIC counts characters used for labeling Y-axis tick marks. This number is needed for the call to GINITL.

9. CONDRW draws a contour display of an image.

11.3 Plot Paraform Tasks

11.3.1 Introduction

Three paraform tasks (PFPL1, PFPL2 and PFPL3 - sometimes called PFPLn below) are available in AIPS for developing plot tasks that read a map and create a plot file to be associated with the map. These tasks use the standard AIPS defaults for adverb values such as INNAME, BLC, TRC, XYRATIO, PIXRANGE, etc. The programs are heavily commented and modular.

The three tasks correspond to the three types of plots that can be found in AIPS. The first type is a plot of an X Y plane of the map or a subimage of the map. In this case, the X and Y axes of the plot are the same as the X and Y axes of the map. Examples of this type are produced by tasks CNTR and GREYS. A second type of plot is when the X axis of the plot is a slice of the X and Y axes of the map and the Y axis of the plot is some other value such as intensity. Task SL2PL will create a plot of this type from a slice of a map. The third type of plot is when the axes of the plot have no real relation to the map axes. An example of this type of plot is the histogram produced by task IMEAN.

The structures of all three paraform tasks are very similar. The major differences are in subroutine PLINIn (the subroutine that initializes the commons used in labeling the plot), PLABLn (this routine does the actual labeling), and in the example plots in subroutine PLTORn. The adverbs received from AIPS also differ slightly. The tasks will be discussed individually in a following section, but first we will describe the general structure of all three programs. The tasks perform the following steps:

1. Open an image file corresponding to the users inputs from AIPS.
2. Create an extension file of type PL (plot) to be associated with the image file. The header of the image file will be updated to include this new extension file.
3. Write the plot file records to draw the borders and labels of the plot. The programmer can customize this section of the program by changing data statements and assignment statements in the main program.
4. Write the rest of the plot file records to the plot file. This is done by subroutine PLTORn. The programmer will have to modify the code in PLTORn for his needs.
5. Do the necessary clean up functions, write end of plot records, close all files, etc.

11.3.2 Getting Started

The first step is choosing a new name and making copies, using the new name, of the source code file and the help file. One should copy files APGNOT:PFPLn.FOR, and HLPFIL:PFPLn.HLP ("n" stands for 1, 2 or 3) to a user directory and work with the program there.

When a task is renamed, some source code must be changed. The first line of the program after the LOCAL INCLUDE

```
PROGRAM PFPLn
```

and the data statement

```
DATA PRGNAM /'PFPLn '/
```

should be changed to use the new name. The name in the HELP file should also be changed. See Appendix A in volume 1 for details of compiling, linking and debugging the task.

11.3.3 Labeling the Plot

The labeling of the plot takes place in two subroutines called by subroutine PLTORn. PLINIn will set a number of variables in common that give the labeling routines and the plot drawing routines information about the corners of the plot, the types of the axes, the type of labeling, the size of the plot borders in characters, and other details.

Subroutine PLABLn uses the information provided by PLINIn to actually write the commands in the plot file to draw the labels, borders, and tic marks.

The programmer can customize the labeling somewhat without changing either PLINIn or PLABLn by setting values in an array PCODE, and changing data statements in the main program.

Optional text can be printed at the bottom of a plot by setting values INTEXT (number of lines of text), and ATEXT (an array containing the actual text lines). These values are currently set in DATA statements in the main program. The programmer can choose to set INTEXT to zero to suppress all of the lines. If the programmer wishes to use more than two lines, then the second dimension of arrays TEXT and ATEXT must be changed in all the routines in which they are declared.

See the section on the individual programs for details on setting PCODES.

11.3.4 Plotting

Plotting consists of reading the map, collecting the data, and then drawing lines or writing grey scale pixels. All of these steps are usually done in subroutine PLTORn. Reading a map is usually done with routine GETROW (see below). Setting a starting point of a line is usually done with routine PLPOS. Setting the end point of a line is done with PLVEC. Grey scale pixels are written with subroutine PLGRY.

11.3.5 Map I/O

This program uses the standard AIPS I/O package grouped into a few subroutines. This approach attempts to make life a little easier by hiding a few of the messy details, but not to eliminate the flexibility of the I/O by hiding it under a complex system. These routines use the "copy mode" approach to I/O in that data is read into a large buffer and then copied from the large I/O buffer to a smaller buffer when a row is needed. This is less efficient than using the bare AIPS I/O routines, but frees the programmer from having to deal with indexes into the large array.

There are four I/O routines in this program, MAKNAM (fills in an array with all the data items that go into specifying a map), INTMIO (initializes the I/O routines to read or write a cataloged map), REIMIO (initializes counters for reading a different subimage or making another pass through a map opened by INTMIO) and GETROW (reads a row of a map). MAKNAM and INTMIO are used in straight forward ways to open the map. The programmer can usually ignore these two routines unless a second map must be opened. If the program must make more than one pass through the data, REIMIO can be used to reset all of the counters. REIMIO assumes that the map is already opened in INTMIO and that a second pass is being made through the data. This routine can NOT be used to read different subimages from the same map at the same time. GETROW must be used (usually in subroutine PLTORn) to read data from the map, one row at a time.

The I/O routines in this program use a common named MAPHDR (include DCAT.INC). This common was chosen to interface with several of the plotting routines which expect this common to contain the map header. Besides the map header, this common contains an array, IMSTUF, which has several data items of interest. IMSTUF(9) is of particular interest since it contains the number of data values (pixels) in each row of the map. This number is usually the upper limit of a loop which operates on each element in the map row. A description of all the elements of IMSTUF are listed in the following table:

```
1.      AIPS I/O Logical unit number
2.      FTAB index
3.      unused
4.      unused
5.      Catalog slot of image.
6.      Size of I/O buffer in bytes.
7.      Disk volume number of image.
8.      Number of dimensions in image.
9.      Number of values read per row of image.
10-16.  Number of values along all 7 axes
17-30.  Window in BLC TRC pairs along all 7 axes.
31-36.  Current position on last six axes.
```

```
37      1 if read forward -1 if backward read on 2nd axis.
```

Minor modifications in the I/O routines could be made to produce routines for reading UV data, but this has not yet been done.

11.3.6 Cleaning Up

Some of the adverbs passed from AIPS may not be used for some types of plots. The programmer can make things easier for the AIPS user by removing them from the help file. The programmer must then remove them from the local include which can be found at the beginning of the file. The variable NPARMS is initialized in an assignment statement in the main program. This must be changed to correspond to the new number of words received from AIPS.

11.3.7 The Three Paraform Plot Tasks

PFPL1

This task should be used when developing a plotting task in which the X and Y axes of the plot are the same as the X and Y axes of the map.

Much of the labeling is controlled by values of array PCODE. The values for the elements of PCODE are summarized in the following table and are set in the main program between the calls to PFnINI and PLTORn.

```
If PCODES(1) equals

     1   then the plot axes consist of an unlabeled
         rectangular border.

     2   then draw a rectangular border plus
         the title and text at the bottom.

     3   then draw a rectangular border, labels,
         and border tick marks
         indicating absolute coordinates (r.a., decl., etc.).

     4   then draw a rectangular border, labels, and border tick marks
         indicating coordinates relative to the coordinates
         of the image reference pixel (units usually in
         arc seconds).

     5   draw border, labels, and border tick marks
         indicating coordinates relative to the center of
         the subimage plotted (units usually in arc seconds).

     6   draw border, labels, and border tick marks
         indicating image pixel numbers.

If PCODES(2) equals

   0       then label the X axis with the X axis value found in the
           map header.

   other   then label the X axis using variable AXUNIT which is set in
           a DATA statement in the main program.
```

```
If PCODES(3) equals

   0       then label the Y axis with the Y axis value found in the
           map header.

   other   then label the Y axis using variable AYUNIT which is set in
           a DATA statement in the main program.

If PCODES(4) equals

   0       then use the "standard" title consisting of map name,
           source name, and frequency.

   other   then use the title given in DATA statement for
           variable ATITLE in the main program.

If PCODES(5) equals

   0       then no grey scale pixels are to be written for the
           plot.

   other   then grey scale pixels with a range given by PIXRNG
           (these values are usually passed from AIPS in adverb
           PIXRANGE) can be written to the plot. This code value
           causes an "init for grey scale" record to be written
           to the plot file.
```

Usually a task will let the AIPS user choose the value of PCODES(1) by setting adverb LTYPE, e.g., PCODES(1) is set to LTYPE after the task gets this adverb value from AIPS.

When using PLPOS and PLVEC, the positions for this type of plot are given in pixels.

The unmodified version of PFPL1 contains code in PLTOR1 to read the map, and draw a grey scale plot. The user should remove this example found between comment lines "** Plot specific code" and "** End plot specific code" and insert the code for his own application.

PFPL2

This task should be used when developing a plotting task in which the X axis of the plot is a slice of some plane of the map, and the Y axis is some other value such as intensity. The PCODE usage is described below.

```
PCODES(1) equals

           The label type of the X axis. The codes are the same
           as for PFPL1.

If PCODES(2) equals

   0       then label the X axis with the units determined by the
           "standard" slice labeling algorithm.

   other   then label the X axis using variable AXUNIT which is set in
           a DATA statement in the main program.
```

```
If PCODES(3) equals

   0       then label the Y axis with the units found in the
           map header for the map intensity.

   other   then label the Y axis using variable AYUNIT which is set in
           a DATA statement in the main program.

If PCODES(4) equals

   0       then use the "standard" title consisting of map name,
           source name, and frequency.

   other   then use the title given in DATA statement for
           variable ATITLE in the main program.

If PCODES(5) equals

   0       then use the "standard" slice message at the bottom of
           the plot. This message will give the center of the slice.
           This message occurs above the message found in TEXT
           as described above.

   other   then do not print the "standard slice message"
```

The example program in PFPL2 will plot a slice of the X Y plane. The user should remove the example found between comment lines "** Plot specific code" and "** End plot specific code" and insert the code for his own application. This example uses no interpolation (it uses the value of the nearest pixel) and is NOT adequate for a production program. See the code in task SLICE for a good set of interpolation routines and a "rolling buffer" scheme.

PFPL3

This task should be used when developing a plotting task in which the X and Y axes have no relation to the map X and Y axes. The plot could be of a function, a histogram of some values, or a table.

The only PCODES values used are PCODES(4) and PCODES(5). If PCODES(4) is 0, then the program plots the "standard" title line. Otherwise, it uses whatever string is in variable ATITLE. If PCODES(5) is not zero, then this signals the existence of grey scale pixels. The program automatically uses whatever strings are in variables AXUNIT and AYUNIT to label the units for X and Y. Thus, the programmer will have to edit the DATA statements for these variables in the main program, or fill them in by some other means.

The example program in the unmodified version of PFPL3 will plot a simple histogram of map intensities. The subroutine PLTOR3 reads the map to determine the histogram values and the range of the Y axis (number of pixels). Then the standard initializing routine (PLINIn) and labeling routine (PLABLn) are called. Finally the histogram is plotted. The programmer must remove the two sections of example code found between two sets of comment lines "** Plot specific code" and "** End plot specific code" and insert the code for his own application.

11.4 Plotting to Devices

There are a wide variety of devices which may be used as plotters by AIPS code. The plot files may be interpreted to each of these devices by a device-dependent task. Currently, AIPS supports TV devices with

TVPL (IIS models 70, 75 and IVAS, DeAnza, Comtal Vision 1/20), Tektronix 4010 and 4012 and emulator devices with TKPL, Versatec and similar electrostatic printer/plotters with PRTPL, and QMS Lasergraphix printers with QMSPL. Many other devices could, in principle, be supported, but it is hard for the AIPS group at the NRAO in Charlottesville to create tasks, and to provide reliable support, for devices not physically present at the NRAO in Charlottesville. For this, we depend primarily on our user community. We will be glad to receive any plotter tasks which you may wish to submit and will do what we can to provide support. We are, of course, quite adept at shipping code to the world-wide AIPS user community.

11.4.1 Versatec

The Versatec is an electrostatic printer/plotter with 200 dots to the linear inch. It works by drawing a row of dots, advancing a 200th of an inch, and drawing the next row of dots. In other words, it is not a randomly addressable device like a TV or the QMS. Therefore, PRTPL must prepare a file containing the entire bit matrix to be plotted. Then the subroutine ZDOPRT reads the file a row at a time, prepends special device plot codes, and sends them to the device (or the device spooler). Unfortunately, these special codes are specific to the Versatec device driver in use and hence are different for other devices (e.g., Printronix) and can even be different for the same device with a different driver. The standard version of ZDOPRT provided with VMS AIPS is intended to work for spooled Versatecs under VMS with drivers of revision C or greater. The precursor comments for this routine are reproduced at the end of this chapter.

To adapt PRTPL to other, similar devices, you will need to do at least two things. The simpler is to correct the setting of YPRDMM at the beginning of subroutine PRTDRW. This sets the number of dots per millimeter in the Y direction and is, in general, not the same as the X dots per millimeter (as is assumed by PRTPL). You should also set the plotter size and X dots per millimeter using the stand-alone program SETPAR, since these are "global" AIPS parameters carried in the device characteristics common (include DDCH.INC. Second, you must produce a version of ZDOPRT suitable to your device. To assist in this, we provide, in the directory APLVMS:, four other versions of ZDOPRT, called ZDOPRn, which may be of assistance.

11.4.2 QMS Laser Printer

The QMS Lasergraphix is a printer/plotter based on a laser print device manufactured by Canon. (Numerous other OEMs are also selling devices based on this engine.) The resolution is 300 dots to the linear inch and there is a full-page, randomly addressable memory. The device is capable of accepting downloaded "fonts", vector commands, and other useful plot commands. AIPS task QMSPL translates AIPS plot files into ASCII print files containing the commands to the QMS. The Z routine ZQMSIO performs the necessary open, write, and close operations. It opens the file in a way which will cause it to be deleted if the program aborts. This is to prevent partial files from clogging the disk and/or being printed and leaving the QMS in a strange state. The close operation redirects the file to "print/delete", or, at the user's request, to "print/keep" in a file named by the user. The precursor comments for ZQMSIO are given at the end of this chapter.

To convert ZQMSIO for use on some other comparable device, two major operations must be performed. First, a ZxxxIO must be written to perform the operations of ZQMSIO. They are likely to be very similar. Second, the command grammar and text of the QMS must be translated to the new device in the QMSPL task (under a new xxxPL name, of course). Where possible, we recommend doing this second operation, rather than creating a wholly new xxxPL. QMSPL contains an excellent algorithm, based on one used by Starlink in Great Britain, for displaying grey scale pixels on dot matrix devices. There are also a number of other scaling algorithms for grey scale pixels in QMSPL, which have proven useful. It would be best to avoid reinventing these wheels. QMSPL represents grey scale pixels in two modes, quick and slow. If there is a significant number of dots per image pixel, then QMSPL can use the quick mode of precomputing a "font" for each of some number of grey levels and then translate the pixels into letters to be printed. A better, but much more expensive, display is produced by computing the dot pattern for each image pixel based on its exact value and an exact implementation of the Starlink algorithm.

In order to understand QMSPL in order to do the translation, it is necessary to have some feel for the QUIC language of QMS. The commands used by QMSPL are summarized below:

```
^PY^-          Go into QUIC command mode
```

```
^IOP              Use portrait mode
^IOL              Use landscape mode
^F^-              Use free format (ignore system-provided
                  carriage returns, line feeds, etc.
^DCnnnn           Make nnnn copies of current page
^ISYNTAX00010     Change measurements to dots, other options
                  remain at defaults
^ISTFX0           Text processing: enable automatic font
                  baselining, don't change auto-justify
^ITnnnn           Set x origin (left page margin) to nnnn dots
^IJnnnn           Set y origin (top page margin) to nnnn dots
^SM00204          Select font 204 in standard vertical,
                  used for labeling
^SM09001          Select "font" 9001, used for grey scales
^DF09001L0AIPSnnn  Define "font" 9001 in landscape (P for
                  portrait) called AIPS version 0 with nnn
                  dots in Y per character
,aannn^Myyyxxx000000 Define character aa (in hex), width nnn
                  dots, using bit map of actual size xxx
                  by yyy dots, with no border dots ---
                  followed by the definition given in 4-bit
                  hex characters padded to integer words
                  on each row.
^ILnnnn           Set character line spacing to nn.nn characters
                  per inch; nnnn = 0 => 6 lines per inch
^ICnnnn           Set character horizontal spacing to nn.nn
                  characters per inch; nnnn = 0 means use
                  proportional, not fixed spacing
^IGV              Turn on vector graphics mode
^IGE              Turn off vector graphics mode
^PWnn             Set pen width to nn dots (nn odd, <= 31)
^Uxxxx:yyyy       Move position to xxxx, yyyy in dots; yyyy is
                  measured down the page
^Dxxxx:yyyy       Draw vector to xxxx, yyyy in dots
^Pnnnn            Start binary transmission, nnnn dots per row;
                  followed by data given in 4-bit hex characters
                  padded to integer multiple of 4 characters
^G                Terminate current command, i.e., stop font
                  definition, binary transmission, etc.
^DF09001L^G       Delete "font" 9001 in landscape mode
^DF09001P^G       Delete "font" 9001 in portrait mode
^ISYNTAX00000     Restore default syntax, measurements in inches
^,                Print this page
^O                Off free format
^PN^-             Exit QUIC mode
```

11.5 Includes

11.5.1 DLOC.INC

used by all position routines; set by SETLOC.

```
C                                                Include DLOC.
```

```
C                                       Position labeling common
      DOUBLE PRECISION RPVAL(4), COND2R, AXDENU, GEOMD1, GEOMD2, GEOMD3,
     *   GEOMD4
      CHARACTER CTYP(4)*20, CPREF(2)*5, SAXLAB(2)*20
      REAL      RPLOC(4), AXINC(4), ROT
      INTEGER   ZDEPTH(5), ZAXIS, AXTYP, CORTYP, LABTYP, SGNROT,
     *   AXFUNC(7), KLOCL, KLOCM, KLOCF, KLOCS, KLOCA, KLOCB,
     *   NCHLAB(2)
      COMMON /LOCATC/ CTYP, CPREF, SAXLAB
      COMMON /LOCATI/ RPVAL, COND2R, AXDENU, GEOMD1, GEOMD2, GEOMD3,
     *   GEOMD4, RPLOC, AXINC, ROT, ZDEPTH,
     *   ZAXIS, AXTYP, CORTYP, LABTYP, SGNROT, AXFUNC, KLOCL, KLOCM,
     *   KLOCF, KLOCS, KLOCA, KLOCB, NCHLAB
C                                                       End DLOC.
```

11.5.2 DGPH.INC

used by the basic plot subroutines GINIT, GINITL, GPOS, etc.

```
C                                                       Include DGPH.
C                                       Plotting common
      INTEGER   GPHSIZ, GPHLUN, GPHIND, GPHPOS, GPHRRN, GPHVOL,
     *   GPHTLO, GPHTHI
      REAL   GPHX1, GPHX2, GPHY1, GPHY2, SCALEF
      CHARACTER GPHNAM*48
      COMMON /GPHCOM/ GPHSIZ, GPHLUN, GPHIND, GPHPOS, GPHRRN, GPHVOL,
     *   GPHX1, GPHX2, GPHY1, GPHY2, SCALEF, GPHTLO, GPHTHI
      COMMON /GPHCHR/ GPHNAM
C                                                       End DGPH.
```

11.5.3 DPLT.INC

used by PFPL1, PFPL2, PFPL3.

```
C                                                       Include DPLT.
C                                       Include for plotting
      DOUBLE PRECISION CATD(128)
      REAL      PBLC(2), PTRC(2), XY, XSCAL, XOFF, YSCAL, YOFF, GFAC,
     *   GOFF, RANGE(2), XLAST, YLAST, CATR(256)
      INTEGER   PLTBLK(256), IOFFPL, IVER, IBLKSZ, IMSTUF(37),
     *   IOBLK(8192), CATBLK(256)
      HOLLERITH CATH(256)
      COMMON /PLTCOM/ IOBLK, PBLC, PTRC, XY, XSCAL, XOFF, YSCAL,
     *   YOFF, GOFF, GFAC, RANGE, XLAST, YLAST, PLTBLK, IOFFPL, IVER,
     *   IBLKSZ, IMSTUF
      COMMON /MAPHDR/ CATBLK
      EQUIVALENCE (CATBLK, CATH, CATR, CATD)
C                                                       End DPLT.
```

11.6 Routines

11.6.1 CHNTIC

Counts the number of characters to the left of a plot used for labeling the vertical axis.

```
CHNTIC (BLC, TRC, CHMAX)
Inputs:
   BLC      R(2)   X AND Y pixels to form bottom left hand
                   corner of the graph.
   TRC      R(2)   X and Y pixels to form the top right hand
                   corner of the graph.
Output:
   CHMAX    I      Max number of characters.
```

11.6.2 CLAB1

Controls some axis drawing and labeling functions: labels each axis with RA/DEC or the type. call CTICS to draw tics and tick labels

```
CLAB1 (BLC, TRC, CH, ILTYPE, XYR, DOGRID, IBUFF, IERR)
Inputs:
   BLC      R(2)    X, Y pixels to form bottom left hand corner
   TRC      R(2)    X, Y pixels to form the top right hand corner
   CH       R(4)    left, bot, right, top : total character offsets
   ILTYPE   I       label type: 1 none, 2 no ticks, 3 RA/DEC
                       4 center relative
   XYR      R       The ratio of the distance between X axis pixels
                    and the distance between Y axis pixels on plot
   DOGRID   L       T => full coord grid, else ticks
In/out:
   IBUFF    I(256)  the updated graphics output buffer.
   IERR     I       error indicator: 0 = No error.
```

11.6.3 CLAB2

Controls some axis drawing and labeling functions: labels each axis with RA/DEC or the type. Call CTICS to draw tics and tick labels. Differs from CLAB1 in that the axes are drawn as subplots (fewer ticks and only one axis line in each direction, arguments AYV and AXV are used).

```
CLAB2 (BLC, TRC, CH, ILTYPE, XYR, AYV, AXV, IBUF, IERR)
Inputs:
   BLC      R(2)    X, Y pixels to form bottom left hand corner
   TRC      R(2)    X, Y pixels to form the top right hand corner
   CH       R(4)    left, bot, right, top : total character offset
   ILTYPE   I       label type: 1 none, 2 no ticks, 3 RA/DEC
                       4 center relative
   XYR      R       Ratio of the distance between X axis pixels and
                    the distance between Y axis pixels on the plot
   AYV      D       Draw the x axis at this Y value
   AXV      D       Draw the y axis at this x value
In/out:
   IBUF     I(256)  the updated graphics output buffer.
   IERR     I       error indicator: 0 = No error.
```

11.6.4 COMLAB

Is an axis drawing and labelling routine for use with the common labeling for contour plots and pol vector plots. It calls GINITL and puts subsidiary labels (source, frequency, Stokes, image name), (peak flux), (contour levels) in file.

```
COMLAB (BLC, TRC, LTYPE, IVER, YGAP, CH, XMULT, XLEVS, XYR, IBUFF,
 *   IERR)
Inputs:
   BLC      R(7)    bottom left hand corner of the map.
   TRC      R(7)    top right hand corner of the map.
   LTYPE    I       label type: 1 none, 2 no ticks, 3 Ra/dec
                    4 center relative, 5 subimg center-rel,
                    6 pixels, 7 as 3 no top labels
                    < 0 => no date/time, else as positive
   IVER     I       plot file version number
   XMULT    R       the multiplier for the LEVS to find the
                    actual contour levels.
   XLEVS    R(30)   the contour levels (when used with XMULT)
In/out:
   IBUFF    I(256)  the updated graphics output buffer.
   YGAP     R       On input: # lines at bottom to skip before
                    peak flux line in addition to standard
                    On output: includes standard
Output:
   IERR     I       error indicator: 0 = No error.
```

11.6.5 CONDRW

Will write commands to a plot file for the execution of a contour plot.

```
CONDRW (IMLUN, IMFIND, IGLUN, IGFIND, XMULT, BLC, TRC, LEVS, IGBLK,
  *   IERR)
Inputs:
   IMLUN    I       logical unit number for the map file.
   IMFIND   I       FTAB index for open map file.
   IGLUN    I       logical unit number of the graph file.
   IGFIND   I       FTAB index for initialized graph file.
   IGBLK    I(256)  I/O block for graph file.
   XMULT    R       Contour interval (image units)
   BLC      R(7)    Bottom left corner
   TRC      R(7)    Top right corner
   LEVS     R(30)   Selected contour intervals in increasing order
                    (any decrease terminates the list)
Common:
   CATBLK   I(256)  map header.
   CNTRBU   R(8192) buffers
Output:
   IERR     I   error code. 0 = ok.
                            9 => QUIT op received from TELL
                           10 => ABOR op received from TELL
```

11.6.6 CTICS

Writes tick marks and tick labels to a plot file. If CPREF(IAX) and CTYP(IAX) are all blank, no tick labels are done.

```
CTICS (LAXIS, BLC, TRC, XYRATO, YX, DOACRS, IBUFF, IERR)
Inputs:
   LAXIS   I        1 => horizontal,  2 => vertical full plots
                    3 => horiz subplot 4 => vertical subplot
   BLC     R(2)     X and Y pixels to form bottom left hand
                    corner of the graph.
   TRC     R(2)     X and Y pixels to form the top right hand
                    corner of the graph.
   XYRATO  R        X to Y scaling factor
   YX      D        LAXIS=3: plot x axis at y = YX; 4: plot y axis
                    at x = YX - out range => BLC(1,2) value
   DOACRS  L        Do coordinate grid (T) or just ticks (F)
In/out:
   IBUFF   I(256)   buffer being used for output to
                    the graphics file.
Outputs:
   IERR    I        error code: 0 => ok
                                1 => bad IAXIS
                                2 => graph drawing error
                                3 => tic algorithm fails
```

11.6.7 GCHAR

Will write a 'draw string' command record to a graph file. The output record description is:

```
I     opcode, 6 in this program.
I     number of characters (NCHAR).
I     angle (IANGL).
I     X offset in characters * 100  (DX * 100).
I     Y offset in characters * 100  (DY * 100).
I(*)  characters (STR).

GCHAR (NCHAR, IANGL, DX, DY, STR, BUFF, IERR)
Inputs:
   NCHAR  I       number of characters in STR
   IANGL  I       angle to print STR. 0=horizontal, 1=vertical.
   DX     R       x offset in characters from current position for
                  the bottom left corner of first character printed.
   DY     R       y offset in characters from current position for
                  the bottom left corner of first character printed.
   STR    C*(*)   string to be printed.
   BUFF   I(256)  buffer to use for I/O.
Output:
   IERR   I       error code:  0 => ok
                               1 => disk problems
                               2 => string too big
Common:
   GPHPOS  incremented by 5 + (NCHAR+3)/4.
   GPNRRN  incremented by 1 if a write to disk is needed.
```

11.6.8 GETROW

This routine will read a row of an image file that has been opened with and initialized with INTMIO. The routine will copy the row from the I/O buffer to the user buffer.

```
 GETROW (IMSTUF, IOBLK, ROW, EOF, IERR)
Inputs:
   IMSTUF    I(37)   IO pointers, LUNs, counters and such. They are
                     set in INTMIO.
In/out:
   IOBLK     R(*)    IO buffer.
Outputs:
   ROW       R(*)    Output row of image.
   EOF       L       TRUE means last row specified in INTMIO by the
                     BLC, TRC arguments has been read.
   IERR      I       Error code, 0=ok, others from MDISK.
```

11.6.9 GFINIS

Places an "end of plot" command in the buffer, writes the last buffer to disk, compresses the plot file if needed, and closes the plot file.
***** NOTE: any catalog operations for the plot file must be performed by the calling program. *******
The plot common is reinitialized in part. The command record has the form:

```
        I     opcode (32767)
```

```
GFINIS (BUFF, IERR)
In/out:
   BUFF  I(256)     plot file work buffer
Output:
   IERR  I          error code: 0 => ok
                                1 => disk error
                                2 => data error
                                3 => compress error
                                4 => close error
```

11.6.10 GINIT

Initializes the graphics common, creates and opens the graphics file, and places the "initialize plot" command in the file.
******* NOTE: cataloging of the graphics file must be handled by the calling program. *********
WARNING: Get PNAME right! If the create fails because the file already exists, GINIT will destroy PNAME and create the desired plot file. GINIT writes 1 or more records to disk containing the task name and parameters. It starts on the next physical record with a command record of the form:

```
I       opcode (1)
I       user number
I(3)    date:  19yy, mm, dd
I       ITYPE: type code 1 => misc.,  2 => CNTR,   3 => GREYS,
           4 => PROFL,  5 => SL2PL,  6 => PCNTR,  7 => IMEAN,
           8 => UVPLT,  9 => GNPLT, 10 => VBPLT, 11 => PFPLn,
          12 => GAPLT, 13 => PLCUB, 14 => IMVIM, 15 => TAPLT,
```

```
            16 => POSSM, 17 => SNPLT, 18 => KNTR,  19 => UVHGM,
            20 => ISPEC

   GINIT (IVOL, PNAME, IGSIZE, ITYPE, NPARM, PARMS, BUFF, LUN, FIND, IERR)
   Inputs:
      IVOL    I          disk volume for file creation
      PNAME   C*48       physical file name for created file
      IGSIZE  I          0 => small 1 => medium 2 => large file
      ITYPE   I          plot type code (if < 0, then ITYPE is taken to
                         be #rows (# cols if writing vertically) and is
                         used to adjust SCALEF to integer*abs(itype)
                         The output ITYPE is set to 1.
      NPARM   I          #  values in input parameter list.
      PARMS   R(NPARM)   task input parameter list (with defaults
                         filled in)
   In/out:
      BUFF    I(256)     plot commands are placed in this buffer and
                         written to disk as needed.  The calling program
                         must not alter this area between calls to GINIT
                         and GFINIS
   Outputs:
      LUN     I          logical unit number of plot file
      FIND    I          location in FTAB for file
      IERR    I          error code: 0 => ok
                            1 => another plot file open
                            2 => input data error
                            3 => create error (no file)
                            4 => open error (file gone)
```

11.6.11 GINITG

Will write an "init for gray-scale" command record to the graphics file. The output record has the form:

```
         I     opcode (3)
         I     lowest pixel value
         I     highest pixel value
         I     number of pixels on x axis
         I     number of pixels on y axis

   GINITG (IGLO, IGHI, BUFF, IERR)
   Inputs:
      IGLO  I          lowest allowed pixel value
      IGHI  I          highest allowed pixel value
   In/Out:
      BUFF  I(256)     graphics buffer
   Output:
      IERR  I          error code: 0 => ok
                                   1 => disk error
                                   2 => input error
```

11.6.12 GINITL

Will write an 'init for line drawing' command record to a graph file. common variables will also be initialized. The output record description is:

```
        I       opcode, 2 in this program.
        I       XYRATO * 100
        I       scale factor used in calculating I   X and Y in
                GPOS and GVEC. Equal to SCALEF.
        I(4)    BLC,TRC integer part (BLC rounded up, TRC down)
        I(4)    floating remainder * 1000 from BLC, TRC
        I(4)    10 * (CNOUT) number of characters
                outside pix area on left, bot, right, top, resp.
        I(5)    array location on axes 3,4,5,6,7

  GINITL (BLC, TRC, XYRATO, CHOUT, DEPTH, BUFF, IERR)
  Inputs:
     BLC     R(2)    number of first pixel in row & col
     TRC     R(2)    last pixel pos in row/col (usually 1 more
                     than the number of pixels)
     XYRATO  R       ratio between x and y axis plotting
     CHOUT   R(4)    number of characters outside pix on left,
                     bottom, right, top, resp.
     DEPTH   I(5)    array location on axes 3 thru 7
  In/Out:
     BUFF    I(256)  buffer to use for I/O.
  Output:
     IERR    I       error code. 0 = ok.
                                 1 => disk problems
  Common:
     GPHPOS  incremented by 20.
     GPHRRN  incremented by 1 if a write to disk is needed.
     GPHX1   set to X1 = BLC(1) (rounded up to integer)
     GPHX2   set to X2 = TRC(1) (rounded down to integer)
     GPHY1   set to Y1 = BLC(2) (rounded up to integer)
     GPHY2   set to Y2 = TRC(2) (rounded down to integer)
```

11.6.13 GMCAT

This routine will write a 'copy misc info into the image catalog' command record to a graph file. The output record description is:

```
        I        opcode, 8 for this record type.
        I        number of words.
        I(INO)   miscellaneous information.

  GMCAT (IHO, IBLK, IPBLK, IERR)
  Inputs:
     INO     I       number of words of misc info.
     IBLK    I(INO)  array containing misc info.
     IPBLK   I       plot file I/O block.
  Output:
     IERR    I       error code 0=ok, 1=no. of words over 20.
  Common:
     GPHPOS  I       incremented by 2 + INO.
```

```
   GPHRRN  I       incremented by 1 if write to disk necessary.
```

11.6.14 GPOS

Will write a 'position vector' command record to a graph file. The output record description is:

```
       I     opcode, 4 in this program.
       I     scaled X position.
       I     scaled Y position.

GPOS (X, Y, BUFF, IERR)
Inputs:
   X       R       x position
   Y       R       y position.
   BUFF    I(256)  buffer to use for I/O.
Output:
   IERR    I       error code. 0 = ok.
                               1 = disk problems.
Common:
   GPHPOS  incremented by 3.
   GPNRRN  incremented by 1 if a write to disk is needed.
```

11.6.15 GRAYPX

Places a "write string of gray values" command in the graphics file. The gray values are clipped using range given in the call to GINITG and placed in graphics file.
NOTE: this command may extend over more than 1 physical record in the file! The command form is:

```
       I          opcode (7)
       I          NPIX: number of values
       I          IANGL; 0 horizontal, 1 vertical
       I(NPIX)    clipped gray values

GRAYPX (NPIX, IANGL, IVALS, BUFF, IERR)
Inputs:
   NPIX   I          number of pixel values
   IANGL  I          direction code: 0 horizontal, 1 vertical
   IVALS  I(NPIX)    pixel values
In/out:
   BUFF   I(256)     graphics working buffer
Output:
   IERR   I          error code:  0 => ok
                                  1 => disk error
                                  2 => input data error
```

11.6.16 GVEC

Will write a 'write vector' command record to a graph file. The output record description is:

```
       I     opcode, 5 in this program.
       I     scaled X position.
       I     scaled Y position.
```

```
GVEC (X, Y, BUFF, IERR)
Inputs:
   X      R       x position
   Y      R       y position.
   BUFF   I(256)  buffer to nse for I/O.
Output:
   IERR   I       error code. 0 = ok.
                              1 = disk problems.
Common:
   GPHPOS  incremented by 3.
   GPHRRN  incremented by 1 if a write to disk is needed.
```

11.6.17 HIPLOT

Places one record in a file's HI extension file reporting that a plot extension was created by the present program.

```
HIPLOT (IVOL, ICNO, IVER, IBUF, IERR)
Inputs:
   IVOL   I         Disk volume
   ICNO   I         Catalog number
   IVER   I         Plot file version number created
Output:
   IBUF   I(256)    Scratch
   IERR   I         Error code: 0 => ok - don't quit on error
                            1 => no HI file
                            2,3,4 => open, add, close error
Uses LUN=29 and inits the HI common
```

11.6.18 INTMIO

This routine will open a map file, set values in common for use with close down routine DIE and set up two arrays containing all the values and counters needed by reading and writing routines compatible with this one.

```
INTMIO (ILUN, ACCESS, NAME, BLC, TRC, IBSIZE, CATBLK, IMSTUF, IERR)
Inputs:
   ILUN     I      Logical unit number to nse for the map file.
   ACCESS   C*4    'READ' or 'WRITE' status to mark catalog.
                   also 'HDWR' - mark write, open non-exclusive
   IBSIZE   I      Size of IO buffer in INTEGER values.
In/Out: (defaults filled in)
   NAME     C*36   "Name string" in the tradition of WaWa IO.
   BLC      R(7)   Bottom left corner of map.
   TRC      R(7)   Top right corner of map
Outputs:
   COMMON /CFILES/  Values updated so that subroutine DIE will
                    close this file.
   CATBLK   I(256)  Map header.
   IMSTUF   I(37)   IO pointers and stuff that are needed by other
                    IO routines compatible with this one. They are:
                   1. LUN
```

```
                  2. FTAB index
                  3.
                  4.
                  5. Catalog slot of image.
                  6. Size of IO buffer in bytes of all things.
                  7. Volume number of image.
                  8. Number of dimensions in image.
                  9. Number of values read per row of image.
                  10-16. Number of values along all 7 axis
                  17-30. Window in BLC TRC pairs along all 7 axis.
                  31-36. Current position on last six axis.
                  37 1 if read fwd -1 is bckwrd read on 2nd axis.
   IERR    I      Error code. 0=ok.
```

11.6.19 LABINI

Performs initializations for labeling: calls SETLOC, converts units (w METSCA) to get reasonable scaling, and, for LTYPEC = 4 (center-relative labeling), converts units ("lies") and prepares text giving true center position.

```
LABINI (BLC, TRC, IDEP, CH, ILTYPE, SLICE, YGAP, TEXT, NTEXT)
Inputs:
   BLC     R(2)      Bottom left corner
   TRC     R(2)      Top right corner
   IDEP    I(5)      Depth on axes 3 - 7
   ILTYPE  I         Labeling type: 1 none, 2 no ticks, 3 RA/dec,
                     4 center-relative, 5 subim center-rel, 6 pixels,
                     7 like 3, but no top line
   SLICE   L         T => for slice display: rotation will be the
                     slice PA - any map rotation; F => for maps: the
                     rotation is any map rot.
In/out:
   CH      R(4)      # chars outside plot to left, bot, right, top:
                     on in # extra for plot, on out totals
                     These must have input, reasonable values, LABINI
                     just adds a 0.5 border and room for the labeling.
                     IF CH(1) < -10., do not count axis label
                     characters with CHNTIC and CH(1) has little
                     meaning on output.
Output:
   YGAP    R         Char row position for 1st line of text at bottom
   TEXT    C(2)*80   Text to put at YGAP, used only if LTYPE=4
   NTEXT   I         Number of lines used in TEXT: 0, 1, 2
```

11.6.20 PLEND

Do some plotting cleanup functions. Write "end of plot" record, close plot file, check for vectors that were off the plot. Then terminates the task with a call to DIE.

```
PLEND (ISTAT, IDEBUG)
Inputs:
  ISTAT  I         0=successful completion, other=dies unnaturally.
```

```
   IDEBUG I        > 0 => don't delete PL file despite errors
```

11.6.21 PLGRY

This routine will put draw grey scale commands in the plot file.

```
PLGRY (IANGLE, NVAL, VALUES, IERR)
Inputs:
   IANGLE  I      Angle code. 0 = horizontal, 1 = vertical.
   NVAL    I      The number of grey scale pixel values.
   VALUES  R(*)   Grey scale values.
Output:
   IERR    I      Error code. 0=ok.
```

11.6.22 PLMAKE

This routine will create and open a plot file, put it in the map header and write the first record into the plot file. PLMAKE assumes that the map/uv file has been marked write and will change it to a read flag. It would be nice if the defaults had been filled into RPARM before this routine is called.

```
PLMAKE (NP, RPARM, IGTYPE, IERR)
Inputs:
   NP     I      Number of words in parameter list
   RPARM  R(NP)  AIPS parameters
   IGTYPE I      Plot file type: 1 misc., 2 CNTR, 3 GREYS, 4 PROFL,
                 5 SL2PL, 6 PCNTR, 7 IMEAN (hist), 8 UVPLT,
                 9 GNPLT, 10 VBPLT, 11 PFPL1, PFPL2, PFPL3.
                 Use 1 unless your inputs match those of these
                 tasks - or take a new number, but
                 AIPSUB:AU8A will need to know about it too.
Output:
   IERR   I      Error code. two digit, first digit indicates
                 subroutine: 1: MAPOPN, 2: MADDEX, 3: ZPHFIL,
                 4: GINIT, second digit indicates error code of
                 that subroutine.
```

11.6.23 PLPOS

This routine will put a 'position vector' command in a plot file.

```
PLPOS (X, Y, IERR)
Inputs:
   X       R    X value.
   Y       R    Y value.
   COMMON /PLTCOM/ (DPLT.INC)
Output:
   IERR    I    Error code.  0 means OK.
```

11.6.24 PLVEC

This routine will put a 'draw vector' command in a plot file. Vectors that are too big are interpolated.

```
PLVEC (X, Y, IERR)
Inputs:
   X       R    X value.
   Y       R    Y value.
Common:
   /PLTCOM/ (DPLT.INC)
Output:
   IERR    I    Error code.  0 means OK.
```

11.6.25 REIMIO

This routine will reinitialize the counters in IMSTUF for reading another subimage of a map opened and set up with INTMIO. All IMSTUF values that can be found in the header are re-initialized even if they are not changed by the standard routines.

```
REIMIO (BLC, TRC, IBSIZE, CATBLK, IMSTUF, IERR)
Inputs:
   BLC      R(7)    Bottom left corner of map.
   TRC      R(7)    Top right corner of map
   IBSIZE   I       Size of IO buffer in words.
   CATBLK   I(256)  Map header.
   IMSTUF   I(*)    (1) LUN
                    (2) FTAB index
                    (5) Catalog slot of image.
                    (6) Size of IO buffer in bytes of all things.
                    (7) Volume number of image.
Outputs:
   IMSTUF   I(*)    (8) Number of dimensions in image.
                    (9) Number of values read per row of image.
                    (10-16)  Number of values along all 7 axes
                    (17-30)  Window in BLC TRC pairs along all 7
                             axes.
                    (31-36)  Current position on last six axis.
                    (37) 1 if read forward, -1 if backward read on
                         2nd axis.
   IERR     I       Error code. 0=ok.
```

11.6.26 STARPL

Plots star positions in a plot file as given by an ST extension file of version VERS. The ST file contains the center position (RA-DEC, GLON-GLAT, ELON-ELAT) of each star and the "uncertainties" in those star positions. The plotted plus signs are scaled by these uncertainties and then further scaled by multiplying by FACTOR.

```
STARPL (FACTOR, IVOL, CNO, VERS, BLC, TRC, PLBUF, IERR)
Inputs:
   FACTOR  R         Star scaling factor: <= 0 => no plot.
   VERS    I         Desired ST file version number: 0 => high
   IVOL    I         File disk number
```

```
   CNO      I          File catalog number
   BLC      R(2)       Plot lower left corner (pixels)
   TRC      R(2)       Plot upper right corner (pixels)
In/Out:
   PLBUF    I(256)     Plot IO buffer
Output:
   IERR     I          Error code: 0 => okay
                                 -1 => there was no ST file
                                 +1 => logical error in ST file
                                 +2 => IO error in ST file
                                 +3 => IO error in plotting
Common:
   /MAPHDR/ CATBLK input   Image header having the ST file
```

Chapter 12

Using the Array Processors

12.1 Overview

Many of the more important of the AIPS tasks do a great deal of computation. The traditional approach to increasing the performance of a cpu is by means of hardware arithmetic units called Array Processors. These array processors (or APs) have their own memory and high speed, pipelined arithmetic hardware enabling them to run much faster than the host for certain specialized operations. Since not all computers running AIPS will have, or need, array processors attached, there is a library of Fortran routines which emulate the functions of the array processor; these routines, and a common in the host memory, constitute the "pseudo-array processor". Since the details of the implementation of these routines will depend on the hardware on which the software is run, these routines are explicitly machine dependent and have names beginning with the letter "Q"; thus the "Q-routines". This chapter will describe the use AIPS makes of array processors and explain how to use APs. At the end of this chapter is a list of the major Q routines with detailed comments on the call sequences.

Modern scientific computers are increasingly including vector hardware as an integral part of the CPU, eliminating the need for array processors. The compiler on vector computers knows about the vector capabilities, which greatly simplifies using the vector hardware. The "Q-routines" developed for array processors — or, more properly, the pseudo AP routines — have been successfully adapted to vector computers. Considerations for vectorization are discussed in a later section.

12.1.1 Why use the Array Processor?

The principal reason for using an array processor is speed. The design of most array processors optimizes its performance for repetitive arithmetic operations, making it much faster at vector arithmetic than the host CPU. Since most APs operate asynchronously from the host CPU, they constitute a co-processor which increases the capacity of the system.

A second advantage of using an array processor is that it contains its own local memory. On systems with limited physical memory, or address space, this can be an important consideration. It will be possible in the near future to get array processors, or fast CPUs, with many megawords of local memory. Such large memories will allow the use of more efficient methods of processing data.

12.1.2 When to Use, and Not to Use, the AP

The array processor is most efficient at very repetitive operations such as doing FFTs and multiplying large vectors. Its efficiency is greatly degraded for non-repetitive operations, or operations requiring a great number of decisions based on the results of computations. In fact, most array processors have very limited capability to make decisions based on the results of computations. Since the APs have their own program and data memory, the AP instructions and the data must be transferred to, and the results transferred from, the AP. These I/O operations may cost more cpu time than the amount saved by using the array processor. As a general rule, use of the AP is more efficient than the CPU when multiple or complex (such as FFT)

operations, which are highly repetitious, are going to be done on relatively large amounts of data (thousands of words or more). In other cases, using the AP will probably not help much and will keep other processes from using this valuable resource.

12.2 The AIPS Model of an Array Processor

The model of an array processor used is colored strongly by our early use of Floating Point Systems FPS AP-120B array processors. However, expressed in general terms, this model can be emulated on other real or virtual (pseudo) array processors. It should be noted that use of the APs requires vectorized programming; hence, implementation on super computers or other vector machines should be relatively efficient. The following describes the fundamental features of the AIPS model of array processors.

1. AIPS uses APs as detached vector arithmetic units. That is, data is sent into the AP, some (usually vector) operation is done, and the results are returned to the host CPU. The principal difficulty in the implementation of AIPS on other array or vector processors is that our concept of a vector operation is rather more general than that of most computing hardware manufacturers. Many of the more complex of the AIPS operations are better described as pipelined scalar operations. In the AIPS usage, high level control and use of disk storage is done in the host CPU and only arithmetic operations are done in the AP.

2. AIPS considers the AP to be a device which can be assigned via QINIT and deassigned via QRLSE. Basically, this means that data will not disappear from the task's assigned AP data memory between these calls. This concept has little meaning for virtual APs, except that the data memory is cleared after a QINIT call.

3. An AP should have a relatively large local data memory. The size of the AP data memory is obtained from a common set by ZDCHIN, which reads it from a disk file. The value in this disk file can be modified by the AIPS utility program SETPAR. In the case of pseudo (virtual) AP's, this memory is physically in the host CPU. A similar implementation could be done for an AP with significantly less capacity than an FPS AP-120B. In some vector implementations of the Pseudo AP the size of the memory given in the DDCH.INC include common is reset in QINIT.

4. In addition to data memory, the AP is assumed to have an array of 16 integer registers (SPAD) which can be read from the host CPU. These are used to communicate the addresses of maxima, minima, etc. This capability is not extensively used.

5. AIPS assumes that the array processor is programmable in that functions are used which are not now, or likely ever to be, in a standard library. If the AP is not programmable or is otherwise incapable of emulating one of the AIPS functions, then these functions must be performed in the host CPU and hidden from the AIPS routines. Alternately, these functions may be reformulated in terms of the functions available; this will be necessary for efficient implementation of long-vector super computers.

6. Communication with the AP by AIPS is via Fortran call statements which specify the data in the AP memory and other control information, transfer data between the AP and host CPU, or synchronize the operation of the AP and host CPU.

7. Data in the AP memory is specified by a base address and an increment. In current implementations these addresses are absolute, but this is not assumed. The calling process is assumed to have absolute control over an address space beginning at address 0 and extending to the address indicated in the device characteristic common (include DDCH.INC) as (1024*KAPWRD-1). Word addressing only is used. In pseudo AP implementations, further memory may be available; this additional memory is indicated by KAP2WD.

8. Many of the most crucial functions used by AIPS routines depend on data-dependent address generation and logic flow. As mentioned above, implementation of AIPS on an array processor without this capability will require reformulation of several of the algorithms (especially gridding and the in-core

CLEAN) in terms of vector operations. This reformulation will likely require vector logical operations, i.e., gather, scatter, merge, and compress operations.

9. AIPS assumes that the AP can handle either integer or real data values (with the same word size). Complex values consist of a pair of real values in adjacent locations, the first being the real part and the second being the imaginary part.

12.3 How to Use the Array Processor

Since the array processors used by AIPS have their own program and data memories, the instructions must be loaded in to the AP and data sent to, and results returned from, the AP. Since the AP runs asynchronously from the host cpu there must also be ways to synchronize the operations. The general operations are given in the following list, with the name of the subroutine AIPS uses for the given operation:

1. Assign / Initialize the AP. (QINIT)
2. Transfer data to the AP. (QPUT)
3. Wait for transfer to complete. (QWD, QWAIT)
4. Load and execute the AP program. (many)
5. Wait for computations to finish. (QWR, QWAIT)
6. Transfer data back to host cpu. (QGET)
7. Wait for transfer to complete. (QWD, QWAIT)
8. Release AP. (QRLSE)

12.3.1 AP Data Addresses

The AIPS convention for specifying data in the AP memory, which follows the Floating Point Systems (FPS) conventions, is to specify data by the zero-relative memory address of the first element in an array, the memory address increment between the elements of an array, and the number of elements in the array. On FPS APs, the memory address is an absolute address, but in implementations on other APs, the address may be a relative address, but this should be hidden from the programmer.

Q Routine Arguments

The call arguments to the Q routines (AP-routines) are integers. The exceptions to this are the host array names passed in QPUT and QGET. The FPS Q routines convert these to 16 bit unsigned integers.

Array Processor Memory Size

Since different array processors will have different memory sizes, the memory size of the AP is carried in the Device Characteristics Common which is obtained by the include DDCH.INC. The size of the AP is in the integer value KAPWRD as the multiple of 1024 words of AP data memory. Any operation with the AP should check that enough data memory is available and, if possible, scale the operation to make full use of the available memory.

Several Pseudo AP implementations have more memory available for certain operations. The amount of this secondary memory is given by KAP2WD. This memory is used as work space by some routines, but is generally available when these routines are not in use.

12.3.2 Assigning the AP

The array processor is assigned to the calling task using the AIPS routine QINIT. QINIT incorporates the AIPS priority system and provides for smooth use of the AP for batch tasks. The AIPS AP priority scheme is to give tasks with lower POPS numbers (the number at the end of the task name when it is running) higher priority. This is done by keeping a list of AP tasks in QINIT. When a task asks for an AP, QINIT then checks to see if any AP tasks with a lower POPS number are running; if so, QINIT suspends the task for a short period and then checks again. The number of times a task goes through the check/suspend loop before asking for the AP at the next opportunity is proportional to its POPS number. QINIT also sets values in common /BPROLC/ (include DBPR.INC which control the AP roller subroutine QROLL. The text of this include is shown at the end of this chapter and the use of the values is described in the detailed description of QROLL given at the end of this chapter. In some vector implementations of the Pseudo-AP QINIT also sets the size of the AP memory (KAPWRD, KAP2WD) in the DDCH.INC included common.

On some systems, batch AIPS tasks present more of a problem. AIPS batch tasks are usually run at lower priority than interactive tasks, so they may grab the AP and then not get enough cpu cycles to finish that AP operation for a very long time. To avoid this problem, QINIT increases the priority of the batch task to that of an interactive task while it has the AP.

QRLSE is used to deassign the AP. QRLSE also lowers the priority of batch tasks after the AP is released.

In the interest of a smooth and friendly system for users, it is important not to hog the AP for long periods of time. The priority system should then work to give lower POPS numbered AIPS users a larger fraction of the time, if they need the AP. A task should in general not keep the AP tied up for more than 5 to 10 minutes at a time, less if that is practical. For tasks which may need to keep the same data in the AP for long periods of time, such as tasks which compute models based on CLEAN components, there is an AP roller subroutine QROLL.

QROLL determines if it is time to roll out the AP based on values set by QINIT, and, if so, will create a scratch file (using the DFIL.INC system), copy the specified contents of the AP memory to a scratch file, release the AP, wait a short period of time, re-assign the AP, and load the previous contents back into the AP memory. Details of the call sequence to QROLL are found at the end of this chapter. IMPORTANT NOTE: QROLL (and APROLL) work properly only for floating point data. Integer values rolled will not be restored correctly.

12.3.3 Data Transfers To and From the AP

The fundamental routines for getting data to and from the Array Processor memory are QPUT and QGET; details of the call sequences can be found at the end of this chapter. In addition, for image-like data, there is the routine APIO.

APIO transfers image-like data between disk files and the array processor. The file open and close and initialization logic are all contained in this routine. Information about the file and the the desired properties of the I/O are passed to APIO in the array FLIST. APIO can access either cataloged 'MA' type files or scratch files using the DFIL.INC common system. APIO can handle arbitrary row lengths. This is done by breaking up the logical records if they are larger than the buffer size. It is **IMPORTANT** to always use the same size buffer when accessing a given file. Usage notes for APIO:

1. Opening the file.

   If APIO determines that the file is not open, it will open it. The file can be either a cataloged file or a scratch file using the DFIL.INC common system. If the catalog slot number given in FLIST is 0 or less, the file is assumed to be a scratch file. File open assumes that the file type is 'MA' (if cataloged); the file is opened patiently without exclusive use.

2. Initialization.

   APIO initializes the I/O using the values in FLIST when it opens the file. It may be initialized again at any time using OPCODE 'INIT'. Also, switching between 'READ' and 'WRIT' will force a flushing of the buffer ('WRIT') and initialization. Any initialization when the current operation is 'WRIT' will cause the buffer to be flushed.

3. Closing the file.

   The file may be closed with a call with opcode 'CLOS'. If the file is being written and a 'CLOS' call is issued, APIO will flush the buffer. This means that, if APIO is being used to write to a disk, it MUST be called with OPCODE='CLOS', 'READ', or 'INIT' to flush the buffer. NOTE: all pending AP operations MUST be complete before calling APIO with opcode 'CLOS'.

4. AP timing calls.

   APIO calls QWD before getting data from, or sending data to, the AP, but does not call QWR. The calling routine should call QWR as appropriate.

More details about the call arguments are found at the end of this chapter, and an example of the use of APIO is given in a later section.

12.3.4 Loading and Executing AP Programs

Loading and executing AP programs is done in a single call to the relevant routine. The call argument also includes the specification of the data, location of the output array, and any processing flags. A list of the AP routines currently supported in AIPS is found at the end of this chapter. If the function desired is not available, then it is possible to write it for the AP.

12.3.5 Timing Calls

Since array processors normally run asynchronously from the host, CPU timing calls are necessary. The subroutine calls basically suspend the operation of the calling program until the specified AP operation is completed. FPS claims that data transfers and computations (not involving the same AP memory) may be overlapped; however, the results of doing this are erratic and this practice should be avoided. On occasion, there appear to be timing problems whose symptoms are erratic and which produce very wrong results, which go away when apparently unnecessary timing calls are added, e.g., calls to QWR between calls to computation routines. We use three timing calls:

1. QWD suspends the calling program until data transfers to or from the AP are complete.
2. QWR suspends the calling program until the AP completes all computations.
3. QWAIT suspends the calling program until all data transfers and computations are complete.

12.3.6 Writing AP Routines

If the current library of AP routines does not contain the desired function, there are two possibilities for coding the function: (1) microcoding the routine or (2) using the Vector Functor Chainer (or equivalent on non-FPS APs) to combine existing functions to create the desired function. If either of these is chosen, the programmer should also write the corresponding pseudo-AP routines, if the task is likely to have general use. The name of the routine should start with the letter Q and be placed in the appropriate libraries.

For routines to be run on real array processors, there should be an intermediate routine that converts the integer addresses etc. to unsigned 2 byte integers. This routine should then call the actual AP routine.

In order to use microcode or Vector Function Chainer (VFC) routines, the following steps must be performed:

1. Compile VFC (or other high level language routines) to assembly (microcode) language. For FPS code, this is done by the FPS routine VFC.
2. Assemble microcode into machine code. For FPS code, this is done using APAL.
3. Link edit microcode routines together to make an executable module. For FPS code, this is done using APLINK, which creates a Fortran or host assembly language routine with the executable module in a data statement. Make sure the integer type declarations are correct. All integers to be passed to FPS handlers should be explicitly declared INTEGER*2.

4. Compile/assemble the Fortran/assembly language module and put in the appropriate subroutine link edit library.

It is beyond the scope of this manual to describe the use of the FPS or other AP software; the reader is referred to the appropriate manual provided by the AP vendor.

Microcoding Routines

It is beyond the scope of this manual to give details about microcoding for array processors; see the AP manuals for these details. The general principles of efficient microcoding are that several of the hardware units - address computation, floating add, floating multiply, and memory access - may be given instructions in a given cycle. In addition, the floating point hardware is pipelined. That is, even though it takes several cycles for an operation, it is broken up into several, single cycle steps and a new operation can be initiated each cycle.

This architecture allows for very efficient loops. The loop may be broken into several sections and one section from each of several passes through the loop may be processed in parallel. Efficient coding of loops may become very complicated, but careful coding may speed up the process by a factor of several to many. The source code for NRAO written microcode is kept in the file FPSSUB:NRAO.AP.

Vector Function Chainer

The principal purpose of the Vector Function Chainer is to combine a number of microcoded routines into a single AP call. This can greatly reduce the overhead of the host cpu talking to the AP; and, if the individual AP operations are relatively numerous and short, chaining routines can make a dramatic improvement in the speed of the overall process.

The Vector Function Chainer uses source code that looks vaguely like Fortran, but has very limited capabilities and essentially no access to the data memory. Hopefully, in the future, there will be efficient Fortran compilers for APs. (FPS has such a compiler for the 120B, but NRAO doesn't have a copy).

12.3.7 FFTs

One of the more common operations using the array processor is the Fast Fourier Transform (FFT). We have adopted the FPS convention for real-to-complex FFTs in packing the real part of the last complex value into the imaginary part of the first value in the array. This is allowed because the imaginary part of the first and last values are always zero. This convention allows the use of the same AP memory space for the input and output arrays from a real-to-complex FFT. For 2-D FFTs this convention is not followed and the Complex FFT of a Real N×M image is (M×2) × (N/2+1) in size.

We also adopt the convention for FFTs that the second half of a one dimensional array comes first, and that the center is N/2+1, where N is the number of elements in the array (always a power of two). In two dimensions, this means basically that the center of the array is at the corners with the first element of an NX x NY array being (NX/2+1, NY/2+1). An exception to this is that the AIPS two-dimensional FFT routine DSKFFT expects the normal order when transforming from the sky plane to the aperture plane (reverse transform).

The AIPS utility routine DSKFFT will FFT a two-dimensional array kept in a DFIL.INC system scratch file. Real-to-complex, complex-to-real, or full complex transforms can be done in either direction. Real-to-complex and complex-to-real transforms return the maximum and minimum values in the output array and real-to-complex transforms can return either the amplitude, real part, or full complex version of the result. Details of the call sequence for DSKFFT are given at the end of this chapter.

The FFT routines require data without blanking, in an array which is a power of two on a side. In addition, the center of an image in a cataloged file may not be in the required (NX/2+1, NY/2+1) position which will produce a phase ramp in the transformed array. Two AIPS utility routines are useful in this case (1) PEAKFN which finds the location of the peak of an image near the center (say of a dirty beam) and (2) PLNGET which will subimage a cataloged file, zero fill the excess, and rotate the center of the image. Detailed descriptions of these routines are given at the end of this chapter.

12.4 Pseudo-Array Processor and Vector Computers

Many modern scientific computing systems are becoming available which have integrated vector hardware and vectorizing compilers; there is little need for array processors on these machines. In addition, many older, scalar systems running AIPS have no array processor either. For these reasons, it is necessary to have software emulations of the functions of the array processor; this emulation of an array processor is called the Pseudo array processor (or Pseudo AP).

The pseudo AP consists of a Fortran common containing memory, which is used as the AP memory, and a set of routines, which perform the same operations on the contents of the "AP memory" as are done by the corresponding true AP routines. Because of the layering into the "Q-routines", the higher level routines need not know if they are using a true or pseudo AP.

There are a number of differences in programming an array processor and writing the same software in Fortran, especially using vectorizing compilers. Microcoding an AP is more flexible than using a vectorizing compiler, but is very much more difficult. On the other hand, memory is very limited on APs, but is more available to the CPU. The following sections will discuss some of the aspects of Q-routines on a vector computer; particular attention will be paid to issues relating to the performance of software.

12.4.1 Vectorization

What is Vectorization?

A vector operation is an operation on a one-dimensional array of values as a whole rather than as individual elements. In practice, the operations are done sequentially using pipelined hardware, but it is useful to think of the operations as being simultaneous. Vectorization is an operation performed by a compiler which takes code describing scalar operations and compiles instructions to do the same operation in vector hardware. In general, compilers will only vectorize explicit loops, so software to be vectorized needs to be cast into a form that the compiler will recognize.

Vector Hardware

The actual hardware used for vector operations differs greatly from one vendor to another, but the backbone of any vector unit is a pipelined arithmetic unit. The operations in this unit are divided into discrete stages (typically around 5) and, in each machine cycle, another operation is begun. Thus, once "the pipe gets rolling", a result comes out each cycle. In addition, different pipelines may run in parallel, either independently or with the output of one going into the other, a process called chaining (or a linked triad on CDC machines). The combination of these various forms of parallelism may result in a speedup of typically 10 to 20 over scalar operations (on IBM vector units this factor is about 4).

Another common, but not universal, feature of vector machines is the vector register (CDC machines don't have vector registers). A vector register is an array of high speed memory elements residing in the CPU, which are the source or destination of vector operations. Vector registers may be either fixed length (32 - 128 elements) or variable, with the number of vector registers being traded off against the length of each.

Vector Operations

There is no unique and universal set of vector operators, but there are a number which are sufficiently common and useful to warrant discussion. Vector operations can be classified into a number of categories: editing functions, arithmetic/logical functions, and reduction functions.

For this discussion, editing functions will include the operations of loading from, or storing data to, memory or of manipulating the elements of a vector. Operations in this class are summarized in the following:

1. LOAD: Load an array of values from memory into a vector register. There may be a constant stride or increment between elements in memory.

2. STORE: Store the contents of a vector register into memory. There may be a constant stride allowed.

3. GATHER: Load an array of values from memory whose addresses are specified into a register; the addresses are contained in another register.
4. SCATTER: Store the contents of a vector register in memory in a specified list of addresses.
5. COMPRESS: Remove elements from a vector based on an array of flags called the vector mask.
6. MERGE: Form a vector from two input vectors based on a vector mask.

Arithmetic/logical vector functions are arithmetic or logical operations on the elements of a vector for unary operations and between the corresponding elements for binary operations. Common operations in this class are given in the following:

1. + : Add the elements of two vectors.
2. − : Subtract the elements of two vectors.
3. * : Multiply the elements of two vectors.
4. AND : Logical AND the elements of two vectors.
5. OR : Logical OR the elements of two vectors.

Reduction operations are those which produce a scalar from a vector. The common reduction operations include the following:

1. SUM : Sum the elements of a vector
2. PROD : Multiply the elements of a vector
3. MIN : Find the minimum value in an vector
4. MAX : Find the maximum value in an vector

None of these operations are explicitly specified in Fortran (maybe in Fortran-8x), but must be recognized by the compiler. This requires both smart compilers and programmers who write in a style recognizable to the compiler. Fortunately, most vector compilers give reasonably understandable diagnostics when they don't vectorize a loop.

Difficulties with Vectorization

There are a number of typical barriers to efficient vectorization of which the programmer should be aware. Brief descriptions of some of these follow.

Dependencies A very common and serious problem in vectorization is the presence of dependencies. There is an implicit assumption in the function of vector hardware that each elemental operation in a vector operation is independent of all the others. An example where this is not true is the following:

```
      DO 10 LOOP = 2,LIMIT
         A(I) = A(I-1) + B(I)
10       CONTINUE
```

In this example, each input value of A is the result of the previous operation. This loop cannot vectorize as written.

A problem in vectorizing the AIPS Q routines is that all "AP memory" values are in the same array with pointers and increments being passed to each routine. Most compilers, when faced with this, will declare that an apparent dependency exists; there could, in fact, be dependencies if incorrect values of the pointers were passed. This problem is solved by including the ZVND.INC include before each loop not containing a real dependency. The contents of this include declares to the compiler that no dependency exists. All vectorizing compilers have such directives and several examples are given later in this chapter. It is very important not to declare that no dependency exists when one does; if you do, the compiler will believe you and you'll get what you've got coming. Another include, ZVD.INC may be used to inhibit vectorization if necessary.

Short Loops The efficiency of vectorization depends on the length of the vectors being operated on. There is a fixed time cost for starting an operation before the first result is available. For very long loops this cost is mostly hidden, for short loops this cost can dominate. The length of a vector which results in an effective speed per element of half of the maximum is referred to as the vector half length of the machine. This value varies from about 7 (Cray XMP) to 100 or more (some configurations of CDC Cyber). This parameter of the system is generalized to the description of short vector (Cray XMP, Convex C1) and long vector (CDC Cyber, Cray 2) machines.

For reasons given above, it is desirable to have loop lengths as long as possible. For nested loops, it is frequently possible to move the longest loop to the innermost loop, although smart compilers, such as the Convex compiler, will attempt to vectorize nested loops. The gather and scatter operations can frequently be used to make loops longer. Gather or scatter operations can usually be triggered using a Fortran array containing the indices in an array. A value named NSHORT in the DDCH.INC included common gives the shortest vector length that should be vectorized on the current hardware.

Branches in Loops Logic complications, such as branches inside of loops, may cause difficulties to compilers. Some older compilers refused to vectorize a loop which contained any IF statements; more recent compilers can vectorize statements of the form:

```
IF (A(I).GT.0.0) B(I) = A(I)
```

IF statements which may cause a branch are likely to inhibit vectorization. Whenever possible, such complications should be moved out of loops or replaced with Gather/Scatter and Compress operations.

Tuning

Most systems running on vector computers contain performance analysis tools (PROFILE on Convex, FLOWTRACE on Cray) which allow the programmer to determine how the CPU time in a program is being spent. Only routines which take a significant fraction of the total program execution time need to have much effort expended on them. A performance analyzer is an extremely useful tool for making a program run faster, although some care needs to be taken in interpreting the output of the performance analysis tools. The analysis process itself may distort the results; very short routines which are called very many times may appear to use significantly more time than is actually the case.

12.4.2 Memory Use

The following sections discuss the efficient use of memory both in general terms and in the AIPS pseudo-AP in particular.

Memory Heirarchy

All computer systems contain several types of data storage elements with different access times. The amount of storage in each category increases as the access time increases. The efficiency of a program will be seriously affected by the efficiency of use of these storage elements; the more often the desired data is in one of the shorter access time memory elements, the faster the program will run. After vectorization, efficient use of memory is the most important factor in determining the speed of the CPU for a given program. This issue is complicated by the fact that the distinction between different speed memories will be hidden by the compiler and operating system.

Not all systems have all types of memory described below, but all have some hierarchal structure. The various types of memory will be discussed in order of decreasing speed.

1. Registers

   The registers in the CPU are the highest speed memory in any system; usage of the registers is controlled by the compiler or by hand coding a routine in assembly language. Careful coding of loops can result in much of the work being done using data already in the registers, rather than all operations needing to access slower speed memory.

Many vector compilers use a "strip mining" technique in which a pass is made through entire loop, processing a number of elements equal to the vector register length each pass. This increases the amount of chaining possible and reduces the time spent waiting for data from slower access memory. The programmer can assist this by putting all related work not requiring logic flow control in one loop rather than several.

2. Cache memory

   Some computers have a limited amount of high speed memory called cache memory. When the cpu requests a value from memory, a block of data (typically 8 words) containing that value is read from the main memory into cache memory and then the referenced value is passed to the cpu. If the value referenced is already in cache memory, it is simply sent to the CPU. The computer system will assure that the contents of main memory track those in cache.

   A parameter used to describe the efficiency of the use of cache is the "cache hit rate", which is the percentage of the time the datum desired resided in the cache. The cache hit rate can be increased by noting that an entire block of memory was copied to cache; thus, sequential access of memory will increase the cache hit rate. In Fortran arrays, the first axis varies the most rapidly in memory; therefore, inner loops over the first axis are the most efficient.

3. Memory

   All systems contain relatively large amounts of storage elements called, simply, the memory. These are mostly semiconductor devices, although older systems used magnetic doughnuts called cores; memory is sometimes referred to as "core memory". This type of storage element will be referred to as physical memory in the following.

4. Virtual Memory

   A number of systems increase the apparent size of a program's memory by means of virtual memory. Virtual memory consists of data storage elements which reside outside of the physical address space of the program. Such storage is referenced in units of pages (typically 512 to 4096 bytes) and is transferred into the programs addressable physical memory when first referenced (called a "page fault"). When the maximum allowable amount of physical memory is reached by a program, the page of memory least recently accessed is removed from physical to virtual memory.

   The physical storage of data in virtual memory may be one or more of several forms. If the requested page has been used before and still exists in the systems semiconductor memory, that page will be restored to the program's physical memory. Some systems (IBM vector machines) may contain a large amount of semiconductor memory exclusively for this use. Virtual memory pages not kept in semiconductor memory will be stored on a mass storage device such as a magnetic disk.

   Virtual memory is best used in the way overlaying is used on machines without virtual memory; once a page is read in, it is used many times and is only removed from physical memory after its use has (at least temporarily) ended. Badly coded routines may page fault on every line of code; this is likely to occur in multi-dimensional arrays larger than will fit in the program's physical memory and the inner loop is over the last axis. *Programs with a very high page fault rate will run very slowly and may destroy the response of the system for all users.* Virtual memory abuse is one of the most serious and antisocial crimes a programmer can commit.

Work Vectors

In synthesis data processing, there are a number of operations which do not vectorize in the forms they are originally specified. A prime example of this is the gridding of uv data, which is discussed in detail in AIPS Memo No. 33. For these difficult cases, it is frequently possible to reformulate the algorithm into a vectorizable form using temporary arrays, such as addresses for gather/scatter operations. These arrays are referred to as work vectors in the following. This technique has been heavily used in the specialized Q routines for various machines. There are several WKVECn work vectors located in the common containing the "AP memory" (DAPC.INC. In all vector machine pseudo APs, the memory in these work arrays greatly exceeds the memory in the primary "AP memory".

Extension of AP Memory

Since there is considerable memory allocated to the work vectors, which are used only for specialized routines, it is desirable to use this memory for other purposes. To this end, the work arrays are placed in the same common after the main "AP memory" in the DAPC.INC include. The amount of this memory (referred to as secondary AP memory) is given in the DDCH.INC include common as KAP2WD in units of 1024 real words. Before using this secondary AP memory, the programmer should be sure that it is not also being used as work vectors; none of the routines described in this chapter use work vectors overlapping the secondary AP memory.

Memory allocation

On computers without virtual memory, all of a program must fit in physical memory; on some of these, Crays in particular, memory is an expensive resource. In addition, programs requiring large amounts of memory will spend most of the time rolled out, if the machine is busy (which Crays usually are). For these reasons, it is desirable to be able to reduce or expand the amount of memory used for the AP on Cray XMPs. This is done using blank common, which Cray will allow to be dynamically allocated, for the "AP" memory. The routine QMEMSZ is then used to allocate or deallocate memory. Since this common will be loaded in its fully expanded form, it is a good idea to call QRLSE as the first executable statement in tasks using the AP to release the AP memory.

12.5 Example of the Use of the AP

In the following example of the use of the array processor, the elements of two scratch files containing arrays N x M using the DFIL.INC system (numbers ISCRA and ISCRB) are added and returned to the file ISCRC. This makes very inefficient use of the AP, but illustrates the basic features. This example also illustrates use of APIO.

```
      SUBROUTINE FILADD (ISCRA, ISCRB, ISCRC, N, M, IRET)
C-----------------------------------------------------------------------
C   FILADD adds two N x M arrays in the DFIL.INC scratch files
C   ISCRA and ISCRB and writes the result in scratch file ISCRC.
C   Inputs:
C      ISCRA     I  DFIL.INC scratch file number of first input file.
C      ISCRB     I  DFIL.INC scratch file number of second input file.
C      ISCRC     I  DFIL.INC scratch file number of output file.
C      N         I  Length of a row in the array
C      M         I  Number of rows in the array.
C   Output:
C      IRET      I  Return error code 0=>OK, otherwise APIO error code.
C-----------------------------------------------------------------------
      INTEGER ISCRA, ISCRB, ISCRC, N, M, IRET
C
      INTEGER   INCR, FLIST(22,3), LOOP, APLOCA, APLOCB, APLOCC, LEN,
      REAL      BUFF1(4096), BUFF2(4096), BUFF3(4096)
C-----------------------------------------------------------------------
C                                       Setup for APIO
      CALL FILL (22, 0, FLIST)
C                                       Size of array
      FLIST(5,1) = N
      FLIST(6,1) = M
C                                       Buffer size (4096 words)
      FLIST(13,1) = 4096 * 2
C                                       Copy for other files
```

```
      CALL COPY (22, FLIST(1,1), FLIST(1,2))
      CALL COPY (22, FLIST(1,1), FLIST(1,3))
C                                     Set LUNs
      FLIST(1,1) = 16
      FLIST(1,2) = 17
      FLIST(1,3) = 18
C                                     Set DFIL.INC file numbers
      FLIST(2,1) = ISCRA
      FLIST(2,2) = ISCRB
      FLIST(2,3) = ISCRC
C                                     Set AP pointers,
      APLOCA = 0
      LEN = N
C                                     Address for B file
      APLOCB = APLOCA + LEN
C                                     Address for C file
      APLOCC = APLOCB + LEN
C                                     Grab AP
      CALL QINIT (0, 0, KAP)
C                                     Start loop.
      DO 100 LOOP = 1,M
C                                     File A to AP
         CALL APIO ('READ', FLIST(1,1), APLOCA, BUFF1, IRET)
C                                     Check for error
         IF (IRET.NE.0) GO TO 999
C                                     File B to AP
         CALL APIO ('READ', FLIST(1,2), APLOCB, BUFF2, IRET)
C                                     Check for error
         IF (IRET.NE.0) GO TO 999
C                                     Wait for data transfer
         CALL QWD
C                                     Add
         CALL QVADD (APLOCA, 1, APLOCB, 1, APLOCC, 1, LEN)
C                                     Wait for operation to finish
         CALL QWR
C                                     Write result to disk.
         CALL APIO ('WRIT', FLIST(1,3), APLOCC, BUFF3, IRET)
C                                     Check for error
         IF (IRET.NE.0) GO TO 999

 100     CONTINUE
C                                     Release the AP
      CALL QRLSE
C                                     Close files.
      CALL APIO ('CLOS', FLIST(1,1), APLOCA, BUFF1, IRET)
C                                     Check for error
      IF (IRET.NE.0) GO TO 999
      CALL APIO ('CLOS', FLIST(1,2), APLOCB, BUFF2, IRET)
C                                     Check for error
      IF (IRET.NE.0) GO TO 999
      CALL APIO ('CLOS', FLIST(1,3), APLOCC, BUFF3, IRET)
C
 999  RETURN
      END
```

12.6 INCLUDEs

12.6.1 DAPC.INC

Several versions of this include exist for the pseudo AP implementations on different systems. This INCLUDE defines the pseudo AP memory.

```
C                                                          Include DAPC
C                                        'Vanilla' Pseudo AP version.
      REAL   APCORE(65536), RWORK(4096)
      INTEGER   APCORI(1), IWORK(4096), SPAD(16)
      COMPLEX CWORK(2048)
      COMMON /APFAKE/RWORK, APCORE
      COMMON /SPF/ SPAD
      EQUIVALENCE (APCORE, APCORI), (RWORK, IWORK, CWORK)



C                                        Convex C1 Version
      INTEGER   APSIZE, PKPWRD, PKPWD2
C                                        "1 MWord" size
      PARAMETER (APSIZE=262144)
C                                        "256 KWord size"
C      PARAMETER (APSIZE=65536)
C                                        PKPWRD="primary" AP size
      PARAMETER (PKPWRD=64)
C                                        PKPWD2="secondary" memory
      PARAMETER (PKPWD2=((APSIZE*5)-PKPWRD*1024)/1024)
      REAL   APCORE(APSIZE+1), WKVEC1(APSIZE/2+1), WKVEC2(APSIZE/2+1),
     *   WKVEC3(APSIZE/2+1), WKVEC4(APSIZE/2+1), WKVEC5(APSIZE/2+1),
     *   WKVEC6(APSIZE/2+1), WKVEC7(APSIZE/2+1), WKVEC8(APSIZE/2+1),
     *   WKVEC9(APSIZE/2+1)
      INTEGER   APCORI(1), SPAD(16),
     *   IWVEC1(APSIZE/2+1), IWVEC2(APSIZE/2+1), IWVEC3(APSIZE/2+1),
     *   IWVEC4(APSIZE/2+1), IWVEC5(APSIZE/2+1), IWVEC6(APSIZE/2+1),
     *   IWVEC7(APSIZE/2+1), IWVEC8(APSIZE/2+1), IWVEC9(APSIZE/2+1)
      COMMON /APFAKE/ APCORE, WKVEC1, WKVEC2, WKVEC3, WKVEC4, WKVEC5,
     *   WKVEC6, WKVEC7, WKVEC8, WKVEC9
      COMMON /SPF/ SPAD
      EQUIVALENCE (APCORE, APCORI),
     *   (WKVEC1, IWVEC1), (WKVEC2, IWVEC2), (WKVEC3, IWVEC3),
     *   (WKVEC4, IWVEC4), (WKVEC5, IWVEC5), (WKVEC6, IWVEC6),
     *   (WKVEC7, IWVEC7), (WKVEC8, IWVEC8), (WKVEC9, IWVEC9)



C                                        Alliant FX Version
      INTEGER   APSIZE, FFTSZE
      PARAMETER (APSIZE=65536)
      REAL   APCORE(APSIZE+1), WKVEC1(APSIZE/2+1), WKVEC2(APSIZE/2+1),
     *   WKVEC3(APSIZE/2+1), WKVEC4(APSIZE/2+1), WKVEC5(APSIZE/2+1),
```

```
     *   WKVEC6(APSIZE/2+1), WKVEC7(APSIZE/2+1), WKVEC8(APSIZE/2+1),
     *   WKVEC9(APSIZE/2+1)
      INTEGER   APCORI(1), SPAD(16),
     *   IWVEC1(APSIZE/2+1), IWVEC2(APSIZE/2+1), IWVEC3(APSIZE/2+1),
     *   IWVEC4(APSIZE/2+1), IWVEC5(APSIZE/2+1), IWVEC6(APSIZE/2+1),
     *   IWVEC7(APSIZE/2+1), IWVEC8(APSIZE/2+1), IWVEC9(APSIZE/2+1)
      COMPLEX
     *   CWVEC1(APSIZE/4+1), CWVEC2(APSIZE/4+1), CWVEC3(APSIZE/4+1),
     *   CWVEC4(APSIZE/4+1), CWVEC5(APSIZE/4+1), CWVEC6(APSIZE/4+1),
     *   CWVEC7(APSIZE/4+1), CWVEC8(APSIZE/4+1), CWVEC9(APSIZE/4+1)
      COMMON /APFAKE/ APCORE, WKVEC1, WKVEC2, WKVEC3, WKVEC4, WKVEC5,
     *   WKVEC6, WKVEC7, WKVEC8, WKVEC9
      COMMON /SPF/ SPAD, FFTSZE
      EQUIVALENCE (APCORE, APCORI),
     *   (WKVEC1, IWVEC1, CWVEC1), (WKVEC2, IWVEC2, CWVEC2),
     *   (WKVEC3, IWVEC3, CWVEC3), (WKVEC4, IWVEC4, CWVEC4),
     *   (WKVEC5, IWVEC5, CWVEC5), (WKVEC6, IWVEC6, CWVEC6),
     *   (WKVEC7, IWVEC7, CWVEC7), (WKVEC8, IWVEC8, CWVEC8),
     *   (WKVEC9, IWVEC9, CWVEC9)
C                                                          End DAPC
```

12.6.2 DBPR.INC

```
C                                                          Include DBPR
C                                        AP roller common
      REAL      DELAY
      DOUBLE PRECISION XTLAST, DELTIM
      LOGICAL   TRUEAP
      COMMON /BPROLC/ XTLAST, DELTIM, DELAY, TRUEAP
C                                                          End DBPR.
```

12.6.3 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(8,16), 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, NBLANK,
     *   NVOL, NBPS, NSPG, NBTB1, NTAB1, NBTB2, NTAB2, NBTB3, NTAB3,
```

```
     *   NTAPED, CRTMAX, PRTMAX, NBATQS, MAXXPR, CSIZPR, NINTRN,
     *   KAPWRD, NWDPDP, NBITWD, NCHLIN, NTVDEV, HTKDEV, BLANKV,
     *   NTVACC, HTKACC, UCTSIZ, BYTFLP, USELIM, NBITCH, NCHPRT,
     *   KAP2WD, MAXXTK, CSIZTK, DASSGN, DEVTAB, SPFRMT, DPFRMT,
     *   NSHORT, TTYCAR
      COMMON /FTABCM/ FTAB
C                                                        End DDCH.
```

12.6.4 ZVND.INC

This include is a vectorizing compiler directive to ignore the apparent dependency in the following loop; several versions are given:

```
C                                       Include for compiler directive
C                                       to ignore apparent dependency.
C                                       Dummy version (nonvectorizing)

C                                       Convex version
C$DIR NO_RECURRENCE

C                                       COS version
CDIR$  IVDEP

C                                       Alliant version
CVD$L NODEPCHK
CVD$L NOSYNC
```

12.6.5 ZVND.INC

This include is a vectorizing compiler directive to assert that there is a dependency in the following loop; several versions are given:

```
C                                       Force scalar compilation
C                                       compiler directive
C                                       Dummy version (nonvectorizing)

C                                       Convex version
C$DIR SCALAR

C                                       COS version
CDIR$  NEXTSCALAR

C                                       Alliant version
CVD$L NOVECTOR
```

12.7 Routines

12.7.1 Utility Routines

APCONV

Is a disk based, two dimensional convolution routine. The image to be convolved and the FFT of the convolving function are passed to APCONV along with two scratch files. NOTE: Uses AIPS LUNs 18, 23, 24, 25.

```
APCONV (NX, NY, LI, LW1, LW2, LO, LC, FACTOR, JBUFSZ, BUFF1, BUFF2,
  *   BUFF3, SMAX, SMIN, IERR)
Inputs:
   NX        I     The number of columns in the input image (must be
                   a power of 2).
   NY        I     The number of rows in the input image.
   LI        I     File number in /CFILES/ of input.
   LW1       I     File number in /CFILES/ of work file no. 1
                      size = (4*NX x NY+2).
   LW2       I     File number in /CFILES/ of work file no. 1
                      size = (4*NX x NY+2).
   LO        I     File number in /CFILES/ of output.
   LC        I     File number in /CFILES/ of FFT of convolving fn.
                      size = (4*NX x NY+2).
   FACTOR    R     Normalization factor for convolving function; i.e.
                   is multiplied by the transform of the convolving
                   function
   JBUFSZ    I     Size of BUFF1,2,3 in AIPS bytes.  Should be
                   large, at least 8192 words.
Output:
   BUFF1     R(*)  Working buffer
   BUFF2     R(*)  Working buffer
   BUFF3     R(*)  Working buffer
   SMAX      R     Maximum value in the output file.
   SMIN      R     Minimum value in the output file.
   IERR      I     Return error code, 0 => OK, otherwise error.
```

APIO

Transfers image-like data between disk files and the array processor. The file open and close and initialization logic are all contained in this routine. Information about the file and the the desired properties of the I/O are contained in the array FLIST. APIO can access either cataloged 'MA' type files or scratch files using the /CFILES/ common (DFIL.INC) system.

APIO can handle arbitrary row lengths with any size buffer larger than one disk block. This is done by breaking up the logical records. NOTE: it is important that data read with APIO have been written by APIO using the same buffer if the buffer is shorter than the row size. The problem is that APIO will break up logical records if they are longer than the buffer size and MDISK may leave blank space on the disk if the shorter logical record does not fill a disk sector.

Useage notes:

1. Opening the file. If APIO determines that the file is not open it will do so. The file can be either a cataloged file or a scratch file using the /CFILES/ common system. If the catalog slot number given in FLIST is 0 or less the file is assumed to be a scratch file. File open assumes that the file type is 'MA' (if cataloged), file is opened patiently without exclusive use.

2. Initialization. APIO initializes the I/O using the values in FLIST when it opens the file. It may be initialized again at any time using OPCODE 'INIT'. Also switching between 'READ' and 'WRIT' will force flushing the buffer ('WRIT') and initialization. Any initialization when the current operation is 'WRIT' will cause the buffer to be flushed.

3. Closing the file. The file may be closed with a call with opcode 'CLOS'. If the file is being written and a 'CLOS' call is issued, APIO will flush the buffer. This means that if APIO is being used to write to a disk it MUST be called with OPCODE='CLOS','READ', or 'INIT' to flush the buffer. NOTE: All pending AP operations MUST be complete before calling APIO with opcode 'CLOS'.

4. AP timing calls. APIO calls APWD before getting data from or sending data to the AP but does not call APWR. The calling routine should call APWR as appropriate.

```
APIO (OPCODE, FLIST, APLOC, BUFFER, IRET)
 Inputs:
    OPCODE    C*4  Code for the desired operation.
                   'INIT' forces the initialization of the I/O.
                   'READ' reads a logical record from the disk and
                          sends it to the specified AP location.
                   'WRIT' Gets data from the AP and writes it to
                          disk.
                   'CLOS' Closes the file and flushes the buffer if
                          necessary.
    FLIST(22) I    An array containing information about the file
                   and the I/O. Parts are to be filled in by the
                   calling routine and are for use by APIO.
                     1 = LUN, must be filled in,
                     2 = disk number for catalogs files or
                         /CFILES/ number for scratch files.
                     3 = catalog slot number for cataloged files,
                         .LE. 0 indicates that the file is a scratch
                         file.
                     4 = Unused
                     5 = Length of a logical record (row) in pixels.
                     6 = Number of rows in a plane.
                     7 = value to be added to 1 for the block offset.
                     9-12 = the window desired in the image, 0's=>
                         all of image.  The logical records must fit
                         in the buffer and be smaller than BUFSZ
                         bytes to subimage rows.
                     13 = Buffer size in bytes.
        Used by APIO:
                     14 = FTAB pointer
                     15 = Number of MDISK calls per logical record.
                     16 = Current OPCODE,
                          0 = none, INIT on next call
                          1 = READ
                          2 = WRITE
                     17 = actual length of logical row.
                     18-22 = Spare.
    APLOC       I   Base address in AP for data.
    BUFFER(*)   R   Working buffer.
 Output:
    IRET        I   Return code, 0 => OK or
                                 1 = Bad OPCODE,
                                 2 = Attempt to window too large
                                     a file.
                                 3 = Buffer too small (<NBPS bytes)
                                 MDISK error codes + 10, or
                                 MINI3 error codes + 20, or
                                 ZOPEN error codes + 30.
```

DSKFFT

Is a disk based, two dimensional FFT. If the FFT all fits in AP memory then the intermediate result is not written to disk. Input or output images in the sky plane are in the usual form (i.e. center at the center, X the first axis). Input or output images in the uv plane are transposed (v the first axis) and the center-at-the-edges convention with the first element of the array the center pixel. NOTE: Uses AIPS LUNs 23, 24, 25.

```
DSKFFT (NR, NC, IDIR, HERM, LI, LW, LO, JBUFSZ, BUFF1, BUFF2,
  *   SMAX, SMIN, IERR)
Inputs:
   NR       I      The number of rows in input array (# columns in
                   output).  When HERM is TRUE and IDIR=-1, NR is
                   twice the number of complex rows in the input file
   NC       I      The number of columns in input array (# rows in
                   output).
   IDIR     I      1 for forward (+i) transform, -1 for inverse (-i)
                   transform.
                   If HERM = .TRUE. the follwing are recognized:
                      IDIR=1 keep real part only.
                      IDIR=2 keep amplitudes only.
                      IDIR=3 keep full complex (half plane)
   HERM     L      When HERM = .FALSE., this routine does a complex to
                   complex transform.
                   When HERM = .TRUE. and IDIR = -1, it does a
                   complex to real transform.  When HERM = .TRUE. and
                   IDIR = 1, it does real to complex.
   LI       I      File number in /CFILES/ of input.
   LW       I      File number in /CFILES/ of work file (may equal LI)
   LO       I      File number in /CFILES/ of output.
   JBUFSZ   I      Size of BUFF1, BUFF2 in bytes.  Should be large
                   at least 4096 words.
Output:
   BUFF1    R(*)   Working buffer
   BUFF2    R(*)   Working buffer
   SMAX     R      For HERM=.TRUE. the maximum value in output file.
   SMIN     R      For HERM=.TRUE. the minimum value in output file.
   IERR     I      Return error code, 0 => okay, otherwise error.
```

PEAKFN

Searches a region around the center of an image to locate the pixel location of the maximum. Will handle data cubes.

```
PEAKFN (LUN, VOL, CNO, IDEPTH, CATBLK, BUFFER, JBUFSZ, PEAKX, PEAKY,
  *   IRET)
Inputs:
   LUN      I        Logical unit number to use.
   VOL      I        Disk on which image resides.
   CNO      I        Catalog slot number of image.
   IDEPTH   I(5)     Depth in image of desired plane.
   CATBLK   I(256)   Catalog header block for image.
   JBUFSZ   I        Size of the BUFFER in bytes
Output:
   BUFFER   R(*)     Real work buffer
```

```
   PEAKX     R        X coordinate of peak pixel location.
   PEAKY     R        Y coordinate of peak pixel location.
   IRET      I        Return code, 0=> OK, otherwise error.
```

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 NX/2+1 (NX 1st dim. of input win.)
   YOFF      I       offset in cells in second dimension of the center
                     from NY/2+1 (NY 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
                      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)
   /MAPHDR/ CATBLK  is set to the input file CATBLK.
```

QROLL

If it is time for the current task to roll out of the AP then QROLL copies the first NWORDs of AP MD memory to a scratch file, gives up the AP, does a task delay for DELAY, grabs an AP and loads the scratch file back into the AP. If NWORD ≥ 0, then the AP is not rolled and the AP is given up and then reassigned.

NOTE: APROLL is called by QROLL and uses commom /CFILES/ for the scratch file. NOTE: LUN 8 is used for I/O and a AIPS "map" I/O slot is opened if the AP memory is actually rolled.

No action is taken if the task is using a "pseudo" AP.

IMPORTANT NOTE: QROLL (and APROLL) work properly only for floating point data. Integer values rolled will not be restored correctly.

```
QROLL (NWORD, BUFFER, BUFSZ, IRET)
Inputs:
   NWORD    I       Number of words of AP memory to save.
                    If <= 0 the contents of the AP memory are
                    not saved.
   BUFFER   R(*)    Work buffer.
   BUFSZ    I       Size of BUFFER in bytes.
Inputs from COMMON /BPROLC/ (DBPR.INC)  (set by QINIT)
   TRUEAP   L       True if a real AP (to be rolled)
   XTLAST   D       Real time AP assigned (min).
   DELTIM   D       Time interval between rolls (min).
   DELAY    R       Time to delay task (seconds).
Output:
   IRET     I       Return error code, 0 => OK
                      2 => couldn't reload AP.
```

12.7.2 Array Processor Routines

The names and functions of the general purpose AP routines are given in the following brief list. A number of specialized routines for CLEANing, gridding uv data and model computations have been omitted.

1. QGET (HOST, AP, N, TYPE) Transfers data from AP to host.
2. QGSP (I, NREG) Reads the value of an SPAD register (FPS and pseudo).
3. QPUT (HOST, AP, N, TYPE) Transfers data from host to AP.
4. QRFT (UDATA, UFT, UPH0, NFT, NDATA) Computes real, inverse Fourier transform from arbitrarily spaced data.
5. QWAIT (no arguments) Suspends host until all transfers and computations are complete.
6. QWD (no arguments) Suspends host until all transfers of data are complete.
7. QWR (no arguments) Suspends host until all computations are complete.
8. QBOXSU (A, I, NB, C, J, N) Does a boxcar sum on a vector.
9. QINIT (I1, 12, I3) Assigns and initializes AP.
10. QRLSE (no arguments) Releases the AP.
11. QCFFT (C, N, F) Complex FFT.
12. QCRVMU (A, I, B, J, C, K, N) Complex - real vector multiply.
13. QCSQTR (CORNER, SIZE, ROW) In-place transpose of square complex matrix.
14. QCVCMU (A, I, B, C, J, N) Multiplies a complex scalar times the complex conjugate of a complex vector producing a real vector.
15. QCVCON (A, I, C, K, N) Takes complex conjugate of complex vector.

16. QCVEXP (A, I, C, K, N) Takes complex exponential of real vector.
17. QCVJAD (A, I, B, J, C, K, N) Adds a complex vector to the complex conjugate of another complex vector.
18. QCVMAG (A, I, C, K, N) Complex vector magnitude squared.
19. QCVMMA (A, I, C, N) Finds the maximum square modulus of a complex vector.
20. QCVMOV (A, I, C, K, N) Copies one complex vector to another.
21. QCVMUL (A, I, B, J, C, K, N, F) Multiplies two complex vectors.
22. QCVSDI (A, I, B, C, J, N) Divides a weighted complex vector by a complex scalar, weight is multiplied by the amplitude of the scalar.
23. QCVSMS (A, I, B, C, J, D, K, N, FLAG) Subtracts a real vector times a complex scalar from a complex vector.
24. QDIRAD (A, IA, B, N) Complex directed add.
25. QHIST (A, I, C, N, NB, AMAX, AMIN) Computes histogram of a vector.
26. QLVGT (A, I, B, J, C, K, N) Logical vector greater than.
27. QMAXMI (A, I, MAX, MIN, N) Finds maximum and minimum values in a vector.
28. QMAXV (A, I, C, N) Finds maximum in an array.
29. QMEMSZ (NWORDS) Expands or contracts the size of the Pseudo AP memory.
30. QMINV (A, I, C, N) Finds minimum in an array.
31. QMTRAN (A, I, C, K, MC, NC) Matrix transpose.
32. QPHSRO (A, I, B, J, PHAS0, DELPHS, N) Imposes a phase gradient on a complex vector.
33. QPOLAR (A, I, C, K, N) Rectangular to polar conversion.
34. QRECT (A, I, C, K, N) Polar to rectangular conversion.
35. QRFFT (C, N, F) Real to complex, or vice versa, Fast Fourier Transform.
36. QSVE (A, I, C, N) Sum of vector elements.
37. QSVESQ (A, I, C, N) Sum of the square of the elements of a vector.
38. QVABS (A, I, C, K, N) Vector absolute value.
39. QVADD (A, I, B, J, C, K, N) Vector add.
40. QVCLIP (A, I, B, C, D, L, N) Vector clip.
41. QVCLR (C, K, N) Vector clear.
42. QVCOS (A, I, C, K, N) Vector cosine.
43. QVDIV (A, I, B, J, C, K, N) Vector division.
44. QVEXP (A, I, C, K, N) Vector exponentiation.
45. QVFILL (A, C, K, N) Vector fill.
46. QVFIX (A, I, C, K, N) Vector real to integer.

47. QVFLT (A, I, C, K, N) Vector integer to real.
48. QVIDIV (A, I, D1, D2, B, J, N) Divides a vector by the product of two scalar integers.
49. QVLN (A, I, C, K, N) Vector natural logarithm.
50. QVMA (A, I, B, J, C, K, D, L, N) Vector multiply and add.
51. QVMOV (A, I, C, K, N) Copies one vector to another.
52. QVMUL (A, I, B, J, C, K, N) Vector multiply.
53. QVNEG (A, I, C, K, N) Takes negative of a vector.
54. QVRVRS (C, K, N) Reverses a vector.
55. QVSADD (A, I, B, C, K, N) Vector scalar add.
56. QVSIN (A, I, C, K, N) Vector sine.
57. QVSMA (A, I, B, C, K, D, L, N) Vector scalar multiply and add.
58. QVSMAFX (A, I, B, C, D, L, N) Vector scalar multiply, add and fix.
59. QVSMSA (A, I, B, C, D, L, N) Vector scalar multiply, scalar add.
60. QVSMUL (A, I, B, C, K, N) Vector scalar multiply.
61. QVSQ (A, I, C, K, N) Vector square.
62. QVSQRT (A, I, C, K, N) Vector square root.
63. QVSUB (A, I, B, J, C, K, N) Subtracts two vectors.
64. QVSWAP (A, I, C, K, N) Swaps two vectors.
65. QVTRAN (M, N, IAD, LV) Transposes a row-stored, M x N array of row vectors of length LV.

12.7.3 AP Routine Call Sequences

A note should be made about the conventions used in the description of the routines. Data addresses are normally denoted by A, B, C, or D and their increments (stride) by I, J, K, L and an element count by N. In the descriptions of the routines, many of the values in AP memory are referred to by the name given to the variable giving the address, e.g., A(mI) is used to denote the value in memory location A + m*I. All input variables are integer unless otherwise marked.

QGET

Transfer data from AP memory to host core.

```
    QGET (HOST, AP, N, TYPE)

  Inputs:
    AP      I     Target area in AP; 0-relative, increment=1
    N       I     Number of elements
    TYPE    I     Data type:
                     0   data is I in host
                     2   data is R in host
  Output:
    HOST(*) R/I Data array in "host"
```

QGSP

Read contents of SPAD register: FPS and Pseudo AP only.

```
      QGSP (I, NREG)

   Inputs:
      NREG  I  SPAD register number desired
   Outputs:
      I     I  Contents of the SPAD register.
```

QPUT

Transfer data from host memory to AP memory.

```
      QPUT (HOST, AP, N, TYPE)

   Inputs:
      AP      I     Target area in AP; 0-relative, increment=1.
      N       I     Number of elements
      TYPE    I     Data type:
                      0   data is I in host
                      2   data is R in host
      HOST(*) R/I Data array in "host"
```

QWAIT

Suspend host task until all AP I/O and computations are complete.

```
      QWAIT
```

QWD

Suspend host task until all AP I/O is complete.

```
      QWD
```

QWR

Suspend host task until all AP computations are complete.

```
      QWR
```

QINIT

Implements AIPS AP priority for true AP, increases the task priority for AIPS batch tasks using a true AP, and assigns an AP.

```
      QINIT (I1, I2, I3)

   Inputs:
      I1   I    Dummy
```

```
   I2   I    Dummy
Outputs:
   I3   I    AP number (Neg. to indicate virtual AP, i.e.,
             not to be rolled.
```

QRLSE

Releases the AP and lowers task priority for AIPS batch tasks using a true AP.

```
QRLSE
```

QBOXSU

Do a boxcar sum on a vector; values at the ends of the vector are the sum of the values within one boxcar length of the ends.

```
QBOXSU (A, I, NB, C, J, N)

Inputs:
   A          input vector base address
   I          input vector increment
   NB         boxcar width
   C          output vector base address; output vector
                  should not overlap input
   J          output increment
   N          number of elements
```

QCFFT

Do an in-place complex fast Fourier transform.

```
QCFFT (C, N, F)

Inputs:
   C      Base address (0-rel) of complex array to transform
   N      Number of points in array (must be power of two.)
   F      Transform direction; 1 -> Forward
                              -1 -> Backward
```

QCRVMU

Multiply the elements of a complex vector by the elements of a real vector.

```
C(mK)   = A(mI)   * B(mJ)            m = 0 to N-1
C(mK+1) = A(mI+1) * B(mJ)

QCRVMU (A, I, B, J, C, K, N)

Inputs:
   A       Source complex vector base address.
   I       Increment of A (normally 2 * integer)
   B       Source real vector base address
```

```
J       Increment of B
C       Destination vector base address
K       Increment of C (normally 2 * integer)
N       Element count
```

QCSQTR

Do an in-place transpose of square matrices of complex values.

```
   QCSQTR (CORNER, SIZE, ROW)

Inputs:
   CORNER    AP location of first corner of matrix encountered.
   SIZE      Size (number of reals) of a row or column.
   ROW       Number of locations in AP between beginnings
             of the rows.
```

QCVCMU

Multiply a scalar complex value times the complex conjugate of a vector, producing a real vector.

```
C(mJ) = B(0) * A(mI) + B(0+1) * A(mI+1)   for m = 0 to N-1

   QCVCMU (A, I, B, C, J, N)

Inputs:
   A     Source complex vector base address.
   I     Increment of A (normally 2 * integer)
   B     Address of scalar (real part)
   C     Destination real vector base address.
   J     Increment of C
   N     Element count (reals)
```

QCVCON

Take complex conjugate of a vector.

```
   C(mK)    =  A(mI)              for m = 0 to N-1
   C(mK+1)  = -A(mI+1)

   QCVCON (A, I, C, K, N)

Inputs:
   A       Source vector base address.
   I       Increment of A (normally 2 * integer)
   C       Destination vector base address
   K       Increment of C (normally 2 * integer)
   N       Element count
```

QCVEXP

Complex exponential of a vector.

```
C(mK)   = COS (A(mI))          for m = 0 to N-1
C(mK+1) = SIN (A(mI))

QCVEXP (A, I, C, K, N)
```

```
Inputs:
   A      Source vector base address.
   I      Increment of A
   C      Destination vector base address
   K      Increment of C (normally 2 * integer)
   N      Element count
```

QCVJAD

Add the elements of one complex vector to the complex conjugate of the elements of another complex vector.

```
C(mK)   = A(mI)   + B(mJ)          for m = 0 to N-1
C(mK+1) = A(mI+1) - B(mJ+1)

QCVJAD (A, I, B, J, C, K, N)
```

```
Inputs:
   A      Source vector base address.
   I      Increment of A (normally 2 * integer)
   B      Source vector base address (conjugate)
   J      Increment of B (normally 2 * integer)
   C      Destination vector base address
   K      Increment of C (normally 2 * integer)
   N      Element count
```

QCVMAG

Square the magnitude of the elements of a complex vector.

```
C(mK) = A(mI)**2 + A(mI+1)**2      for m = 0 to N-1

QCVMAG (A, I, C, K, N)
```

```
Inputs:
   A    Source vector base address
   I    A address increment (normally 2 * integer)
   C    Destination vector base address
   K    C address increment
   N    Element count
```

QCVMMA

Find the maximum of the square modulus of a complex vector.

```
    max (A(mI)**2 + A(mI+1)**2)      over m = 0 to N-1

    QCVMMA (A, I, C, N)

  Inputs:
    A     Source vector base address
    I     Increment of A (normally 2 * integer)
    C     Destination vector.
            0 = MAX(A ** 2) (real)
            1  = location of max (integer)
    N     Element count
  Also:
          SPAD(15) = index of max.
```

QCVMOV

Copy one complex vector to another.

```
    C(mK)   = A(mI)              for m = 0 to N-1
    C(mK+1) = A(mI+1)

    QCVMOV (A, I, C, K, N)

  Inputs:
    A     Source vector base address
    I     A address increment (normally 2 * integer, >= 2)
    C     Destination vector base address
    K     C address increment (normalIy 2 * integer)
    N     Element count
```

QCVMUL

Multiply the elements of two complex vectors.

```
    C(mK)    = A(mI) * B(mJ)   - F * A(mI+1) * B(mJ+1)
    C(mK+1)) = A(mI) * B(mj+1) + F * A(mI+1) * B(mJ)
                                       for  m = 0 to N-1

    QCVMUL (A, I, B, J, C, K, N, F)

  Inputs:
    A     Source vector base address
    I     A address increment (normally 2 * integer)
    B     Source vector base address
    J     B address increment (normally 2 * integer)
    C     Destination vector base address
    K     C address increment (normally 2 * integer)
    N     Element count
    F     Conjugate flag, 1 => normal complex multiply
                         -1 => multiply with conjugate of A
```

QCVSDI

Divide the elements of a complex vector with weights by a complex scalar. The complex vector is expected to have data in the order real, imaginary, weight. The weight is multiplied by the amplitude of the complex scalar. This is used for AIPS uv data.

```
C(mJ)   = (1./(B(1)**2+B(2)**2)) * (A(mI)  *B(1) + A(mI+1)*B(2))
C(mJ+1) = (1./(B(1)**2+B(2)**2)) * (A(mI+1)*B(1) - A(mI)  *B(2))
C(mJ+2) = A(mI+2) * SQRT(B(1)**2+B(2)**2)       for m = 0 to N-1

   QCVSDI (A, I, B, C, J, N)

Inputs:
   A      Source vector base address.
   I      Increment of A (normally 3)
   B      Source scalar address.
   C      Destination vector base address
   J      Increment of C (normally 3)
   N      Element count
```

QCVSMS

Subtract the elements of a real vector times the elements of a complex scalar from a complex vector, alternately i (SQRT(-1)) times the real vector times the complex scalar is subtracted from the complex vector. Since the element count is expected to be small, the looping is not very efficient.

```
If FLAG > 0
   D(mK)   = A(mI)   - B(1) * C(mJ)
   D(mK+1) = A(mI+1) - B(2) * C(mJ)     for m = 0 to N-1

If FLAG < 0
   D(mK)   = A(mI)   + B(2) * C(mJ)
   D(mK+1) = A(mI+1) - B(1) * C(mJ)     for m = 0 to N-1

   QCVSMS (A, I, B, C, J, D, K, N, FLAG)

Inputs:
   A      Source complex vector base address.
   I      Increment of A (normally 2 * integer)
   B      Source  complex scalar address.
   C      Source real vector base address
   J      Increment of C
   D      Destination complex vector base address
   K      Increment of D (normally 2 * integer)
   N      Element count
   FLAG   Flag, if < 0 multiply complex scalar by i
```

QDIRAD

Do a complex directed add: adds a complex vector to a complex vector whose addresses are given in the first vector.

```
B(A(mIA))   = B(A(mIA))   + A(mIA+1)   for m = 0 to N-1
B(A(mIA)+1) = B(A(mIA)+1) + A(mIA+2)
```

```
QDIRAD (A, IA, B, N)

Inputs:
   A   Source vector base address
         0 => address (integer) to be added to
              (address is zero relative)
         1,2 => complex value (reals)
   IA  Increment for A (normally 3)
   B   Destination vector base address
   N   Element count
```

QHIST

Compute the histogram of a vector: histogram element (NB-1)*(DATA-MIN)/(MAX-MIN), where DATA is the data value, is incremented.

```
QHIST (A, I, C, N, NB, AMAX, AMIN)

Inputs:
   A        Source vector base address.
   I        A address increment.
   C        Histogram base address
               Histogram must be cleared before first call.
   N        Element count for A
   NB       Number of bins in histogram
   AMAX     Address of histogram maximum.
   AMIN     Address of histogram minimum.
```

QLVGT

Logical vector greater than.

```
C(mK) = 1.0    if A(mI) >  B(mJ)
C(mK) = 0.0    if A(ml) =< B(mJ)   for m = 0 to N-1

QLVGT (A, I, B, J, C, K, N)

Inputs:
   A    Source vector base address
   I    A address increment
   B    Source vector base address
   J    B address increment
   C    Destination vector base address
   K    C address increment
   N    Element count
```

QMAXMI

Search the given vector for maximum and minimum values.

```
QMAXMI (A, I, MAX, MIN, N)
```

```
Inputs:
   A        Source vector base address
   I        Increment of A
   MAX      Location for maximum.
   MIN      Location for minimum.
   N        Element count.
```

QMAXV

Find maximum value of a vector and address of the maximum.

```
   QMAXV (A, I, C, N)

Inputs:
   A    Source vector base address
   I    A address increment
   C    Destination  base address
           C(0) = Max (A(mI))  m = 0 to N-1
           C(1) = address. also in SPAD 15.
   N    Element count
```

QMEMSZ

manipulates the size of the blank common used for the "AP" memory and vector work space. The size of the blank common is expanded to NWORDS, if that is larger than the current size, or is reduced to zero, if NWORDS is zero.
Note: this routine should be called only from Q routines. At present, it is only available for Cray XMPs under COS.

```
   QMEMSZ (NWORDS)

Inputs:
   NWORDS  I  The number of words desired in the blank common
Input from common /SPF/ (include D/CAPC.INC)
   MEMSIZ  I  The current size of the blank common.
              -1 => minimum size.
Output to common /SPF/
   MEMSIZ  I  The current size of the blank common.
              -1 => minimum size.
```

QMINV

Find minimum value of a vector and address of the minimum.

```
   QMINV (A, I, C, N)

Inputs:
   A    Source vector base address
   I    A address increment
   C    Destination  base address
           C(0) = Max (A(mI))  m = 0 to N-1
           C(1) = address. also in SPAD 15
```

```
N    Element count
```

QMTRAN

Transpose a matrix.

```
C((p+qMC)K) = A((q+pNC)I)
                  for  p = 0 to MC-1
                  and  q = 0 to NC-1

QMTRAN (A, I, C, K, MC, NC)

Inputs:
   A    Source matrix base address
   I    A address increment
   C    Destination  matrix base address
   K    C address increment
   MC   Number of columns of A
   NC   Numbers of rows of A
```

QPHSRO

Add a phase gradient to a complex array.

```
B(j) = A(j)*EXP(-i*(PHAS0+j*DELPHS))    for j = 0 to N-1

   or

B(mJ)   = A(mI) * cos(P0+mDP) - A(mI+1) * sin(P0+mDP)
B(mJ+1) = A(mI) * sin(P0+mDP) + A(mI+1) * cos(P0+mDP)
                              for m = 0 to N-1
     where cos(P0) = PHAS0(0),   sin(P0) = PHAS0(0+1)
           cos(DP) = DELPHS(0),  sin(DP) = DELPHS(0+1)

   QPHSRO (A, I, B, J, PHAS0, DELPHS, N)

Inputs:
   A         Source vector base address.
   I         Increment of A (normally 2 * integer)
   B         Destination base address.
   J         Increment of B (normally 2 * integer)
   PHAS0     Address of complex unit vector with
               phase PHAS0
   DELPHS    Address of complex unit vector with
               phase DELPHS
   N         Element count
```

QPOLAR

Rectangular to polar conversion.

```
C(mK)   = SQRT   (A(mI)**2 + A(mI+1)**2)
C(mK+1) = ARCTAN (A(mI+1) / A(mI))        for m = 0 to N-1

QPOLAR (A, I, C, K, N)
```

Inputs:

A	Source vector base address
I	A address increment (normally 2 * integer)
C	Destination vector base address
K	C address increment (normally 2 * integer)
N	Element count

QRECT

Polar to rectangular vector conversion:

```
COS (tab+fract) = COS(tab)*COS(fract) - SIN(tab)*SIN(fract)
SIN (tab+fract) = SIN(tab)*COS(fract) + COS(tab)*SIN(fract)

C(mK)   = A(mI) * COS (A(mI+1))
C(mK+1) = A(mI) * SIN (A(mI+1))    for m = 0 to N-1

QRECT (A, I, C, K, N)
```

Inputs:

A	Source vector base address
I	A address increment (normally 2 * integer)
C	Destination vector base address
K	C address increment (normally 2 * integer)
N	Element count

QRFFT

Does an in-place real-to-complex forward or complex-to-real inverse FFT.

```
QRFFT (C, N, F)
```

Inputs:

C	Base address of source and destination vector
N	Real element count (power of 2)
F	flag, 1=>forward FFT, -1=> reverse FFT.

QRFT

Compute a real, inverse fourier transform from arbitrarily, but uniformly, spaced data.

```
QRFT (UDATA, UFT, UPHO, NFT, NDATA)
```

Inputs:

UDATA	AP base address of input data.
UFT	AP base address of output F. T.
UPHO	AP base address of phase information for F. T. 0 = COS((TWOPI/(NG*NFT))*(1-ICENT)(1-BIAS))

```
                1 = SIN((TWOPI/(NG*NFT))*(1-ICENT)(1-BIAS))
                2 = COS((TWOPI/(NG*NFT))*(1-ICENT))
                3 = SIN((TWOPI/(NG*NFT))*(1-ICENT))
                4 = COS((TWOPI/(NG*NFT))*(1-BIAS))
                5 = SIN((TWOPI/(NG*NFT))*(1-BIAS))
                6 = COS((TWOPI/(NG*NFT)))
                7 = SIN((TWOPI/(NG*NFT)))
                ICENT = center pixel of grid
                BIAS = center of data array (1 rel)
                NG = No. tabulated points per cell.
    NFT     Number of FT points
    NDATA   Number of data points.
```

QSVE

Sum the elements of a vector

```
    C = SUM (A(mI))  m = 0 to N-1

    QSVE (A, I, C, N)

  Inputs:
    A       Source vector base address.
    I       Increment of A
    C       Destination scalar address
    N       Element count
```

QSVESQ

Sum the squares of the elements of a vector

```
    C = SUM (A(mI) * A(mI)) for m = 0 to N-1

    QSVESQ (A, I, C ,N)

  Inputs:
    A       Source vector base address.
    I       Increment of A
    C       Destination scalar address
    N       Element count
```

QVABS

Take the absolute value of the elements of a vector.

```
    C(mK) = ABS (A(mI))  for m = 0 to N-1

    QVABS (A, I, C, K, N)

  Inputs:
    A     Source vector base address
    I     A address increment
```

```
      C    Destination vector base address
      K    C address increment
      N    Element count
```

QVADD

Add the elements of two vectors.

```
      C(mK) = A(mI) + B(mJ)  for m = 0 to N-1

      QVADD (A, I, B, J, C, K, N)

   Inputs:
      A    First source vector base address
      I    A address increment
      B    Second source vector base address
      J    B address increment
      C    Destination vector base address
      K    C address increment
      N    Element count
```

QVCLIP

Limits the values in a vector to a specified range.

```
     D(mL) = B      if A(mI) <  B
           = A(mI)  if B     <= A(mI)  <  C
           = C      if C     <= A(mI)    for m = 0 to N-1

      QVCLIP (A, I, B, C, D, L, N)

   Inputs:
      A    Source vector base address
      I    A address increment
      B    Address of lower limit
      C    Address of upper limit
      D    Destination vector base address
      L    D address increment
      N    Element count
```

QVCLR

Fill a vector with zeroes.

```
      C(mK) = 0   for m = 0 to N-1

      QVCLR (C, K, N)

   Inputs:
      C     Destination vector base address
      K     C address increment
      N     Element count
```

QVCOS

Take the cosine of elements in a vector.

```
C(mK) = COS (A(mI))   for m = 0 to N-1

QVCOS (A, I, C, K, N)
```

Inputs:

A	Source vector base address
I	A address increment
C	Destination vector base address
K	C address increment
N	Element count

QVDIV

Divide the elements of two vectors

```
C(mK) = B(mJ) / A(mJ)   for m = 0 to N-1

QVDIV (A, I, B, J, C, K, N)
```

Inputs:

A	First source vector base address
I	A address increment
B	Second source vector base address
J	B address increment
C	Destination vector base address
K	C address increment
N	Element count

QVEXP

Exponentiate the elements of a vector.

```
C(mK) = EXP (A(mI))   for m = 0 to N-1

QVEXP (A, I, C, K, N)
```

Inputs:

A	Source vector base address
I	A address increment
C	Destination vector base address
K	C address increment
N	Element count

QVFILL

Fill a vector with a constant.

```
C(mK) = A   for m = 0 to N-1
```

```
QVFILL (A, C, K, N)

Inputs:
   A    Source scalar base address
   C    Destination vector base address
   K    C address increment
   N    Element count
```

QVFIX

Convert the elements of a vector from real to integer.

```
C(mK) = FIX (A(mI))   for m = 0 to N-1

QVFIX (A, I, C, K, N)

Inputs:
   A    Source vector base address
   I    A address increment
   C    Destination vector base address
   K    C address increment
   N    Element count
```

QVFLT

Convert the elements of a vector from integer to real.

```
C(mK) = FLOAT (A(mI))   for m = 0 to N-1

QVFLT (A, I, C, K, N)

Inputs:
   A    Source vector base address
   I    A address increment
   C    Destination vector base address
   K    C address increment
   N    Element count
```

QVIDIV

Divide the given vector by the product of two integers.

```
B(mJ) = A(mI) / (D1 * D2)     for m = 0,N-1

QVIDIV (A, I, D1, D2, B, J, N)

Inputs:
   A       Source vector base address.
   I       Increment for A
   D1      First dividend. Actual value, not an address.
   D2      Second dividend. Actual value, not an address.
   B       Destination vector base address.
   J       Increment for B
```

```
N       Element count.
```

QVLN

Take the natural logarithm of the elements of a vector.

```
C(mK) = LOGe (A(mI))    for m = 0 to N-1

QVLN (A, I, C, K, N)

Inputs:
   A      Source vector base address.
   I      Increment of A
   C      Destination vector base address
   K      Increment of C
   N      Element count
```

QVMA

Multiply two vectors and add a third.

```
D(mL) = (A(mI) * B(mJ)) + C(mK)   for m = 0 to N-1

QVMA (A, I, B, J, C, K, D, L, N)

Inputs:
   A    First source vector base address
   I    A address increment
   B    Second source vector base address
   J    B address increment
   C    Third source vector base address
   K    C address increment
   D    Destination vector base address
   L    D address increment
   N    Element count
```

QVMOV

Copy the elements of one vector to another.

```
C(mK) = A(mI)   for m = 0 to N-1

QVMOV (A, I, C, K, N)

Inputs:
   A      Source vector base address.
   I      Increment of A
   C      Destination vector base address
   K      Increment of C
   N      Element count
```

QVMUL

Multiply the elements of two vectors.

```
C(mK) = A(mJ) * B(mJ)   for m = 0 to N-1

QVMUL (A, I, B, J, C, K, N)
```

Inputs:

A	First source vector base address
I	A address increment
B	Second source vector base address
J	B address increment
C	Destination vector base address
K	C address increment
N	Element count

QVNEG

Take the negative of the elements of a vector.

```
C(mK) = - A(mI)   for m = 0 to N-1

QVNEG (A, I, C, K, N)
```

Inputs:

A	Source vector base address
I	A address increment
C	Destination vector base address
K	C address increment
N	Element count

QVRVRS

Reverse the elements in a vector.

```
C(mK) = C((N-m)K)  for m = 0 to N-1

QVRVRS (C, K, N)
```

Inputs:

C	Source and destination vector base address
K	C address increment
N	Element count

QVSADD

Add a scalar to the elements of a vector

```
C(mK) = B + A(mI)   for m = 0 to N-1

QVSADD (A, I, B, C, K, N)
```

```
Inputs:
   A    Source vector base address
   I    A address increment
   B    Adding scalar address
   C    Destination vector base address
   K    C address increment
   N    Element count
```

QVSIN

Take the sine of the elements of a vector.

```
C(mK) = SIN (A(mI))    for m = 0 to N-1  (A in radians)

QVSIN (A, I, C, K, N)

Inputs:
   A    Source vector base address
   I    A address increment
   C    Destination vector base address
   K    C address increment
   N    Element count
```

QVSMA

Multiply the elements of a vector by a scalar and add to the elements of another vector.

```
D(mL) = (A(mI) * B) + C(mK)    for m = 0 to N-1

QVSMA (A, I, B, C, K, D, L, N)

Inputs:
   A    First source vector base address
   I    A address increment
   B    Source scalar base address
   C    Second source vector base address
   K    C address increment
   D    Destination vector base address
   L    D address increment
   N    Element count
```

QVSMAFX

Multiply the elements of a vector by a scalar, add a scalar and round to an integer.

```
D(mL) = FIX (ROUND((A(mI)*B)+C))  for m = 0 to N-1

QVSMAFX (A, I, B, C, D, L, N)

Inputs:
   A    Source vector base address
   I    A address increment
   B    Multiplying scalar address
```

```
C    Adding scalar address
D    Destination vector base address
L    D address increment
N    Element count
```

QVSMSA

Multiply the elements of a vector by a scalar and add a second scalar.

```
D(mL) = (A(mI)*B)+C    for m = 0 to N-1

QVSMSA (A, I, B, C, D, L, N)

Inputs:
   A    Source vector base address
   I    A address increment
   B    Multiplying scalar address
   C    Adding scalar address
   D    Destination vector base address
   L    D address increment
   N    Element count
```

QVSMUL

Multiply the elements of a vector by a scalar.

```
C(mK) = A(mI) * B   for m = 0 to N-1

QVSMUL (A, I, B, C, K, N)

Inputs:
   A    Source vector base address
   I    A address increment
   B    Multiplying scalar address
   C    Destination vector base address
   K    C address increment
   N    Element count
```

QVSQ

Square the elements of a vector

```
C(mK) = A(mI)**2   for m = 0 to N-1

QVSQ (A, I, C, K, N)

Inputs:
   A    Source vector base address
   I    A address increment
   C    Destination vector base address
   K    C address increment
   N    Element count
```

QVSQRT

Take the square root of the elements of a vector.

```
C(mK) = SQRT (A(mI)) for m = 0 to N-1

QVSQRT (A, I, C, K, N)
```

Inputs:

A	Source vector base address
I	A address increment
C	Destination vector base address
K	C address increment
N	Element count

QVSUB

Subtract the elements of two vectors.

```
C(mK) = B(mJ) - A(mI)    for m = 0 to N-1

QVSUB (A, I, B, J, C, K, N)
```

Inputs:

A	First source vector base address
I	A address increment
B	Second source vector base address
J	B address increment
C	Destination vector base address
K	C address increment
N	Element count

QVSWAP

Swap the elements of a vector.

```
A(mI) = C(mK) and C(mK) = A(mI)   for m = 0 to N-1

QVSWAP (A, I, C, K, N)
```

Inputs:

A	First source/destination vector base address
I	A address increment
C	Second source/destination vector base address
K	C address increment
N	Element count

QVTRAN

Transpose a (row-stored) M X N array of row vectors of length LV. The starting address is given by IAD. The algorithm works in place. It is adapted from Boothroyd's CACM ALG.#302. Other, probably better, algorithms, are CACM #'S 380 and 467, but they're not as simple to program.

```
QVTRAN (M, N, IAD, LV)

Inputs:
   M     First dimension of the vector array
   N     Second dimension of the vector array
   IAD   Base address of the array
   LV    Length of the vectors.
```

Chapter 13

Tables in AIPS

13.1 Overview

This chapter is an attempt to describe the use of AIPS tables extension files and to describe the structure of these files. The next section describes general tables utility routines followed by routines which simplify the access to specific types of AIPS tables. The final section describes the structure of the tables files and the fundamental routines to access AIPS tables.

Table files consist of an extensive and rather flexible header and a table organized as rows and columns. An entry in a column may be either a single value or an array of values with the same data type. Each column has a specified format and is stored in the appropriate binary form for the local computer. The columns are ordered on disk in an order appropriate to computer addressing, but are accessed in any desired logical column order via a lookup list.

The extension file contains not only the rows and columns, but also a variety of other information. Each column has an associated 24-character column "title" and an 8-character "units" field. Each row has a "selection" flag which allows the user to access temporarily a subset of his table. The strings used to specify the current selection are stored in the file for display. The file may also contain general information applying to the full table in the form of keyword/value pairs. This information will be called the table "header" data.

13.2 General Tables Routines

There are a number of utility routines which perform operations on AIPS tables. Hopefully there will be many more of these as the use of tables in AIPS increases. The following list gives a short description of these routines; details of the call sequences are given at the end of this chapter. Also of interest to the programmer is the AIPS task PRTAB for printing the contents of a table file.

1. ALLTAB copies all tables, allows a list of exclusions.
2. ISTAB determines if an extension file type exists and if it is a table.
3. TABCOP copies the entire contents of one or more tables of a given type.
4. TABKEY reads or writes keyword/value pairs to a table header.
5. TABMRG merges or adds adjacent, like rows in a table.
6. TABSRT sorts the rows in a table file using up to 4 keys.

13.3 Specific Tables Routines

Because of the generality of the tables routines, the low-level use of tables is rather cumbersome. For this reason, there are a number of specialized routines which simplify the access to a given type of table. In

general, these routines come in pairs - one to create/initialize the I/O and the other to read or write to the file. If there are keyword/value pairs associated with a given table type, they are processed by the initialization routine. These specialized routines usually return the contents of a row into properly named variables, which avoids the use of equivalencing in the calling routine.

Since many of the tables in AIPS are subject to modification, especially those associated with the calibration package; the number of columns in each table is likely to change. For this reason it is strongly recommended that the number of columns in these tables be taken from the PARAMETER include PUVD.INC especially to be used for declaring array sizes. Comments in the text of this include file, which is shown at the end of this chapter, tell which PARAMETER belongs to which table type.

The specific tables routines are briefly described in the following list; details of the call sequences are given at the end of this chapter.

1. ANTINI and TABAN access AN (Antenna) tables.
2. BLINI and TABBL access BL (baseline dependent calibration) tables.
3. BPINI and TABBP access BP (bandpass) tables.
4. CALINI and TABCAL access calibration (CL) tables.
5. CCINI creates/initializes CC (CLEAN component or gaussian model files).
6. CCMERG sums CC components at the same position including the two needed sorts.
7. CHNCOP copies selected portions of a Frequency (FQ) table.
8. CHNDAT reads/writes/creates the contents of FQ (Frequency descriptor) tables.
9. FLGINI and TABFLG access FG (Flag) tables.
10. FQINI and TABFQ access FQ (FreQuency) tables.
11. GETNAN determines the number of subarrays and the numbers of antennas in each from the AN table(s).
12. NDXINI and TABNDX access NX (Index) tables.
13. SNINI and TABSN access solution (SN) tables.
14. SOUINI and TABSOU access SU (Source) tables.

13.4 The Format Details

There are several distinct types of information kept in a table file. Most important is the data tabulated, referred to as "row data". Associated with each column is label information; this includes a label, units, element count, and data type. There is also a provision for storing general information about the file in the form of keyword/value pairs. A keyword/value pair consists of a string of characters (Keyword) which gives a label to a value (Value) which may be any of a number of data types.

13.4.1 Row Data

The row data are stored as an integer number of rows per disk record (256 integers) or as an integer number of disk records per row. The columns are given a physical order appropriate to addressing on all computers. The logical order is carried in the file header record (physical record 1, see below) and in a set of array indices for addressing by the programs. The type of data is specified by code numbers. These codes and the physical ordering are as follows:

```
      ORDER                    ARRAY   BASIC CODE  +  LENGTH
   double-precision floating    D          1       +  10 * LENGTH
   single-precision floating    R          2       +  10 * LENGTH
   hollerith (4 char / word)    H          3       +  10 * LENGTH
   integer                      I          4       +  10 * LENGTH
   logical                      L          5       +  10 * LENGTH
   bit (NBITWD / integer)       I          7       +  10 * LENGTH
   select flag                  I          9           -

Declarations:
      INTEGER   I(*)
      LOGICAL   L(*)
      HOLLERITH H(*)
      REAL      R(*)
      DOUBLE PRECISION D(*)
      EQUIVALENCE (I, L, H, R, D)
```

13.4.2 Physical File Format

The data, control, and header information are written in the Table file via ZFIO in 512-byte (256-integer) blocks. The order on disk, by physical record number, is:

```
record   1      : Control information and lookup table (see later)
         2      : DATPTR(128) subscript of the appropriate array for
                        logical column n
                  DATYPE(128) type code for logical column n
         3 - 4  : Selection strings now in force
         5 - m  : Titles (24 HOLLERITH) in physical column order
       m+1 - i  : Units (8 HOLLERITH) in physical column order
       i+1 - k  : Table header (keyword/value pairs, see below)
       k+1 - *  : Row data in n rows/record or n records/row
where
       m =     5 +  (NCOL-1) / (256 / 6)
       i = m + 1 +  (NCOL-1) / (256 / 2)
       k = i + 1 +  (NKEY-1) / (256 / 5)
       NCOL = number logical columns not including the select column
       NKEY = maximum number of keyword/value pairs
```

13.4.3 Control Information

Physical record one contains file control data needed to do the I/O operations and maintain the physical file. It is prepared by subroutine TABINI and modified by TABIO. The latter subroutine returns the record to disk on OPCODE = 'CLOS'. Its contents are:

```
 1          Number 512-byte records now in file
 2
 3          Max number of rows allowed in current file
 4
 5          Number of rows (logical records) now in file
 6
 7
 8          # values/logical (# words/row incl. selection flag)
 9          > 0 => number rows / physical record
```

```
            < 0 => number physical records / row
 10         Number logical columns/row (not including selection
            column)
 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 write access date: ZDATE(33), ZTIME(36)
 39 - 40  H Last write access task name
 41
 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  H "*AIPS TABLE*" string to verify that this is a table
 57 - 59
 60         If 1 then the table CANNOT be written as FITS ASCII
 61         Number of selection strings now in file
 62         Next available address for a selection string
 63         First address of selection string 1
 64         First address of selection string 2
 65         First address of selection string 3
 66         First address of selection string 4
 67         First address of selection string 5
 68         First address of selection string 6
 69         First address of selection string 7
 70         First address of selection string 8
********** for TABIO / TABINI use only **********
 71         IOP : 1 => read, 2 => write
 72         Number words per logical record (incl. select)
 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 (0 - 5)
 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         File logical unit number (LUN)
 82         FTAB pointer for open file (IND)
**********
 83 -100       Reserved
**********
101 -128  H Table title.
```

```
129 -256     lookup table as COLPTR (logical column) = physical column
```

13.4.4 Keyword/value records

The keyword/value pairs are stored in 5 single-precision floating locations, 256 / 5 per physical record. The keyword is an 8-character string stored as a HOLLERITH. It is left justified. The value is stored left justified in the 3rd and 4th reals using as many words as needed (see table below). The keyword type is given in a integer following the 4th word.

The keyword data type codes which specify the type of the binary value are:

```
1    double-precision floating point
2    single-precision floating point
3    8-character HOLLERITH string  (4 chars / element)
4    integer
5    logical
```

In the call sequence to TABIO, the variable RECORD is an integer array used to convey the data to the I/O operations. For keyword/value pairs, RECORD is divided as follows:

```
RECORD(1)  1st 4 chars of the keyword
RECORD(2)  2nd 4 chars of keyword
RECORD(3)  value
RECORD(5)  type code

where the value occupies the following number of integer words:
         type 1    NWDPDP
              2    1
              3    2
              4    1
              5    1
```

13.4.5 I/O buffers

The call to TABINI specifies two buffers, one for I/O scratch and control and the other for the data pointers which will be used by the calling program to access the column data. The first, called BUFFER, is used as

```
BUFFER(  1)-BUFFER(128) control pointers
BUFFER(129)-BUFFER(256) lookup table
BUFFER(257)-BUFFER(***) current physical record(s) of table data
                        where *** = 512 if there are >= 1 rows/rec,
                        *** = (n+1)*256 if there are n recs/row.
```

The call sequence of TABINI has an argument, NBUF, which gives the length of BUFFER. This is used solely to check that BUFFER is large enough to handle the present table file. BUFFER is also provided by the programmer to TABIO, which will modify the control and data portions. The programmer should not modify BUFFER between the call to TABINI and the call to TABIO with OPCODE 'CLOS', except to insert a title for the table in words 101 - 128, or to correct the sort order information.

The second buffer, called DATP, is used by the non-I/O portions of the table package. DATP(1,1) - DATP(128,1) contains the subscript of the appropriate array for the logical columns. DATP(1,2) - DATP(128,2) contains the data type / element count for each logical column. When TABINI is to create the table extension file, the programmer must fill in DATP(1,2) - DATP(NCOL,2) before calling TABINI. A complete set of DATP will be returned by TABINI under all circumstances.

13.4.6 Fundamental Table Access Subroutines

There is a set of basic table handling routines which apply to all tables files. The following list gives a short description; the details of the call sequences and usage are found at the end of this chapter.

1. TABINI creates/opens/catalogs an AIPS table.
2. TABIO does I/O to a tables file. Row data, keyword/value pairs and control information are passed through this subroutine.
3. GETCOL returns the value and value type at a specified row and column from an open table.
4. FNDCOL locates the logical column number for a column with a specified label.
5. PUTCOL puts a specified the value into a specified row and column from an open table.

13.4.7 Table Reformating Subroutines

Several of the tables used with the calibration package are subject to change as new things are added. In order to minimize the confusion there are a number of routines which check if the table uses the current format and if not reformats the table adding any new columns. A short description of these routines is given here and a more complete description is given at the end of this chapter.

1. BLREFM reformats a BL (BaseLine dependent calibration) table.
2. BPREFM reformats a BP (BandPass calibration) table.
3. CLREFM reformats a CL (CaLibration) table.
4. SNREFM reformats a SN (calibration SolutioN) table.

13.5 Includes

13.5.1 PUVD.INC

A number of generally useful parameters including the number of columns in each of the calibration tables is given in this include as PARAMETERS.

```
C                                                        Include PUVD
C                                       Parameters for uv data
      INTEGER   MAXANT,  MXBASE,  MAXIF, MAXFLG, MAXFLD, MAXCHA
C                                       MAXANT = Max. no. antennas.
      PARAMETER (MAXANT=45)
C                                       MXBASE = max. no. baselines
      PARAMETER (MXBASE= ((MAXANT*(MAXANT+1))/2))
C                                       MAXIF=max. no. IFs.
      PARAMETER (MAXIF=15)
C                                       MAXFLG= max. no. flags active
      PARAMETER (MAXFLG=1000)
C                                       MAXFLD=max. no fields
      PARAMETER (MAXFLD=16)
C                                       MAXCHA=max. no. freq. channels.
      PARAMETER (MAXCHA=512)
C                                       Parameters for tables
      INTEGER MAXCLC, MAXSNC, MAXANC, MAXFGC, MAXNXC, MAXSUC,
     *   MAXBPC, MAXBLC, MAXFQC
C                                       MAXCLC=max no. cols in CL table
      PARAMETER (MAXCLC=41)
```

```
C                                       MAXSNC=max no. cols in SN table
      PARAMETER (MAXSNC=20)
C                                       MAXANC=max no. cols in AN table
      PARAMETER (MAXANC=12)
C                                       MAXFGC=max no. cols in FG table
      PARAMETER (MAXFGC=8)
C                                       MAXNXC=max no. cols in NX table
      PARAMETER (MAXNXC=7)
C                                       MAXSUC=max no. cols in SU table
      PARAMETER (MAXSUC=21)
C                                       MAXBPC=max no. cols in BP table
      PARAMETER (MAXBPC=14)
C                                       MAXBLC=max no. cols in BL table
      PARAMETER (MAXBLC=14)
C                                       MAXFQC=max no. cols in FQ table
      PARAMETER (MAXFQC=5)
C                                                   End PUVD.
```

13.6 Routines

Following are the descriptions of the call sequences and usage notes for the routines discussed in this chapter.

13.6.1 Routines Applying to All Tables

ALLTAB

Copies all Table extension file(s). The output files must be new — old ones cannot be rewritten. The output file must be opened WRIT in the catalog and will have its CATBLK updated on disk.

```
ALLTAB (NONOT, NOTTYP, LUNOLD, LUNNEW, VOLOLD, VOLNEW, CNOOLD,
  *   CNONEW, CATNEW, BUFF1, BUFF2, IRET)
Inputs:
   NONOT      I     Number of "Forbidden" types to copy.
   NOTTYP(*)  C*2   Table types to ignore (2 char meaningful, blank
                    filled)
   LUNOLD     I     LUN for old file
   LUNNEW     I     LUN for new file
   VOLOLD     I     Disk number for old file.
   VOLNEW     I     Disk number for new file.
   CNOOLD     I     Catalog slot number for old file
   CNONEW     I     Catalog slot number for new file
In/out:
   CATNEW(256)I     Catalog header for new file.
Output:
   BUFF1(1024) I     Work buffer
   BUFF2(1024) I     Work buffer
   IRET        I     Return error code  0 => ok, otherwise TABCOP
                                         or 10*CATIO error.
```

FNDCOL

Is used with AIPS Table extension files. It locates the logical column number(s) which are titled with specified strings.

```
FNDCOL (NKEY, KEYS, LKEY, LORDER, BUFFER, KOLS, IERR)
Inputs:
   NKEY    I            Number columns to be found
   KEYS    C*24(NKEY)   Column titles to locate
   LKEY    I            Number characters to check in each of KEYS
                        (legal values 1 through 24)
   LORDER  L            T => logical order desired, else phys.
In/out:
   BUFFER  I(>512)      TABINI/TABIO buffer/ header/ work area
Output:
   KOLS    I(NKEY)      Logical column numbers: 0 => none,
                              -1 => more than one (!)
   IERR    I            Error code: 0 => ok, 1 - 10 from ZFIO
                             >10 = 10 + # of failed columns
```

GETCOL

Returns the value and value type found in an open table file at the specified logical column and row.

```
GETCOL (IRNO, ICOL, DATP, BUFFER, RTYPE, RESULT, SCRTCH, IERR)
Inputs:
   IRNO    I         Table row number: n.b. I
   ICOL    I         Table column number
   DATP    I(256)    Pointer array returned by TABINI
In/out:
   BUFFER  I(*)      Control area set up by TABINI, used in TABIO
Output:
   RTYPE   I         10 * length + Type of column:
                     1 -> D,  2 -> R,  4 -> I,  5 -> L*?,  6 -> I.
                     Characters hollerith, bits packed.
   RESULT  ?(*)      Value of column: use D, R, H, I equivalenced
                     arrays, note: may be an array; if length = 0,
                     then no value and RESULT is unchanged.
   SCRTCH  I(*)      Scratch large enough to hold a row
   IERR    I         Error code: 0 => OK.
                       -1 => OK, but row is flagged
                        1 file not open,  2 input error
                        3 I/O error,      4 read past EOF
                        5 bad data type
```

ISTAB

Given an extension type, volume, version, determine if the extension file exists, and if so, is it a standard table.

```
ISTAB (ETYPE, IVOL, ISLOT, IVER, LUN, BUFFER, TABLE, EXIST, FITASC,
  *   IERR)
Inputs:
   ETYPE  C*2  Extension type like 'CC'.
   IVOL   I    Disk Volume.
   ISLOT  I    Map catalog slot number.
   LUN    I    AIPS LUN to use for opening and reading a test block.
In/Out:
   TABLE  L    True if this extension type is a table or this
```

```
                version does not exist.
     EXIST  L   True if the extension file exists, else false.
     FITASC L   True if the file can be written as a FITS ASCII table.
  Output:
     IERR   I   Error code, 0=>OK else failed
```

PUTCOL

Enters the value and value type into an open table file at the specified logical column and row.

```
PUTCOL (IRNO, ICOL, DATP, BUFFER, VALUE, SCRTCH, IERR)
Inputs:
   IRNO    I         Table row number
   ICOL    I         Table column number
   DATP    I(256)    Pointer array returned by TABINI
   VALUE   ?(*)      Value of column: use D, R, H, I equivalenced
                     arrays, note: may be an array; if length = 0,
                     then no value and VALUE is unchanged.
In/out:
   BUFFER  I(*)      Control area set up by TABINI, used in TABIO
Output:
   SCRTCH  I(*)      Scratch large enough to hold a row
   IERR    I         Error code: 0 => OK.
                       -1 => OK, but row is flagged
                        1 file not open,  2 input error
                        3 I/O error,      4 read past EOF
                        5 bad data type
```

TABCOP

Copies Table extension file(s). The output file must be a new extension - old ones cannot be rewritten. The output file must be opened WRIT in the catalog and will have its CATBLK updated on disk.

```
TABCOP (TYPE, INVER, OUTVER, LUNOLD, LUNNEW, VOLOLD, VOLNEW,
  *   CNOOLD, CNONEW, CATNEW, BUFF1, BUFF2, IRET)
Inputs:
   TYPE     C*2     Extension file type (e.g. 'CC','AN')
   INVER    I       Version number to copy, 0 => copy all.
   OUTVER   I       Version number on output file, if more than one
                    copied (INVER=0) this will be the number of the
                    first file.  If OUTVER = 0, it will be taken as
                    1 higher than the previous highest version.
   LUNOLD   I       LUN for old file
   LUNNEW   I       LUN for new file
   VOLOLD   I       Disk number for old file.
   VOLNEW   I       Disk number for new file.
   CNOOLD   I       Catalog slot number for old file
   CNONEW   I       Catalog slot number for new file
In/out:
   CATNEW   I(256)  Catalog header for new file.
Output:
   BUFF1    I(256)  Work buffer
   BUFF2    I(256)  Work buffer
   IRET     I       Return error code  0 => ok
```

```
                    1 => files the same, no copy.
                    2 => no input files exist
                    3 => failed
                    4 => no output files created.
                    5 => failed to update CATNEW
                    6 => output file exists
```

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     I          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.
```

TABIO

Does random access I/O to Tables extension files. Mixed reads and writes are allowed if TABINI was called 'WRIT'. Writes are limited by the size of the structure (i.e. no. columns for units and titles) or to the current maximum logical record plus one. 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.
```

TABKEY

Reads or writes KEYWORDs from or to an AIPS table file header. The order of the keywords is arbitrary. Table file must have been previously opened with TABINI.

```
TABKEY (OPCODE, KEYWRD, NUMKEY, BUFFER, LOCS, VALUES, KEYTYP, IERR)
Inputs:
   OPCODE   C*4        Operation desired, 'READ', 'WRIT',
                          'ALL ' => Read all.
   KEYWRD   C(*)*8     Keywords to read/write
```

```
   BUFFER   I(*)        Buffer being use for table I/O >= 512 words.
In/out:
   NUMKEY   I           Number of keywords to read/write.
                           Input on OPCODE='ALL' = max. to read.
                           Output on OPCODE='ALL' = no. read.
   LOCS     I(NUMKEY)   The word offset of first integer
                        word of keyword value in array VALUES.
                        Output on READ, input on WRIT.
                        On READ this value will be -1 for keywords
                        not found.
  VALUES    I           The array of keyword values; due to word
                        alignment problems on some machines values
                        longer than a integer should be copied,
                        eg. if the  5th keyword (XXX) is a R:
                             IPOINT = LOCS(5)
                             CALL COPY (1, VALUES(IPOINT), XXX)
                        Output on READ, input on WRIT
  KEYTYP    I(NUMKEY)   The type code of the keywords:
                           1 = Double precision floating
                           2 = Single precision floating
                           3 = Character string (8 HOLLERITH chars)
                           4 = integer
                           5 = Logical
Output:
   IERR     I           Return code, 0=>OK,
                           1-10 =>TABIO error
                           19   => unrecognized data type.
                           20   => bad OPCODE
                           20+n => n keywords not found on READ.
```

TABMRG

Merges an AIPS general table-format file. It does not sort the file - so sorts must be done first if merging is to make any sense. TABMRG compares row N with row N+1 and merges the two, summing columns SKOL and using row N for the others, if each of columns EKOL are within TKOL of each other.

```
TABMRG (DISK, CNO, TYPE, INVER, OUTVER, EKOL, SKOL, TKOL, ITABL,
  *   INBUF, OTABL, ONBUF, CATBLK, OUTNUM, IERR)
Inputs:
   DISK     I          Disk number to use
   CNO      I          Primary file catalog number
   TYPE     C*2        Extension file type (e.g. CC)
   INVER    I          Input extension file number (0 -> high)
   EKOL     I(*)       Columns which must be "equal" by logical
                       column number.  A zero MUST terminate the
                       list unless it is 25 long.  Must have >0.
   SKOL     I(*)       Columns which will be summed on a merge by
                       column number.  A zero MUST terminate the
                       list unless it is 25 long.  May be 0.
   TKOL     R(*)       Tolerance for equality by logical column
                       number (parallel to EKOL list)
   INBUF    I          Number I   words in ITABL
   ONBUF    I          Number I   words in OTABL
In/out:
```

```
      OUTVER  I          Output extension file number (0 -> high+1)
      CATBLK  I(256)     Primary file header
   Output:
      ITABL   I(>512)    Scratch for input table IO
      OTABL   I(>512)    Scratch for output table IO
      OUTNUM  I          Number rows in output file.
      IERR    I          Error code: 0 -> ok
                            1 -> input file open error
                            2 -> input parameter error
                            3 -> output file create/open error
                            4 -> ZFIO IO error
                            5 -> TABIO IO error
```

TABSRT

Subroutine to sort an AIPS table extension file. First key changes the most slowly. A linear combination of two columns or a substring of a bit or character string may be used. The columns and factors are specified in KEY and FKEY, the first (slower varying key) is:

```
KEY_VALUE1 = COL_VALUE(KEY(1,1)) * FKEY(1,1) +
             COL_VALUE(KEY(2,1)) * FKEY(2,1)
```

The faster changing key value is:

```
KEY_VALUE2 = COL_VALUE(KEY(1,2)) * FKEY(1,2) +
             COL_VALUE(KEY(2,2)) * FKEY(2,2)
```

In the case of bit or character strings only one column is used to generate the key values.

```
A KEY(m,n) < 0 => use ABS (COL_VALUE(-KEY(m,n))).

   TABSRT (DISK, CNO, TYPE, INVER, OUTVER, KEY, FKEY, BUFFER, BUFSZ,
     *   TABUFF, NBUF, CATBLK, IERR)
   Inputs:
      DISK         I    Disk number of the file.
      CNO          I    Catalog slot number.
      TYPE         I    Two character type code (e.g. 'CC')
      INVER        I    Input version number
      OUTVER       I    Output version number
      KEY(2,2)     I    Sort keys: may be linear combination of two
                        numeric value columns.  KEY contains the column
                        numbers and FKEY contains the factors.  If the
                        column is a string (bit or char.) then
                        FKEY(1,n)=first char/bit and FKEY(2,n)=number
                        of char/bit and KEY(2,n) is ignored.
                        KEY(2,n)=0 => ignore, <0 => use abs. value.
                        Column no. is the logical number.
      FKEY(2,2)    R    Key  coefficients, 0=>1, see above.
      BUFSZ        I    Size of BUFFER in bytes.
      NBUF         I    Size of TABUFF in (I)   words.
      CATBLK(256)  I    Catalog header record.
   Output:
      TABUFF(*)    I    Buffer large enough to handle I/O to table.
      BUFFER(*)    R    I/O work buffer
      IERR         I    Error code, 0 => OK, else error.
```

```
                    10 => Couldn't find or open file.
Useage Notes:
     Normally the keys are sorted into ascending order, to sort into
descending order negate the values of FKEYn.
     TWO standard scratch files will be created and entered into the
/CFILES/ common.  These scratch files will be deleted on normal
termination.  Include DFIL.INC should be included in the main
routine and a call made to DIE rather than DIETSK should
be made at the end of the program execution.  The values in BADD
(adverb BADDISK) in the /CFILES/ common should be initialized.
     IF a disk based sort is required, then a 4-way merge sort will
be used.
     Since keys are converted into floating point numbers some
accuracy may be lost sorting on character or bit strings.
     For a 1 key sort use KEY2(1) = 0.
NB: This routine will modify the contents of common /MAPHDR/
(DCAT.INC)
```

13.6.2 Routines Applying to Specific Tables

ANTINI

Creates and initializes antenna (AN) tables.

```
ANTINI (OPCODE, BUFFER, DISK, CNO, VER, CATBLK, LUN,
  *   IANRNO, ANKOLS, ANNUMV, ARRAYC, GSTIA0, DEGPDY, SAFREQ, RDATE,
  *   POLRXY, UT1UTC, IATUTC, ANAME, NUMORB, NOPCAL, IERR)
Inputs:
   OPCODE          C*4  Operation code:
                        'WRIT' = create/init for write or read
                        'READ' = open for read only
   BUFFER(512)     I    I/O buffer and related storage, also defines
                        file if open.
   DISK            I    Disk to use.
   CNO             I    Catalog slot number
   VER             I    GA file version
   CATBLK(256)     I    Catalog header block.
   LUN             I    Logical unit number to use
Input/output (file keywords):
   ARRAYC(3)       D    Array center X coord. (meters, earth center)
   GSTIA0          D    GST at IAT=0 (degrees) on ref. date
   DEGPDY          D    Earth rotation rate (deg/IAT day)
   SAFREQ          D    Obs. Reference Frequency for subarray(Hz)
   RDATE           C*8  Reference date as 'DD/MM/YY'
   POLRXY(2)       R    Polar position X,Y (meters) on ref. date
   UT1UTC          R    UT1-UTC  (time sec.)          "
   IATUTC          R    IAT-UTC  (time sec.)          "
   ANAME           C*8  Array name
   NUMORB          I    Number of orbital parameters
   NOPCAL          I    Number of polarization calibration constants.
   ANNUMV(MAXANC)  I    Element count in each column. On input only
                        used if the file is created.
Output:
   IANRNO          I    Next scan number, start of the file if READ,
                        the last+1 if WRITE
```

```
   ANKOLS(MAXANC)  I    The column pointer array in order, ANNAME,
                        STABXYZ, ORBPARM, NOSTA, MNTSTA, STAXOF,
                        POLTYA, POLAA, POLCALA, POLTYB, POLAB, POLCALB
   IERR            I    Return error code, 0=>OK, else TABINI or TABIO
                        error.
Useage NOTE: use the include 'DANT.INC' for the declarations in
ANTINI and TABAN.
```

BLINI

Creates and initializes baseline correction (BL) extension tables.

```
BLINI (OPCODE, BUFFER, DISK, CNO, VER, CATBLK, LUN,
  *   IBLRNO, BLKOLS, BLNUMV, NUMANT, NUMPOL, NUMIF, IERR)
 Inputs:
  OPCODE       C*4 Operation code:
                   'WRIT' = create/init for write or read
                   'READ' = open for read only
  BUFFER(1024) I   I/O buffer and related storage, also defines file
                   if open.
  DISK         I   Disk to use.
  CNO          I   Catalog slot number
  VER          I   BL file version
  CATBLK(256)  I   Catalog header block.
  LUN          I   Logical unit number to use
 Input/output
  NUMANT       I    Number of antennas
  NUMPOL       I    Number of polarizations.
  NUMIF        I    Number of IFs
 Output:
  IBLRNO       I   Next scan number, start of the file if 'READ',
                   the last+1 if WRITE
  BLKOLS(MAXBLC) I   The column pointer array in order, TIME, SOURID,
                   SUBARRAY, ANTENNA1, ANTENNA2, FREQ. ID,
                   REALM1, IMAGM1, REALA1, IMAGA1,
                   Following used if 2 polarizations per IF
                   REALM2, IMAGM2, REALA2, IMAGA2.
  BLNUMV(MAXBLC) I   Element count in each column.
  IERR         I   Return error code, 0=>OK, else TABINI or TABIO
                   error.
```

BLREFM

Routine to change the format of the BL table from an old format to the current one if necessary. NOTE: routine uses LUN 45 as a temporary logical unit number.

```
BLREFM (DISK, CNO, VER, CATBLK, LUN, IRET)
Inputs:
   DISK            I       Volume number
   CNO             I       Catalogue number
   VER             I       Version to check/modify
   CATBLK(256)     I       Catalogue header
   LUN             I       LUN to use
Output:
```

```
   IRET            I       Error, 0 => OK
Note, routine will leave no trace of its operation, i.e. BL table
will be closed on output and will have same number as one specified.
Difference will be only that number of columns has changed if that
is required.
```

BPINI

Creates and initializes bandpass (BP) extension tables.

```
BPINI (OPCODE, BUFFER, DISK, CNO, VER, CATBLK, LUN,
  *   IBPRNO, BPKOLS, BPNUMV, NUMANT, NUMPOL, NUMIF, NUMFRQ,
  *   BCHAN, IERR)
Inputs:
  OPCODE       C*4 Operation code:
                   'WRIT' = create/init for write or read
                   'READ' = open for read only
  BUFFER(4096) I   I/O buffer and related storage, also defines file
                   if open.
  DISK         I   Disk to use.
  CNO          I   Catalog slot number
  VER          I   BL file version
  CATBLK(256)  I   Catalog header block.
  LUN          I   Logical unit number to use
 Input/output
  NUMANT       I    Number of antennas
  NUMPOL       I    Number of polarizations.
  NUMIF        I    Number of IFs
  NUMFRQ       I    Number of frequency channels
  BCHAN        I    Start channel number
Output:
  IBPRNO       I   Next scan number, start of the file if 'READ',
                   the last+1 if WRITE
  BPKOLS(MAXBPC) I   The column pointer array in order:
                     TIME, INTERVAL, SOURID,
                     SUBARRAY, ANTENNA,
                     BANDW (of individual channel), IFFREQ,
                     FREQ. ID,
                     REFANT1, REAL1, IMAG1,
                     Following used if 2 polarizations per IF
                     REFANT2, REAL2, IMAG2.
  BPNUMV(MAXBPC) I   Element count in each column.
  IERR         I   Return error code, 0=>OK, else TABINI or TABIO
                   error.
```

BPREFM

Routine to change the format of the BP table from an old format to the new one if necessary.
NOTE: routine uses LUN 45 as a temporary logical unit number.

```
BPREFM (DISK, CNO, VER, CATBLK, LUN, IRET)
Inputs:
   DISK            I       Volume number
   CNO             I       Catalogue number
```

```
   VER             I       Version to check/modify
   CATBLK(256)     I       Catalogue header
   LUN             I       LUN to use
Output:
   IRET            I       Error, 0 => OK

Note, routine will leave no trace of its operation, i.e. BP table
will be closed on output and will have same number as one specified.
Difference will be only that number of columns has changed if that
is required.
```

CALINI

Creates and initializes calibration (CL) extension tables.

```
CALINI (OPCODE, BUFFER, DISK, CNO, VER, CATBLK, LUN,
  *   ICLRNO, CLKOLS, CLNUMV, NUMANT, NUMPOL, NUMIF, GMMOD, IERR)
Inputs:
  OPCODE       C*4 Operation code:
                   'WRIT' = create/init for write or read
                   'READ' = open for read only
  BUFFER(512)  I   I/O buffer and related storage, also defines file
                   if open.
  DISK         I   Disk to use.
  CNO          I   Catalog slot number
  VER          I   CL file version
  CATBLK(256)  I   Catalog header block.
  LUN          I   Logical unit number to use
 Input/output
  NUMANT       I    Number of antennas
  NUMPOL       I    Number of IFs per pair
  NUMIF        I    Number of IF pairs
  GMMOD        R    Mean gain modulus
 Output:
  ICLRNO       I   Next scan number, start of the file if 'READ',
                   the last+1 if WRITE
  CLKOLS(MAXCLC) I   The column pointer array in order, TIME,
                     TIME INT., SOURCE ID., ANTENNA NO., SUBARRAY,
                     FREQID, ROT.MEAS. GEODELAY, GEOPHASE, GEORATE,
                     DOPOFF, CLKGD 1, DCLKGD 1, CLKPD 1, DCLKPD 1,
                     ATMGD 1, DATMGD 1, ATMPD 1, DATPGD 1,
                     REAL1, IMAG1, RATE 1, DELAY 1, TSYS1, WEIGHT1,
                     REFANT 1
                     Following used if 2 polarizations per IF
                     CLKGD 2, DCLKGD 2, CLKPD 2, DCLKPD 2,
                     ATMGD 2, DATMGD 2, ATMPD 2, DATPGD 2,
                     REAL2, IMAG2, RATE 2, DELAY 2, TSYS2, WEIGHT2,
                     REFANT 1
  CLNUMV(MAXCLC) I   Element count in each column.
  IERR         I   Return error code, 0=>OK, else TABINI or TABIO
                   error.
```

CCINI

Creates and/or opens for writing (and reading) a specified CC (components table) file.

```
CCINI (LUN, NCOL, VOL, CNO, VER, CATBLK, BUF, IERR)
Inputs:
   LUN     I         Logical unit number to use
   VOL     I         Disk number
   CNO     I         Catalog number
In/out:
   NCOL    I         Number of columns: 3 or 7 are allowed.
   VER     I         Input: desired version number 0 -> new
                     Output: that used
   CATBLK  I(256)    File catalog header block
Output:
   BUF     I(768)    First 512 words required for later calls to
                     TABIO
   IERR    I         Error codes from TABINI or TABIO
```

CCMERG

Sorts AIPS CC tables to bring all components at the same cell together, then it sums them, and finally it resorts the file into the original order (by flux of the new components).

```
CCMERG (DISK, CNO, INVER, OUTVER, INPCMP, OUTCMP,
  *   JBUFS, BUFFER, IRET)
Inputs:
   DISK    I     File disk number
   CNO     I     File catalog number
   JBUFS   I     Number R   words in BUFFER
In/out:
   INVER   I     Input CC version number: 0 => MAXVER
   OUTVER  I     Output CC version number: 0 => MAXVER+1
Output:
   INPCMP  I     Number components on input.
   OUTCMP  I     Number components on output.
Common: /MAPHDR/ CATBLK for the affected image file
The routine assumes that the CATBLK is in this common already and
that the file has been opened in the catalog for WRITE.  (The image
file itself does not need to be open.)  The routine assumes that
the DFIL.INC common is initialized especially IBAD (BADDISK).
```

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
```

CHNCOP

Copies selected portions of a Frequency (FQ) table extension file.

```
CHNCOP (INVER, OUTVER, LUNOLD, LUNNEW, VOLOLD,
  *   VOLNEW, CNOOLD, CNONEW, CATOLD, CATNEW, BIF, EIF, FREQID,
  *   BUFF1, WORK1, WORK2, IRET)
Inputs:
   LUNOLD      I     LUN for old file
   LUNNEW      I     LUN for new file
   VOLOLD      I     Disk number for old file.
   VOLNEW      I     Disk number for new file.
   CNOOLD      I     Catalog slot number for old file
   CNONEW      I     Catalog slot number for new file
   BIF         I     First IF to copy to output.
   EIF         I     Last IF to copy.
   FREQID      I     FREQ ID to copy.
   CATOLD(256)I      Catalog header for old file.
In/out:
   INVER       I     Version number to copy, 0 => copy all.
   OUTVER      I     Version number on output file, if more than one
                     copied (INVER=0) this will be the number of the
                     first file.  If OUTVER = 0, it will be taken as
                     1 higher than the previous highest version.
   CATNEW(256)I      Catalog header for new file.
Output:
   BUFF1(256) I    Work buffer
   WORK1(512) D    Work buffer to hold frequency table.
   WORK2(512) I    Work buffer to hold sideband table.
   IRET       I    Return error code  0 => ok
                                        6 => asked for too many IFs.
                                        other = CHNDAT error.
```

CLREFM

Routine to change the format of the CL table from an old one to a new format if necessary.
NOTE: routine uses LUN 45 as a temporary logical unit number.

```
CLREFM (DISK, CNO, VER, CATBLK, LUN, IRET)
Inputs:
```

```
   DISK            I       Volume number
   CNO             I       Catalogue number
   VER             I       Version to check/modify
   CATBLK(256)     I       Catalogue header
   LUN             I       LUN to use
Output:
   IRET            I       Error, 0 => OK

Note, routine will leave no trace of its operation, i.e. CL table
will be closed on output and will have same number as one specified.
Difference will be only that number of columns has changed if that
is required.
```

FLGINI

Creates and initializes FLAG (FG) extension tables.

```
FLGINI (OPCODE, BUFFER, DISK, CNO, VER, CATBLK, LUN,
  *   IFGRNO, FGKOLS, FGNUMV, IERR)
 Inputs:
  OPCODE          C*4 Operation code:
                      'WRIT' = create/init for write or read
                      'READ' = open for read only
  BUFFER(512)     I   I/O buffer and related storage, also defines file
                      if open.
  DISK            I   Disk to use.
  CNO             I   Catalog slot number
  VER             I   FG file version
  CATBLK(256)     I   Catalog header block.
  LUN             I   Logical unit number to use
 Output:
  IFGRNO          I   Next scan number, start of the file if 'READ',
                      the last+1 if WRITE
  FGKOLS(MAXFGC)  I   The column pointer array in order, SOURCE,
                      SUBARRAY, ANTS, TIMERANG, IFS, CHANS, PFLAGS,
                      REASON
  FGNUMV(MAXFGC)  I   Element count in each column.
  IERR            I   Return error code, 0=>OK, else TABINI or TABIO
                      error.
```

FQINI

Creates and initializes frequency (FQ) extension tables.

```
FQINI (OPCODE, BUFFER, DISK, CNO, VER, CATBLK, LUN,
  *   IFQRNO, FQKOLS, FQNUMV, NUMIF, IERR)
Inputs:
  OPCODE          C*4 Operation code:
                      'WRIT' = create/init for write or read
                      'READ' = open for read only
  BUFFER(4096)    I   I/O buffer and related storage, also defines file
                      if open.
  DISK            I   Disk to use.
  CNO             I   Catalog slot number
```

```
  VER              I   FQ file version
  CATBLK(256)      I   Catalog header block.
  LUN              I   Logical unit number to use
Input/output
  NUMIF            I    Number of IFs
Output:
  IFQRNO           I   Next row number, start of the file if 'READ',
                       the last+1 if WRITE
  FQKOLS(MAXFQC)   I   The column pointer array in order:
                       FQID, IFFREQ, IFCHW, IFTBW, IFSIDE
  FQNUMV(MAXFQC)   I   Element count in each column.
  IERR             I   Return error code, 0=>OK, else TABINI or TABIO
                       error.
```

GETNAN

Determines the number of subarrays in a data set from the number of AN files and returns the highest antennas number in each subarray If no antennas are found, one subarray with 28 antennas assumed. If an error occurs, information about subarrays from AN files found is returned; although an error code is returned.

```
GETNAN (DISK, CNO, CATBLK, LUN, BUFFER, NUMAN, IRET)
Inputs:
   DISK      I         Disk to use.
   CNO       I         Catalog slot number
   CATBLK    I(256)    Catalog header block.
   LUN       I         Logical unit number to use
   BUFFER    I(512)    I/O buffer and related storage.
Output:
   NUMAN     I(51)     1st element = no. subarrays followed by
                       the highest antenna number in each subarray.
   IRET      I         Return error code, 0 => ok,
                          else TABINI or TABIO error.
                          10 = no AN files.
```

NDXINI

Creates and initializes INDEX (NX) extention tables.

```
NDXINI (OPCODE, BUFFER, DISK, CNO, VER, CATBLK, LUN,
  *   INXRNO, NXKOLS, NXNUMV, IERR)
Inputs:
  OPCODE        C*4 Operation code:
                    'WRIT' = create/init for write or read
                    'READ' = open for read only
  BUFFER(512)   I   I/O buffer and related storage, also defines file
                    if open.
  DISK          I   Disk to use.
  CNO           I   Catalog slot number
  VER           I   NX file version
  CATBLK(256)   I   Catalog header block.
  LUN           I   Logical unit number to use
Output:
  INXRNO          I   Next scan number, start of the file if 'READ',
```

```
                    the last+1 if WRITE
  NXKOLS(MAXNXC) I  The column pointer array in order, TIME,
                    TIME INTERVAL, SOURCE ID, SUBARRAY, START VIS,
                    END VIS, FREQID.
  NXNUMV(MAXNXC) I  Element count in each column.
  IERR           I  Return error code, 0=>OK, else TABINI or TABIO
                    error.
```

SNINI

Creates and initializes solution (SN) extension tables.

```
SNINI (OPCODE, BUFFER, DISK, CNO, VER, CATBLK, LUN,
  *   ISNRNO, SNKOLS, SNNUMV, NUMANT, NUMPOL, NUMIF, NUMNOD, GMMOD,
  *   RANOD, DECNOD, ISAPPL, SNTYPE, IERR)
Inputs:
  OPCODE       C*4 Operation code:
                   'WRIT' = create/init for write or read
                   'READ' = open for read only
  BUFFER(512)  I   I/O buffer and related storage, also defines file
                   if open.
  DISK         I   Disk to use.
  CNO          I   Catalog slot number
  VER          I   SN file version
  CATBLK(256)  I   Catalog header block.
  LUN          I   Logical unit number to use
Input/output
  NUMANT       I    Number of antennas
  NUMPOL       I    Number of IFs per group
  NUMIF        I    Number of IF groups
  NUMNOD       I    Number of interpolation nodes. Will handle
                    up to 25 interpolation nodes.
  GMMOD        R    Mean gain modulus
  RANOD(*)     R    RA offset of interpolation nodes (deg.)
  DECNOD(*)    R    Dec. offset of interpolation nodes (deg.)
  ISAPPL       L    True if this SN table has been applied to
                    the  CL table.
  SNTYPE       I    "solution" type, 1=>"Clock", 2=>"atmosphere"
Output:
  ISNRNO       I   Next scan number, start of the file if 'READ',
                   the last+1 if WRITE
  SNKOLS(MAXSNC) I   The column pointer array in order, TIME,
                     TIME INT., SOURCE ID., ANTENNA NO., SUBARRAY,
                     FREQ. ID., IFR, MODE NO.,
                     REAL1, IMAG1, DELAY1, RATE1, WEIGHT1, REFANT 1,
                     Following used if 2 polarizations per IF
                     REAL2, IMAG2, DELAY2, RATE2, WEIGHT2, REFANT 2
  SNNUMV(MAXSNC) I   Element count in each column.
  IERR         I   Return error code, 0=>OK, else TABINI or TABIO
                   error.
```

SNREFM

Routine to change the format of the SN table from an old one to the new one if necessary. NOTE: routine uses LUN 45 as a temporary logical unit number.

```
SNREFM (DISK, CNO, VER, CATBLK, LUN, IRET)
Inputs:
   DISK            I       Volume number
   CNO             I       Catalogue number
   VER             I       Version to check/modify
   CATBLK(256)     I       Catalogue header
   LUN             I       LUN to use
Output:
   IRET            I       Error, 0 => OK

Note, routine will leave no trace of its operation, i.e. SN table
will be closed on output and will have same number as one specified.
Difference will be only that number of columns has changed if that
is required.
```

SOUINI

Creates and initializes SOURCE (SU) extension tables.

```
SOUINI (OPCODE, BUFFER, DISK, CNO, VER, CATBLK, LUN,
  *   NUMIF, VELTYP, VELDEF, ISURNO, SUKOLS, SUNUMV, IERR)
Inputs:
  OPCODE          C*4 Operation code:
                      'WRIT' = create/init for write or read
                      'READ' = open for read only
  BUFFER(512)     I   I/O buffer and related storage, also defines file
                      if open.
  DISK            I   Disk to use.
  CNO             I   Catalog slot number
  VER             I   SU file version
  CATBLK(256)     I   Catalog header block.
  LUN             I   Logical unit number to use
Input/Output:
  NUMIF           I   Table keyword, gives the number of IFs
  VELTYP          C*8 Velocity type,
  VELDEF          C*8 Velocity defination 'RADIO','OPTICAL',
Output:
  ISURNO          I   Next scan number, start of the file if 'READ',
                      the last+1 if WRITE
  SUKOLS(MAXSUC)  I   The column pointer array in order, ID. NO.,
                      SOURCE, QUAL, CALCODE, IFLUX, QFLUX, UFLUX,
                      VFLUX, FREQO, BANDWIDTH, RAEPO, DECEPO, EPOCH,
                      RAAPP, DECAPP, LSRVEL, LRESTF, PMRA, PMDEC
  SUNUMV(MAXSUC)  I   Element count in each column.
  IERR            I   Return error code, 0=>OK, else TABINI or TABIO
                      error.
```

TABAN

Does I/O to Antenna (AN) tables. Usually used after setup by ANTINI.

```
TABAN (OPCODE, BUFFER, IANRNO, ANKOLS, ANNUMV, ANNAME,
  *   STAXYZ, ORBPRM, HOSTA, MNTSTA, STAXOF, POLTYA, POLAA, POLCA,
  *   POLTYB, POLAB, POLCB, IERR)
```

```
Inputs:
   OPCODE   C*4         Operation code:
                        'READ' = read entry from table.
                        'WRIT' = write entry in table.
                        'CLOS' = close file, flush on write
   BUFFER   I(512)      I/O buffer and related storage, also defines
                        file if open. Should have been returned by
                        ANTINI or TABINI.
   IANRNO   I           Next scan number to read or write.
   ANKOLS   I(MAXANC)   The column pointer array in order, ANNAME,
                        STABXYZ, ORBPARM, NOSTA, MNTSTA, STAXOF, POLTYA,
                        POLAA, POLCALA, POLTYB, POLAB, POLCALB
   ANNUMV   I(MAXANC)   Element count in each column.
Input/output: (written to or read from antenna file)
   ANNAME   C*8         Station name
   STAXYZ   D(3)        X,Y,Z offset from array center
   ORBPRM   D(*)        Orbital parameters.
   NOSTA    I           Station number
   MNTSTA   I           Mount type, 0=altaz, 1=equatorial, 2=orbiting
   STAXOF   R           Axis offset
   POLTYA   C*2         Feed A feed poln. type 'R','L','X','Y'
   POLAA    R           Feed A feed position angle.
   POLCA    R(*)        Feed A poln. cal parameter. (note 2)
   POLTYB   C*2         Feed B feed poln. type 'R','L','X','Y'
   POLAB    R           Feed B feed position angle.
   POLCB    R(*)        Feed B poln. cal parameters.
Output:
   IANRNO   I           Next GAIN number.
   IERR     I           Error code, 0=>OK else TABIO error.
                        Note: -1=> read but record deselected.
Usage NOTE: use the include 'DANT.INC' for the declarations in
ANTINI and TABAN.
```

TABBL

Does I/O to baseline (BL) extention tables. Usually used after setup by BLINI.

```
TABBL (OPCODE, BUFFER, IBLRNO, BLKOLS, BLNUMV,
  *   NUMPOL, TIME, SOURID, SUBA, ANT1, ANT2, FREQID,
  *   FACMUL, FACADD, IERR)
Inputs:
  OPCODE       C*4 Operation code:
                   'READ' = read entry from table.
                   'WRIT' = write entry in table.
                   'CLOS' = close file, flush on write
  BUFFER(1024) I   I/O buffer and related storage, also defines file
                   if open. Should have been returned by BLINI or
                   TABINI.
  IBLRNO       I   Next entry number to read or write.
  BLKOLS(MAXBLC)   I   The column pointer array in order, TIME,
                   SOURID, SUBARRAY, ANTENNA1, ANTENNA2, FREQID,
                   REALM1, IMAGM1, REALA1, IMAGA1,
                   Following used if 2 polarizations per IF
                   REALM2, IMAGM2, REALA2, IMAGA2.
```

```
  BLNUMV(MAXBLC)   I   Element count in each column.
  NUMPOL       I   Number of polarizations per IF.
Input/output: (written to or read from baseline file)
  TIME         R   Center time of record (Days)
  SOURID       I   Source ID number.
  SUBA         I   Subarray number.
  ANT1         I   First antenna number.
  ANT2         I   Second antenna number.
  FREQID       I   Freqid #
  FACMUL(2,2,m)R   Multiplicative correction,  m IFs
                   second dimension is polarization,
                   (1,*,*) = real, (2,*,*) = imag.
  FACADD(2,2,m)R   Additive correction, m IFs
Output:
  IBLRNO       I   Next solution number.
  IERR         I   Error code, 0=>OK else TABIO error.
                   Note: -1=> read but record deselected.
```

TABBP

Does I/O to bandpass (BP) extention tables. Usually used after setup by BPINI.

```
TABBP (OPCODE, BUFFER, IBPRNO, BPKOLS, BPNUMV,
  *   NUMIF, NUMFRQ, NUMPOL, TIME, INTERV, SOURID, SUBA, ANT,
  *   BANDW, IFFREQ, FREQID, REFANT, REAL, IMAG, IERR)
Inputs:
  OPCODE       C*4 Operation code:
                   'READ' = read entry from table.
                   'WRIT' = write entry in table.
                   'CLOS' = close file, flush on write
  BUFFER(4096) I   I/O buffer and related storage, also defines file
                   if open. Should have been returned by BPINI or
                   TABINI.
  IBPRNO       I   Next entry number to read or write.
  BPKOLS(MAXBPC) I   The column pointer array in order,
                     TIME, INTERVAL, SOURID,
                     SUBARRAY, ANTENNA,
                     BANDW (of individual channel), IFREQ, FREQID,
                     REFANT1, REAL1, IMAG1, .. etc for all channels
                     Following used if 2 polarizations per IF
                     REFANT2, REAL2, IMAG2.
  BPNUMV(MAXBPC) I   Element count in each column.
  NUMIF        I   Number of IF's
  NUMFRQ       I   Number of chns
  NUMPOL       I   Number of polarizations per IF.
Input/output: (written to or read from baseline file)
  TIME         D   Center time of record (Days)
  INTERV       R   Time interval of record (Days)
  SOURID       I   Source ID number.
  SUBA         I   Subarray number.
  ANT          I   Antenna number.
  BANDW        R   Bandwidth of an individual channel (Hz)
  IFFREQ(m)    D   Reference frequency for each IF (Hz)
  FREQID       I   Freq. id number
```

```
  REFANT(2)     I   Reference Antenna; one for each poln
  REAL(2,n,m)   R   Real part of complex bandpass
                    m IFS; n channels; 2 polns
  IMAG(2,n,m)   R   Imag part of complex bandpass
                    m IFS; n channels; 2 polns
Output:
  IBPRNO        I   Next solution number.
  IERR          I   Error code, 0=>OK else TABIO error.
                    Note: -1=> read but polzn #1 flagged
                          -2=> read but polzn #2 flagged
                          -3=> both flagged
```

TABCAL

Does I/O to CALIBRATION (CL) extention tables. Usually used after setup by CALINI.

```
TABCAL (OPCODE, BUFFER, ICLRNO, CLKOLS, CLNUMV,
  *   NUMPOL, NUMIF, TIME, TIMEI, SOURID, ANTNO, SUBA, FREQID, IFR,
  *   GEODLY, GEOPHA, GEORAT, DOPOFF, CLKGD, DCLKGD, CLKPD, DCLKPD,
  *   ATMGD, DATMGD, ATMPD, DATMPD, CREAL, CIMAG, DELAY, RATE, TSYS,
  *   WEIGHT, REFA, IERR)
Inputs:
  OPCODE        C*4 Operation code:
                    'READ' = read entry from table.
                    'WRIT' = write entry in table.
                    'CLOS' = close file, flush on write
  BUFFER(512)   I   I/O buffer and related storage, also defines file
                    if open. Should have been returned by TABINI or
                    TABINI.
  ICLRNO        I   Next scan number to read or write.
  CLKOLS(MAXCLC) I    The column pointer array in order, TIME,
                      TIME INT., SOURCE ID., ANTENNA NO., SUBARRAY,
                      FREQID, IFR (Ionesph. Faraday Rot.),
                      GEODELAY, GEOPHASE, GEORATE, DOPPOFF,
                      CLKGD 1, DCLKGD 1, CLKPD 1, DCLKPD 1,
                      ATMGD 1, DATMGD 1, ATMPD 1, DATPGD 1,
                      REAL1, IMAG1, RATE 1, DELAY 1, TSYS1, WEIGHT1,
                      REFANT 1
                      Following used if 2 polarizations per IF
                      CLKGD 2, DCLKGD 2, CLKPD 2, DCLKPD 2,
                      ATMGD 2, DATMGD 2, ATMPD 2, DATPGD 2,
                      REAL2, IMAG2, RATE 2, DELAY 2, TSYS2, WEIGHT2,
                      REFANT 2
  CLNUMV(MAXCLC) I    Element count in each column.
  NUMPOL        I   Number of polarizations per IF.
  NUMIF         I   Number of IFs.
Input/output: (written to or read from CAL file)
  TIME          D   Center time of CAL record (Days)
  TIMEI         R   Time interval covered by record (days)
  SOURID        I   Source ID as defined in the SOURCE table.
  ANTNO         I   Antenna number.
  SUBA          I   Subarray number.
  FREQID        I   Freqid #
  IFR           R   Ionospheric Faraday Rotation (rad/m**2)
```

```
  GEODLY       D   Geometric delay at TIME (sec)
  GEOPHA       D   Phase of sinusoid (turns)
  GEORAT       D   Time rate of change of GEOPHA (Hz)
  DOPOFF(*)    R   Doppler offset for each IF (Hz)
  CLKGD(2,*)   R   "Clock" Group delay (sec) 1/poln/IF
  DCLKGD(2,*)  R   Time derivative of "Clock" Group delay (sec/sec)
  CLKPD(2,*)   R   "Clock" Phase delay (sec) 1/poln/IF
  DCLKPD(2,*)  R   Time derivative of "Clock" Phase delay (sec/sec)
  ATMGD(2,*)   R   "Atmos" Group delay (sec) 1/poln/IF
  DATMGD(2,*)  R   Time derivative of "Atmos" Group delay (sec/sec)
  ATMPD(2,*)   R   "Atmos" Phase delay (sec) 1/poln/IF
  DATMPD(2,*)  R   Time derivative of "Atmos" Phase delay (sec/sec)
  CREAL(2,*)   R   Real part of the complex gain, 1/poln/IF
  CIMAG(2,*)   R   Imag part of the complex gain, 1/poln/IF
  DELAY(2,*)   R   Residual group delay (sec), 1/poln/IF
  RATE(2,*)    R   Residual fringe rate (Hz), 1/poln/IF
  TSYS(2,*)    R   System temperature (K), 1/poln/IF
  WEIGHT(2,*)  R   Weight of solution, 1/poln/IF
  REFA(2,*)    I   Reference antenna use for cal. solution.
Output:
  ICLRNO       I   Next CAL number.
  IERR         I   Error code, 0=>OK else TABIO error.
                   Note: -1=> read but record deselected.
```

TABFLG

Does I/O to FLAG (FG) extention tables. Usually used after setup by FLGINI.

```
TABFLG (OPCODE, BUFFER, IFGRNO, FGKOLS, FGNUMV,
  *   SOURID, SUBA, ANTS, TIMER, IFS, CHANS, PFLAGS, REASON, IERR)
Inputs:
  OPCODE          C*4 Operation code:
                      'READ' = read entry from table.
                      'WRIT' = write entry in table (must have been
                      opened with 'WRIT'.
                      'FLAG' = like 'WRIT' but entry deselected.
                      'CLOS' = close file, flush on write
  BUFFER(512)     I   I/O buffer and related storage, also defines file
                      if open. Should have been returned by FLGINI or
                      TABINI.
  IFGRNO          I   Next FLAG entry number to read or write.
  FGKOLS(MAXFGC)  I   The column pointer array in order, SOURCE,
                      SUBARRAY, ANTS, TIMERANG, IFS, CHANS, PFLAGS,
                      REASON
  FGNUMV(MAXFGC)    I   Element count in each column.
Input/output: (written to or read from FLAG file)
  SOURID          I   Source ID as defined in the SOURCE table.
  SUBA            I   Subarray number.
  ANTS(2)         I   Antenna numbers, 0=>all
  TIMER(2)        R   Start and end time of data to be flagged (Days)
  IFS(2)          I   First and last IF numbers to flag. 0=>all
  CHANS(2)        I   First and last channel numbers to flag. 0=>all
  PFLAGS(4)       L   Polarization flags, same order as in data.
                      .TRUE. => polarization flagged.
```

```
  REASON          C*24 Reason for flagging
Output:
  IFGRNO          I    Next scan number.
  IERR            I    Error code, 0=>OK else TABIO error.
                       Note: -1=> read but record deselected.
```

TABFQ

Does I/O to frequency (FQ) extension tables. Usually used after setup by FQINI.

```
TABFQ (OPCODE, BUFFER, IFQRNO, FQKOLS, FQNUMV,
  *   NUMIF, FQID, IFFREQ, IFCHW, IFTBW, IFSIDE, IERR)
Inputs:
  OPCODE          C*4 Operation code:
                      'READ' = read entry from table.
                      'WRIT' = write entry in table.
                      'CLOS' = close file, flush on write
  BUFFER(4096)    I   I/O buffer and related storage, also defines file
                      if open. Should have been returned by FQINI or
                      TABINI.
  IFQRNO          I   Next entry number to read or write.
  FQKOLS(MAXFQC)  I   The column pointer array in order,
                      FQID, IFFREQ, IFCHW, IFTBW, IFSIDE
  FQNUMV(MAXFQC   I   Element count in each column.
  NUMIF           I   Number of IF's
Input/output: (written to or read from frequency table)
  FQID            I   Frequency ID number, is random parameter
                      in uv-data.
  IFFREQ(*)       D   Reference frequency for each IF (Hz)
  IFCHW(*)        R   Bandwidth of an individual channel (Hz)
  IFTBW(*)        R   Total bandwidth of the IF (Hz)
  IFSIDE(*)       I   Sideband of the IF (-1 => lower, +1 => upper)
Output:
  IFQRNO          I   Next row number.
  IERR            I   Error code, 0=>OK else TABIO error.
```

TABNDX

Does I/O to INDEX (NX) extention tables. Usually used after setup by NDXINI.

```
TABNDX (OPCODE, BUFFER, INXRNO, NXKOLS, NXNUMV,
  *   TIME, DTIME, IDSOUR, SUBARR, VSTART, VEND, FREQID, IERR)
Inputs:
  OPCODE       C*4 Operation code:
                   'READ' = read entry from table.
                   'WRIT' = write entry in table.
                   'CLOS' = close file, flush on write
  BUFFER(512)  I   I/O buffer and related storage, also defines file
                   if open. Should have been returned by NDXINI or
                   TABINI.
  INXRNO       I   Next scan number to read or write.
  NXKOLS(MAXNXC)  I   The column pointer array in order, TIME,
                      TIME INTERVAL, SOURCE ID, SUBARRAY, START VIS,
                      END VIS, FREQID.
```

```
    NXNUMV(MAXNXC)  I   Element count in each column, set by NDXINI.
  Input/output: (written to or read from INDEX file)
    TIME          R   Center time of the scan (Days)
    DTIME         R   Duration of scan (Days)
    IDSOUR        I   Source ID as defined in then SOURCE table.
    SUBARR        I   Subarray number.
    VSTART        I   First visibility number in file.
    VEND          I   Last visibility number in file.
    FREQID        I   Freqid of scan
  Output:
    INXRNO        I   Next scan number.
    IERR          I   Error code, 0=>OK else TABIO error.
                      Note: -1=> read but record deselected.
```

TABSN

Does I/O to solution (SN) extention tables. Usually used after setup by SNINI.

```
  TABSN (OPCODE, BUFFER, ISNRNO, SNKOLS, SNNUMV,
    *   NUMPOL, TIME, TIMEI, SOURID, ANTNO, SUBA, FREQID, IFR,
    *   NODENO, CREAL, CIMAG, DELAY, RATE, WEIGHT, REFA, IERR)
  Inputs:
    OPCODE        C*4 Operation code:
                      'READ' = read entry from table.
                      'WRIT' = write entry in table.
                      'CLOS' = close file, flush on write
    BUFFER(512)   I   I/O buffer and related storage, also defines file
                      if open. Should have been returned by TABINI or
                      TABINI.
    ISNRNO        I   Next scan number to read or write.
    SNKOLS(MAXSNC) I    The column pointer array in order, TIME,
                        TIME INT., SOURCE ID., ANTENNA NO., SUBARRAY,
                        FREQID, IFR, NODE NO.,
                        REAL1, IMAG1, DELAY1, RATE1, WEIGHT1, REFANT 1,
                        Following used if 2 polarizations per IF
                        REAL2, IMAG2, DELAY2, RATE2, WEIGHT2, REFANT 2,
    SNNUMV(MAXSNC) I    Element count in each column.
    NUMPOL        I   Number of polarizations per IF.
  Input/output: (written to or read from solution file)
    TIME          D   Center time of solution record (Days)
    TIMEI         R   Time interval covered by record (days)
    SOURID        I   Source ID as defined in the SOURCE table.
    ANTNO         I   Antenna number.
    SUBA          I   Subarray number.
    FREQID        I   Freqid #
    IFR           R   Ionospheric Faraday Rotation (rad/m**2)
    NODENO        I   Interpolation node number
    CREAL(2,*)    R   Real part of the complex gain, 1 /Poln/IF
    CIMAG(2,*)    R   Imag part of the complex gain, 1 /Poln/IF
    DELAY(2,*)    R   Residual group delay (sec), 1 /Poln/IF
    RATE(2,*)     R   Residual fringe rate (Hz), 1 /Poln/IF
    WEIGHT(2,*)   R   Weight of solution, 1 /Poln/IF
    REFA(2,*)     R   Ref. ant. of solution, 1 /Poln/IF
  Output:
```

```
  ISNRNO        I   Next solution number.
  IERR          I   Error code, 0=>OK else TABIO error.
                    Note: -1=> read but record deselected.
```

TABSOU

Does I/O to SOURCE (SU) extention tables. Usually used after setup by SOUINI.

```
TABSOU (OPCODE, BUFFER, ISURNO, SUKOLS, SUNUMV, IDSOU, SOUNAM,
  *   QUAL, CALCOD, FLUX, FREQO, BANDW, RAEPO, DECEPO, EPOCH,
  *   RAAPP, DECAPP, LSRVEL, LRESTF, PMRA, PMDEC, IERR)
Inputs:
  OPCODE          C*4 Operation code:
                      'READ' = read entry from table.
                      'WRIT' = write entry in table.
                      'CLOS' = close file, flush on write
  BUFFER(768)     I   I/O buffer and related storage, also defines file
                      if open. Should have been returned by SOUINI or
                      TABINI.
  ISURNO          I   Next scan number to read or write.
  SUKOLS(MAXSUC)  I   The column pointer array in order, ID. NO.,
                      SOURCE, QUAL, CALCODE, IFLUX, QFLUX, UFLUX,
                      VFLUX, FREQO, BANDWIDTH, RAEPO, DECEPO, EPOCH,
                      RAAPP, DECAPP, LSRVEL, LRESTF, PMRA, PMDEC
  SUNUMV(MAXSUC)  I   Element count in each column.
Input/output: (written to or read from SoUrce file)
  IDSOUR          I   Source ID as defined in the SOURCE table.
  SOUNAM          C*16 Source name
  QUAL            I   Source qualifier.
  CALCOD          C*4 Calibrator code
  FLUX(4,*)       R   Total flux density I, Q, U, V pol, (Jy)
                      1 set per IF.
  FREQO(*)        D   Frequency offset (Hz) from IF nominal.
  BANDW           D   Bandwidth (Hz)
  RAEPO           D   Right ascension at mean EPOCH (degrees)
  DECEPO          D   Declination at mean EPOCH (degrees)
  EPOCH           D   Mean Epoch for position in yr. since year 0.0
  RAAPP           D   Apparent Right ascension (degrees)
  DECAPP          D   Apparent Declination(degrees)
  LSRVEL(*)       D   LSR velocity (m/sec) of each IF
  LRESTF(*)       D   Line rest frequency (Hz) of each IF
  PMRA            D   Proper motion (deg/day) in RA
  PMDEC           D   Proper motion (deg/day) in declination
Output:
  ISURNO          I   Next source number.
  IERR            I   Error code, 0=>OK else TABIO error.
                      Note: -1=> read but record deselected.
```

Chapter 14

FITS Tapes

14.1 Overview

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

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

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

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

14.2 Philosophy

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

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

The main advantage of FITS is its flexibility. Due to the self-defining nature of the files, a large range of data transport needs are fulfilled. The introduction of new keywords gives the ability to add new pieces

of information as needed, and the use of generalized extension files allows almost unlimited flexibility in the type of information to be stored. Thus, FITS can grow with the needs of the Astronomical community.

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

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

14.3 Image Files

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

14.3.1 Overall Structure

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

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

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

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

14.3.2 Header Records

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

```
keyword = value                / comment
```

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

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

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

Keywords

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

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

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

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

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

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

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

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

History

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

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

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

AIPS Non-standard Image File Keywords

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

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

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

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

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

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

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

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

Coordinate Systems

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

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

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

Example Image Header

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

```
0000000001111111111222222222233333333334444444444555555555666666666
1234567890123456789012345678901234567890123456789012345678901234567890123456789
SIMPLE  =                    T /
BITPIX  =                   16 /
NAXIS   =                    4 /
NAXIS1  =                 2048 /
NAXIS2  =                 1024 /
NAXIS3  =                    1 /
NAXIS4  =                    1 /
EXTEND  =                    T / Tables following main image
BLOCKED =                    T / Tape may be blocked
OBJECT  = '3C405   '           / Source name
TELESCOP= '        '           /
INSTRUME= '        '           /
OBSERVER= 'PERL    '           /
DATE-OBS= '27/10/82'           /Observation start date dd/mm/yy
DATE-MAP= '14/07/83'           /Date of last processing dd/mm/yy
BSCALE  =    7.04625720812E-05 /Real = Tape * BSCALE + BZERO
BZERO   =    2.18688869476E+00 /
BUNIT   = 'JY/BEAM '           /Units of flux
EPOCH   =     1.950000000E+03 /Epoch of RA, Dec
DATAMAX =     4.495524406E+00 /Max pixel value
DATAMIN =    -1.217470840E-01 /Min pixel value
CTYPE1  = 'RA---SIN'           /
CRVAL1  =    2.99435165226E+02 /
CDELT1  =    -4.166666986E-05 /
CRPIX1  =     1.024000000E+03 /
CROTA1  =     0.000000000E+00 /
CTYPE2  = 'DEC--SIN'           /
CRVAL2  =    4.05961940065E+01 /
CDELT2  =     4.166666986E-05 /
CRPIX2  =     5.130000000E+03 /
CROTA2  =     0.000000000E+00 /
CTYPE3  = 'FREQ    '           /
CRVAL3  =    4.86635000000E+09 /
CDELT3  =     1.250000000E+07 /
CRPIX3  =     1.000000000E+00 /
CROTA3  =     0.000000000E+00 /
CTYPE4  = 'STOKES  '           /
CRVAL4  =    1.00000000000E+00 /
CDELT4  =     1.000000000E+00 /
CRPIX4  =     1.000000000E+00 /
CROTA4  =     0.000000000E+00 /
HISTORY UVLOD /DATA BASE CREATED BY USER 76 AT 14-JUL-1983 10:17:08
HISTORY UVLOD OUTNAME='CYGA       ' OUTCLASS='XY    '
HISTORY UVLOD OUTSEQ=    1   OUTDISK=  3
        ...
ORIGIN  = 'AIPSNRAO VLA VAX3       '   /
DATE    = '19/08/83'                   / TAPE WRITTEN ON DD/MM/YY
HISTORY AIPS   IMNAME='CYGA        ' IMCLASS='IMAP  ' IMSEQ=   1
```

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

Units

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

14.3.3 Data Records

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

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

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

14.4 Random Group (UV data) Files

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

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

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

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

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

14.4.1 Header Record

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

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

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

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

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

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

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

14.4.2 Data Records

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

Units of Random Parameters

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

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

Units of the Regular Axis Coordinates

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

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

Weights and Flagging

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

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

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

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

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

Antennas and Subarrays

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

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

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

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

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

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

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

Coordinates

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

Sort Order

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

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

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

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

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

14.4.3 Typical VLA Record Structure

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

```
00000000011111111112222222223333333333444444444455555555566666666666
12345678901234567890123456789012345678901234567890123456789012345678
SIMPLE  =                    T /
BITPIX  =                   16 /
NAXIS   =                    7 /
NAXIS1  =                    0 /No standard image just group
NAXIS2  =                    3 /
NAXIS3  =                    4 /
NAXIS4  =                    1 /
NAXIS5  =                    2 /
NAXIS6  =                    1 /
NAXIS7  =                    1 /
EXTEND  =                    T /Tables following main image
BLOCKED =                    T /Tape may be blocked
OBJECT  = 'MULTI   '           /Source name
TELESCOP= 'VLA     '           /
INSTRUME= 'VLA     '           /
OBSERVER= 'VC35    '           /
DATE-OBS= '02/12/84'           /Observation start date dd/mm/yy
DATE-MAP= '09/11/86'           /Date of last processing dd/mm/yy
BSCALE  =   4.98494155534E-05 /Real = tape * BSCALE + BZERO
BZERO   =   0.00000000000E+00 /
BUNIT   = 'JY      '           /Units of flux
EPOCH   =     1.950000000E+03 /Epoch of RA, Dec
BLANK   =              -32768 /Tape value of blank pixel
CTYPE2  = 'COMPLEX '           /
CRVAL2  =   1.00000000000E+00 /
CDELT2  =     1.000000000E+00 /
CRPIX2  =     1.000000000E+00 /
CROTA2  =     0.000000000E+00 /
CTYPE3  = 'STOKES  '           /
CRVAL3  =  -1.00000000000E+00 /
CDELT3  =    -1.000000000E+00 /
CRPIX3  =     1.000000000E+00 /
CROTA3  =     0.000000000E+00 /
CTYPE4  = 'FREQ    '           /
CRVAL4  =   1.66499989984E+09 /
CDELT4  =     5.000000000E+07 /
```

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

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

14.4.4 Single Dish Data

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

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

Single Dish Random Parameters

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

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

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

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

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

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

Single Dish Regular Axis Coordinates

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

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

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

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

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

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

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

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

Example Single Dish Data File Header

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

```
0000000001111111111222222222233333333334444444444555555555566666666661
1234567890123456789012345678901234567890123456789012345678901234567890123456789
SIMPLE  =                    T /
BITPIX  =                   16 /
NAXIS   =                    6 /
NAXIS1  =                    0 /No standard image just group
NAXIS2  =                    3 /
NAXIS3  =                    1 /
NAXIS4  =                   16 /
NAXIS5  =                    1 /
NAXIS6  =                    1 /
EXTEND  =                    T /Tables following main image
BLOCKED =                    T /Tape may be blocked
OBJECT  =  'ALL SKY '           /Source name
TELESCOP=  '300 FT  '           /
INSTRUME=  '6 CM    '           /
OBSERVER=  'PHANTOM '           /
DATE-OBS=  '02/12/86'           /Observation start date dd/mm/yy
DATE-MAP=  '09/11/87'           /Date of last processing dd/mm/yy
BSCALE  =    4.98494155534E-05 /Real = tape * BSCALE + BZERO
BZERO   =    0.00000000000E+00 /
BUNIT   =  'JY      '           /Units of flux
EPOCH   =       1.950000000E+03 /Epoch of RA, Dec
BLANK   =                -32768 /Tape value of blank pixel
CTYPE2  =  'COMPLEX '           /
CRVAL2  =    1.00000000000E+00 /
CDELT2  =       1.000000000E+00 /
CRPIX2  =       1.000000000E+00 /
CROTA2  =       0.000000000E+00 /
CTYPE3  =  'STOKES  '           /
CRVAL3  =   -1.00000000000E+00 /
```

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

14.5 Extension Files

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

14.5.1 Standard Extension

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

```
keyword = value     / comment
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

14.5.2 Tables Extension

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

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

Tables Header Record

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

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

14. TTYPEnnnn (character) the label for field nnn. (optional, order arbitrary)

15. TUNITnnn (character) the physical units of field nnn. (optional, order arbitrary)

16. TSCALnnn (floating) the scale factor for field nnn. True_value = tape_value * TSCAL + TZERO. Note: TSCALnnn and TZEROnnn are not relevant to A-format fields. Default value is 1.0 (optional, order arbitrary)

17. TZEROnnn (floating) the offset for field nnn. (See TSCALnnn.) Default value is 0.0 (optional, order arbitrary)

18. TNULLnnn (character) the (tape) value of an undefined value. Note: an exact left-justified match to the field width as specified by TFORMnnn is required. (optional, order arbitrary)

19. AUTHOR (character) the name of the author or creator of the table. (optional, order arbitrary)

20. REFERENC (character) the reference for the table. (optional, order arbitrary)

21. END must always be the last keyword and the remainder of the record must be blank filled.

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

Table Data Records

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

Example Table Header and Data

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

```
         1         2         3         4         5         6         7
1234567890123456789012345678901234567890123456789012345678901234567890123456789
XTENSION= 'TABLE   '           / Extension type
BITPIX  =                    8 / Printable ASCII codes
NAXIS   =                    2 / Table is a matrix
NAXIS1  =                   60 / Width of table in characters
NAXIS2  =                  449 / Number of entries in table
PCOUNT  =                    0 / Random parameter count
GCOUNT  =                    1 / Group count
TFIELDS =                    3 / Number of fields in each row
EXTNAME = 'AIPS CC '           / AIPS clean components
EXTVER  =                    1 / Version number of table
TBCOL1  =                    1 / Starting char. pos. of field n
TFORM1  = 'E15.6   '           / Fortran format of field n
TTYPE1  = 'FLUX    '           / Type (heading) of field n
TUNIT1  = 'JY      '           / Physical units of field n
TSCAL1  =                  1.0 / Scale factor for field n
TZERO1  =                  0.0 / Zero point for field n
TBCOL2  =                   17 / Starting char. pos. of field n
TFORM2  = 'E15.6   '           / Fortran format of field n
TTYPE2  = 'DELTAX  '           / Type (heading) of field n
TUNIT2  = 'DEGREES '           / Physical units of field n
TSCAL2  =                  1.0 / Scale factor for field n
TZERO2  =                  0.0 / Zero point for field n
```

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

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

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

                    .
                    .
                    .
```

14.5.3 3- D Tables Extension

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

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

3-D Tables Header Record

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

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

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

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

Table Data Records

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

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

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

```
                1         2         3
01234567890123456789012345678901
seeeeeeeemmmmmmmmmmmmmmmmmmmmmmm
```

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

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

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

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

```
          1         2         3
01234567890123456789012345678901
seeeeeeeeeeemmmmmmmmmmmmmmmmmmmm

3       4         5         6
23456789012345678901234567890123
mmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm
```

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

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

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

Example 3- D Table Header

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

```
        1         2         3         4         5         6
123456789012345678901234567890123456789012345678901234567890123456789012345678901234
XTENSION= 'A3DTABLE'           / Extension type
BITPIX  =                    8 / Binary data
NAXIS   =                    2 / Table is a matrix
NAXIS1  =                   12 / Width of table in bytes
NAXIS2  =                  449 / Number of entries in table
PCOUNT  =                    0 / Random parameter count
GCOUNT  =                    1 / Group count
TFIELDS =                    3 / Number of fields in each row
EXTNAME = 'AIPS CC '           / AIPS CLEAN components
EXTVER  =                    1 / Version number of table
TFORM1  = '1E      '           / Count and data type of field n
TTYPE1  = 'FLUX    '           / Type (heading) of field n
TUNIT1  = 'JY      '           / Physical units of field n
TFORM2  = '1E      '           / Count and data type of field n
TTYPE2  = 'DELTAX  '           / Type (heading) of field n
TUNIT2  = 'DEGREES '           / Physical units of field n
TFORM3  = '1E      '           / Count and data type of field n
TTYPE3  = 'DELTAY  '           / Type (heading) of field n
TUNIT3  = 'DEGREES '           / Physical units of field n
END
```

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

14.5.4 Older AIPS Tables

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

General Form of Header

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

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

Data Records

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

CC Files

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

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

AN Files

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

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

14.6 AIPS FITS INCLUDEs

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

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

14.6.1 DFUV.INC

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

14.6.2 DFIT.INC

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

14.6.3 VFUV.INC

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

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

14.6.4 VFIT.INC

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

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

14.7 AIPS FITS Parsing Routines

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

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

2. FPARSE parses a FITS header card, interpretes the card image, and puts the data value into the correct location in the AIPS catalog header. This routine is for standard uv FITS headers, but with the substitution of the INCLUDEs DFIT.INC and VFIT.INC for DFUV.INC and VFUV.INC, would work for images as well. The routine will work for FITS image tapes written on the VLA pipeline.

3. GETCRD obtains a given card image from a header block of FITS data and looks for keywords in a supplied table.

4. GETSYM finds the next symbol in a buffer. A symbol is defined to begin with a letter and have up to 8 alpha-numeric characters.

5. GETLOG obtains the value of a logical variable from a buffer.

6. GETNUM converts an ASCII numeric field into a double precision value.

7. GETSTR obtains a character string from a buffer.

8. GETKEY returns keyword values from a buffer.

9. GTWCRD reads input buffer and finds symbol value from a list of possible symbols; accepts wild cards.

10. IDWCRD searches a character string for a symbol value from a list of possible symbols using wild cards. Like GTWCRD except it works from a character string rather than a FITS buffer.

11. R3DTAB reads 3-D table data from tape and translates it to an AIPS table file.

12. RWTAB translates ASCII table data to an AIPS table file.

13. TABAXI reads a card image array containing a FITS table header and returns the values for required keywords. Like EXTREQ except it works from a character string rather than a FITS buffer.

14. TABHDK parses a character card image processing FITS table header keywords. Like TABHDR except it works from a character string rather than a FITS buffer.

15. TABHDR reads a FITS table header and returns the values for recognized symbols.

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

14.7.1 EXTREQ

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

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

14.7.2 FPARSE

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

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

14.7.3 GETCRD

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

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

14.7.4 GETKEY

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

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

14.7.5 GETLOG

Obtains the value of a logical variable from buffer.

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

14.7.6 GETNUM

Converts ASCII numeric field into double precision number.

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

14.7.7 GETSTR

Obtains a character string from a buffer.

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

14.7.8 GETSYM

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

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

14.7.9 GTWCRD

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

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

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

14.7.10 IDWCRD

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

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

14.7.11 R3DTAB

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

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

14.7.12 RWTAB

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

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

14.7.13 TABAXI

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

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

14.7.14 TABHDK

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

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

14.7.15 TABHDR

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

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

14.8 References

Wells, Greisen, and Harten 1981, *Astronomy and Astrophysics* Supplement series, vol. 44, pp 363 - 370. Greisen and Harten, 1981, *Astronomy and Astrophysics* Supplement Series, vol. 44, pp 371 - 374. Harten, Grosbøl, Tritton, Greisen and Wells 1984, draft reproduced in the IAU Commission 9 *Astronomical Image Processing Circular.* Also appeared in *Mem. S. A. It.*, 56, 437 (1985).

Chapter 15

The Z Routines

15.1 Overview

The AIPS system has a number of types of routines in which the details depend on the hardware and/or operating system upon which the code is running. These types of routines, which may vary from system to system, are denoted by the first letter of the name and include: (1) those which depend primarily on the operating system or CPU hardware (denoted by a "Z", thus the "Z" routines), (2) those which depend on the image display (TV) hardware and/or software (the "Y" routines) and (3) those which depend on array or vector hardware and/or software (the "Q" routines). This chapter discusses the "Z" routines; the "Y" and "Q" routines are discussed elsewhere in this manual.

In principle, all that is required to make AIPS work on a new machine is to develop a disk file structure and create a set of "Z", "Q" and "Y" routines to interface AIPS programs to the operating system, the file structure, the array or vector functions and the image display. If routines other than "Z" (or "Y" and "Q") routines are modified, then they will have to be modified every time the AIPS system is updated. For this reason, we recommend that *no routines other than "Z", "Y" or "Q" routines should be modified as part of the installation on a new system.*

This chapter will describe the functions of the upper layer of Z routines; in any implementation, there will probably be additional lower level machine-dependent routines. These Z routines form the basis of a virtual operating system under which the application code runs. Careful study of an existing implementation of AIPS is recommended before attempting a new installation.

The upper layer of the "Z" routines is defined by the contents of the directory with logical name APLGEN. Many of the routines in this directory are in fact generic and call second level routines to do the actual system-dependent operation. The second level "Z" routines called from first level "Z" routines in APLGEN are also included in APLGEN although many are in stubbed form. Many of the first level "Z" routines appear in APLGEN in stubbed form for completeness. Second (or lower) level "Z" routines should never be called from non "Z" routines.

For purposes of discussion, the Z routines will be divided up into a number of overlapping categories:

1. *System Functions* - These routines do various system functions, such as starting and stopping processes, inquiring what processes are running, and inquiring how much space is available on a given disk drive.
2. *Disk I/O and File Manipulation* - These routines create, destroy, expand, contract, open, close, read, and write disk files.
3. *Device I/O* - These routines talk to the terminals, the tape drive, graphics devices, image displays, etc. This area overlaps heavily with the disk I/O area.
4. *Data Manipulation Routines* - These routines convert data formats from external numbers and characters to local and vice versa, and move bits and bytes.
5. *Directory and Text File Routines* - These routines read the directories for, and read and/or write the contents of, text files.

6. *Television I/O routines.* These routines are discussed in the chapter on televisions and only a brief description is given in this chapter.

7. *Virtual Devices* - These routines communicate through network protocols to other processes, which may reside on another computer, which are connected to a physical device.

8. *Miscellaneous* - There are a number of routines, such as that which initializes the Device Characteristics Common, which do not easily fit in one of the other categories.

A detailed description of the call sequences to the non-TV specific first level "Z" routines and listings of the relevant INCLUDE files are at the end of this chapter.

15.1.1 Device Characteristics Common

Many of the parameters describing the host operating system and installation in AIPS programs are carried in the Device Characteristics Common, which is obtained using the include DDCH.INC. The text of this include file can be found at the end of this chapter.

The contents of the Device Characteristics common are initialized by a call to ZDCHIN. Details of the call sequence can be found at the end of this chapter. Many of the values in the Device Characteristics common are read from a disk file. The values in this file can be read and changed using the stand-alone utility program SETPAR. The constants kept in this common, the values in DEVTAB, and the use of logical unit numbers are described in the chapter on disk I/O.

15.1.2 FTAB

The FTAB array in the device characteristics common is used to keep AIPS and system I/O tables. The FTAB has separate areas for the three different kinds of I/O: (1) device I/O to devices which may not need I/O tables, (2) non-map or regular I/O, which is single buffered, wait mode I/O and 3) map I/O, which can be double buffered, non-wait mode I/O.

The FTAB has space for one system I/O table for non-map files and two system I/O tables for map files and space for 16 integer words for application routines to use for map I/O. The size of the entries in FTAB for the different types of I/O are carried in the Device Characteristics Common. The type of the I/O (map or non-map) is declared by the calling routine to the file/device open routine ZOPEN. In general, the FTAB is used to carry any system dependent information necessary for I/O to the device or file. Note: the size of FTAB is dimensioned in the DDCH.INC include for all applications.

The FTAB is divided up by ZDCHIN into three areas, one for each type of I/O. These areas are described in the following:

1. *Non-FTAB I/O* - This area has NTAB1 entries, each NBTB1 bytes long. The first integer word in each entry is zero, if that entry is not in use, and the LUN of the corresponding device, if the entry is in use.

2. *FTAB "non-map" I/O* - This area has NTAB2 entries, each NBTB2 bytes long. The first of these is zero, if that entry is not in use, and the LUN of the corresponding device, if the entry is in use. Following is space for one copy of whatever system I/O table is required for the host system.

3. *FTAB "map" I/O* - This area has NTAB3 entries, each NBTB3 bytes long. The first 16 integer words in each entry are reserved for application routines; the first of these is zero, if that entry is not in use, and the LUN of the corresponding device, if the entry is in use. Following these 16 integers is space for two copies of whatever system I/O table is required for the host system.

Note that a byte is defined in this manual as half a integer, except for most tape and other device I/O where it means 8 bits.

15.1.3 Logicals

The specification of directories inside of AIPS is almost always in terms of a logical directory; that is a variable whose value is the name of the directory or otherwise specifies the directory. Under VMS these are VMS logicals and under Unix they are environment variables. If AIPS is to be implemented in an operating system that does not have an equivalent facility then it must be invented for the AIPS environment. In file names these logicals are given in the form used in VMS, *i.e.* "logical:filename".

15.1.4 Disk Files

The AIPS system uses binary files for data and text files for source code and other information. The location and physical name of the various files depends very much on the host system and installation. The physical name of a file is derived by ZPHFIL and the location of a file is determined by ZPHFIL and ZOPEN (or ZTOPEN or ZTXOPN for text files).

Binary (data) files

Binary files are divided into two types, "map" and "non-map" files corresponding to the two types of I/O. Normally most AIPS binary files on a given disk are put in a single area or directory. Current implementations of AIPS use 8 characters for the basic physical name and 3 more if private catalogs are supported. Applications software will handle up to 48 characters in a name including logicals and/or directory names.

An example from a VAX system with private catalogs is "DA0n:ttrcccvv.uuu" ; where n is the one relative AIPS disk drive number, DA0n: is a logical variable which is assigned to a directory, tt is a two character file type (e.g., 'MA'), r is the data format version code ('A','B',...), ccc is the catalog slot number (hex), vv is the version (hex) (01 for "MA" and "UV" files), and uuu is the user's number in hexadecimal notation.

"Map" type files are files on which it should be possible to double buffer. It should be possible to expand or contract "map". These files should be as contigious as possible since contiguous files are more efficient, but they cause problems finding space for large files. These files should be capable of random access with I/O beginning on a disk sector boundary.

"Non-map" files should also be expandable and contractable. These files should be capable of random access with I/O beginning on a disk sector boundary. In practice, there is no longer any real difference between "map" and "non-map" files and the distinction in AIPS is purely for historical reasons.

Text files

Text files are used for storing source code and control information such as the RUN and HELP/INPUT files as well as a means of communicating information between AIPS and the external world. There are currently two systems of accessing text files; the older one which is read-only is for access to AIPS system text files. A second system will read or write arbitrary text files.

AIPS system text files are accessed using a combination of ZTOPEN, ZTREAD and ZTCLOS. The type of the text file is specified to ZPHFIL as one of several types; the directory may be further selected by the ZTOPEN argument VERSON which can specify the version (directory or area). The member (or file) name is specified to ZTOPEN and may contain up to eight characters. These types, and the files kept in each of the AIPS system text file directories, are described in the following:

1. HE - These are the HELP files which specify which AIPS adverbs are to be sent to tasks and contain the primary user documentation.

2. RU - The RUN files usually contain instructions for the AIPS program. Other types of text files may appear in this area as input for AIPS tasks.

General text files may be accessed using a combination of ZTXOPN, ZTXIO, and ZTXCLS. ZTXOPN will open a file whose specifications is up to 48 characters and must include a logical (environment variable under Unix) pointing to the directory. A file to be written must not previously exist unless the APPEND option is specified. ZTXIO can read or write lines to/from this text file and ZTXCLS will close it.

15.2 System Functions

There are a number of functions involving processes or system resources, which must be done in a system-dependent fashion. These include controlling processes (starting, killing, suspending and resuming) and determining the time, date, name of the current process, and the amount of CPU time used by the current task. Some of these may require special privileges.

The AIPS interactive program may start independent processes, called tasks, which do most of the computations. In order to start a task, AIPS first writes the task's adverbs (determined from the associated HELP file and the current POPS adverb values) together with an initial value of the task return code (-999) into the task data (TD) file, closes the TD file, and starts the task.

AIPS then loops with a fixed time delay (3 sec. for interactive, 8 sec. for interactive with POPS adverb DOWAIT=TRUE and 20 sec. for batch) until one of two conditions exist. These conditions are (1) the value in the TD file of the return code has changed or (2) the task is no longer running. In case 1, AIPS resumes normal operation; in case 2, if the value of the return code is unchanged, the task is assumed to have failed and any scratch files are destroyed. In case 1 or case 2, if the value of the return code is modified, AIPS continues and processes the return code. A non-zero return code indicates that the task failed.

The following list gives a short description of the first level system routines; complete descriptions of the call sequences can be found at the end of this chapter.

- ZABORT establishes or carries out (when appropriate) abort handling.
- ZACTV8 activate the requested program, returning process ID information.
- ZCPU return current process CPU time and IO count.
- ZDATE return the local date.
- ZDCHIN initialize message, device and Z-routine characteristics commons.
- ZDELAY delay current process a specified interval.
- ZFREE display available disk space.
- ZGNAME get name of current process.
- ZPRIO raise or lower the process priority.
- ZSETUP performs system-level operations after VERNAM, TSKNAM, NPOPS known.
- ZSTAIP does any system cleanup needed at the end of interactive AIPS session.
- ZTACTQ inquires if a task is currently active on the local computer.
- ZTIME return the local time of day.
- ZTKILL deletes (or kills) the specified process.
- ZTQSPY display AIPS account or all processes running on the system.
- ZTRLOG translate a logical name.
- ZWHOMI determines AIPSxn task name; sets NPOPS, assigns TV and TK devices.

The following items are second level system routines in APLGEN; others may be needed in a given implementation.

- ZABOR2 establishes or carries out (when appropriate) abort handling.
- ZDCHI2 initialize device and Z-routine characteristics commons - local vals.
- ZDCHIC set more system parameters; make them available to C routines.

- ZDELA2 delay current process a specified interval.
- ZFRE2 return AIPS data disk free space information.
- ZPRI2 raise or lower the process priority.
- ZSTAI2 does any system cleanup needed at the end of interactive AIPS session.
- ZTACT2 inquires if a task is currently active on the local computer.
- ZTQSP2 display AIPS account or all processes running on the system.

15.3 Disk I/O and File Manipulation Routines

This section describes the routines needed for manipulating disk data (binary) files. The physical names of disk data files are always constructed by ZPHFIL and these files are always opened by ZOPEN. There are separate routines for writing to the message file (ZMSGCL, ZMSGDK, and ZMSGOP) to avoid recursion when reporting an error message from one of the I/O routines.

Short descriptions of the first level disk file routines are given in the following list; detailed descriptions of the call sequences are given at the end of the chapter.

- ZCLOSE closes open devices: disk, line printer, terminal.
- ZCMPRS release space from the end of an open disk file.
- ZCREAT creates a disk file.
- ZDESTR destroy a closed disk file.
- ZEXIST return file size and, consequently, whether file exists.
- ZEXPND expand an open disk file — either map or non-map are allowed.
- ZFIO reads and writes single 256-integer records to non-map disk files.
- ZFULLN convert file name to full pathname with no logicals.
- ZMIO random-access, quick return (double buffer) disk I/O for large blocks.
- ZMKTMP convert a "temporary" file name into a unique name.
- ZMSGCL close Message file or terminal.
- ZMSGDK disk I/O to message file.
- ZMSGOP open a message file or message terminal.
- ZMSGXP expand the message file.
- ZOPEN open binary disk files and line printer and TTY devices.
- ZPHFIL construct a physical file or device name from AIPS logical parameters.
- ZPHOLV construct a physical file - version for UPDAT.
- ZRENAM rename a disk file.
- ZTFILL zero-fill, initialize a file I/O table (FTAB) entry.
- ZWAIT wait for asynchronous ("MAP") I/O to finish.

The following items are second level disk I/O routines in APLGEN; others may be needed in a given implementation.

- ZCMPR2 truncate a disk file, returning blocks to the system.
- ZCREA2 create the specified disk file.
- ZDACLS close a disk file.
- ZDAOPN open the specified disk file.
- ZDEST2 destroy a closed disk file.
- ZEXIS2 return size of disk file and if it exists.
- ZEXPN2 expand an open disk file.
- ZFI2 read/write one 256-integer record from/to a non-map disk file.
- ZMI2 read/write large blocks of data from/to disk, quick return.
- ZMSGWR call MSGWRT based on call arguments - for C routines to call MSGWRT.
- ZPATH convert a file name 'Logical:file' to full path name.
- ZRENA2 rename a file.
- ZWAI2 wait for read/write large blocks of data from/to disk.

15.4 Device (non-disk) I/O Routines

Several of the routines discussed in the disk I/O section will also work on other devices. There are a number of special functions required for non-disk devices. One example of these is the routine to talk to a terminal; some operating systems don't allow Fortran I/O to a terminal, so this I/O is done through the routine ZTTYIO.

The following list gives a short description of these routines; complete descriptions of the call sequences can be found at the end of this chapter.

- ZBKLD1 initialize environment for BAKLD.
- ZBKLD2 does BACKUP operation: load images from tape to directory.
- ZBKLD3 clean up system things for BAKLD ending.
- ZBKTP1 initialize BACKUP to tape operation for BAKTP.
- ZBKTP2 write a cataloged file plus extensions to BACKUP tape in BAKTP.
- ZBKTP3 clean up host environment at end of BAKTP.
- ZDOPRT reads bit file and causes it to be plotted on printer/plotter.
- ZENDPG advance printer if needed to avoid electrostatic-printer "burn-out"
- ZLASCL close and spool a laser printer print/plot file.
- ZLASIO open, write to, close and spool a laser printer print/plot file.
- ZLASOP open a laser printer print/plot file.
- ZLPCLS close an open printer device.
- ZLPOPN open a line-printer text file.
- ZLWIO open, write to, close and spool a PostScript print/plot file.

- ZLWOP open a PostScript (LaserWriter) print/plot file.
- ZMOUNT mount or dismount magnetic tape device.
- ZPRMPT prompt user and read 80-characters from CRT screen.
- ZPRPAS prompt user and read 12-character password (invisible) from CRT.
- ZTAPE mount, dismount, position, write EOF, etc. for tapes.
- ZTAPIO tape operations for IMPFIT (compressed FITS transport tape).
- ZTKBUF flush TK buffer if needed, then store 8-bit byte in buffer.
- ZTKCLS close the TK device.
- ZTKFI2 read/write from/to a Tektronix device.
- ZTKOPN open a TK device.
- ZTPCLD close pseudo-tape disk file.
- ZTPCLS closes a tape device (real or pseudo-tape disk).
- ZTPMID pseudo-tape disk read/write for 2880-bytes records.
- ZTPMIO read/write tape devices with quick return I/O methods.
- ZTPOPD open a pseudo-tape, sequential disk file for FITS.
- ZTPOPN open tape or pseudo-tape device.
- ZTPWAD "wait" for IO operation to complete on pseudo-tape disk file (ZTPMID).
- ZTPWAT wait for asynchronous IO to finish on tape or pseudo-tape disk.
- ZTTBUF reads terminal input with no prompt or wait - simulates TV trackball.
- ZTTYIO read/write buffer to terminal.

The following items are second level system routines in APLGEN; others may be needed in a given implementation.

- ZLASC2 spool a closed laser printer print/plot file.
- ZLPCL2 queue a file to the line printer and delete.
- ZLPOP2 open a line-printer text file - actual OPEN call.
- ZMOUN2 mount or dismount magnetic tape device.
- ZTAP2 position (forward/back record/file), write EOF, etc. for tapes.
- ZTKCL2 close a Tektronix device.
- ZTKOP2 read/write from/to a Tektronix device.
- ZTPCL2 close a tape device.
- ZTPMI2 tape read/write.
- ZTPOP2 open a tape device for double-buffer, asymchronous IO.
- ZTPWA2 wait for read/write from/to a tape device.
- ZTTCLS close a terminal device.
- ZTTOP2 open a message terminal.
- ZTTOPN open a terminal device.

15.5 Data Manipulation Routines

The internal form in which characters, reals and integers are stored varies from computer to computer, but a given FITS data tape should be able to be read on any AIPS system. Thus it is necessary to be able to convert between the external (FITS or other) formats and the internal formats. The format of data on FITS tape files is discussed in another chapter.

The floating point formats, word size and the byte and word order etc. on the current machine are given in commons in the DDCH.INC INCLUDE.

- BYTFLP (I) Byte flip, 0=none, 1=bytes, 2=words, 3=both
- NBITWD (I) no. bits / word
- NBITCH (I) no. bits per character
- NWDPDP (I) no. words / double-precision floating point
- SPFRMT (I) Single precision floating-point format code: 0=> OTHER, 1=> IEEE, 2=> VAX F, 3=> VAX G, 4=> IBM (not supported yet).
- DPFRMT (I) Double precision floating-point format code (see codes for SPFRMT)

The following list gives the names and uses of the upper level data manipulation "Z" routines. Details of the call sequences are given later in this chapter.

- ZBYMOV move 8-bit bytes from in-buffer to out-buffer.
- ZBYTFL interchange bytes in buffer if needed to go between local and standard.
- ZC8CL convert packed ASCII buffer to local character string.
- ZCLC8 convert local character string to packed ASCII buffer.
- ZDHPRL convert 64-bit HP floating buffer to local DOUBLE PRECISION values.
- ZDM2DL convert Modcomp REAL*6 and REAL*8 to local double precision.
- ZGETCH get a character from a HOLLERITH word.
- ZGTBIT get array of bits from a word.
- ZI16IL convert FITS-standard 16-bit integers to local integers.
- ZI32IL convert FITS-standard 32-bit integers from buffer into local integers.
- ZI8IL convert 8-bit unsigned integers in buffer to local integers.
- ZILI16 convert local integers to 16-bit FITS integers in a buffer.
- ZILI32 convert local integer into FITS-standard 32-bit integers.
- ZMCACL convert Modcomp compressed ASCII to Hollerith characters (for FILLR).
- ZPTBIT put array of bits into a word.
- ZPUTCH inserts 8-bit "character" into a word.
- ZR32RL convert 32-bit IEEE floating buffer to local REAL values.
- ZR64RL convert 64-bit IEEE floating-point buffer to local "DOUBLE PRECISION".
- ZR8P4 converts pseudo I*4 to double precision - for tape handling only.
- ZRDMF convert DEC Magtape Format (36 bits data in 40 bits) to 2 integers.

- ZRHPRL convert 32-bit HP floating buffer to local REAL values.
- ZRLR32 converts buffer of local REAL values to IEEE 32-bit floating-point.
- ZRLR64 convert buffer of local double precision values to IEEE 64-bit float.
- ZRM2RL convert Modcomp to local single precision floating point.
- ZUVPAK Pack visibility data, 1 correlator per real with magic value blank.
- ZUVXPN Expands packed visibility data and adds weight.
- ZX8XL convert FITS table bit array to AIPS bit array.
- ZXLX8 convert AIPS bit array to FITS binary table bit array.

The following item is a second level binary data conversion routine; others may be needed in a particular implementation.

- ZBYTF2 interchange bytes in buffer if needed to go between local and standard.

15.6 Directory and Text File Routines

Text files are used for source code and other information and have been discussed previously in this chapter. There are currently two systems for access to text files one for AIPS system files and the other for general text files. The following list briefly describes the functions of the first level routines for text files; detailed descriptions of the call sequences are found at the end of this chapter.

- ZTCLOS close text file opened with ZTOPEN.
- ZTOPEN open text file - logical area, version, member name as arguments.
- ZTREAD read next 80-character record in sequential text file (ZTOPEN type).
- ZTXCLS close text file opened via ZTXOPN.
- ZTXIO read/write a line to a text file.
- ZTXMAT return list of files in specified area beginning with specified chars.
- ZTXOPN open a text file for read or write.

The following items are second level text file routines in APLGEN; others may be needed in a given implementation.

- ZDIR build a full path name to files in AIPS-standard areas (HE, RU, ...).
- ZTOPE2 open text file for ZTOPEN.
- ZTXMA2 find all file names matching a given wildcard specification.
- ZTXOP2 translate the file name and open a text file.

15.7 Television I/O

System dependent routines are usually needed to communicate with a physical display device. These routines were discussed in some detail in the chapter on displays and are only brielfy described here.

- ZARGCL close an ARGS TV device.
- ZARGMC issues a master clear to an ARGS TV.
- ZARGOP open ARGS TV device.
- ZARGXF translates IIS Model 70 commands into calls to ZARGS for ARGS TV.
- ZDEACL close DeAnza TV device.
- ZDEAMC issue a master clear to the TV - for DeAnzas this is a No-Op.
- ZDEAOP opens DeAnza TV device.
- ZDEAXF do I/O to DeAnza TV.
- ZIPACK pack/unpack long integers into short integer buffer.
- ZIVSOP opens IVAS TV device - using the IIS package.
- ZM70CL close an IIS Model 70 TV device, flushing any buffer.
- ZM70MC issues a master clear to an IIS Model 70 TV.
- ZM70OP open IIS Model 70 TV device.
- ZM70XF read/write data to IIS Model 70 TV with buffering.
- ZTTBUF reads terminal input with no prompt or wait - simulates TV trackball.
- ZV20CL close a Comtal Vision 1/20 TV device.
- ZV20MC issue a master clear to the TV - for Comtal this is a No-Op.
- ZV20OP open Comtal Vision 1/20 TV device.
- ZV20XF read/write data to Comtal Vision 1/20 TV device.

The following are second level TV I/O routines in APLGEN; in a given implementation there will be others.

- ZARGC2 close an ARGS TV device.
- ZARGO2 open ARGS TV device.
- ZARGS sends command to/from the ARGS TV device.
- ZDEAC2 close DeAnza TV device.
- ZDEAO2 opens DeAnza TV device.
- ZDEAX2 do actual read/write from/to DeAnza device.
- ZM70C2 close IIS Model 70/75 TV device.
- ZM70M2 issues a master clear to an IIS Model 70 TV.
- ZM70O2 opens IIS Model 70.75 TV device.
- ZM70X2 read/write from/to IIS Model 70/75 device.
- ZV20C2 close Comtal Vision 1/20 TV device.
- ZV20O2 opens Comtal Vision 1/20 TV device.
- ZV20X2 does I/O to Comtal Vision 1/20 TV device.

15.8 Virtual Devices

In many cases it is desirable to communicate with a physical device that is directly connected to a process that may reside on another computer. These devices are refered to here as virtual devices as the communication with them is through network protocols rather than directly. These routines communicate with a generic device as they may or may not know what physical device they are communicating with. Currently, these exist only for the display device (TV).

- ZVTVCL close connection in client (virtual-TV) to server (remote, real-TV).
- ZVTVGC close and reopen connection in server (real-TV) to client (virtual-tv).
- ZVTVOP opens connection from client (virtual-TV) to server (real-TV).
- ZVTVRC closes channel in server (real-TV) to client (virtual-TV).
- ZVTVRO open scket in server (real-TV) to any client (virtual-TV).
- ZVTVRX does IO for server (real TV) to client (Virtual-TV) incl close/reopen.
- ZVTVXF sends data from the client (virtual TV) to server (real TV).

The following are second level virtual device routines in APLGEN; in a given implementation there may be others.

- ZVTVC2 close virtual TV connection to remote, real-TV computer.
- ZVTVC3 close connection in real-TV computer to client, virtual-TV computer.
- ZVTVO2 open connection in client (virtual-TV) to server (remote, real-TV).
- ZVTVO3 open connection in server (real-TV) to client (virtual-TV).
- ZVTVX2 writes/reads to/from server for the client (virtual TV) machine.
- ZVTVX3 reads/writes from/to client (virtual TV) for the server (real TV).

15.9 Miscellaneous

Several Z routines don't fit naturally into any of the above categories. The following list gives a brief description of each; details of the call sequences and functions are given at the end of this chapter.

- ZADDR determine if 2 addresses inside computer are the same.
- ZERROR prints strings associated with system error codes for Z routines.
- ZHEX encode an integer into hexadecimal characters.
- ZKDUMP display portions of an array in various Fortran formats.
- ZMSGER prints strings associated with system error codes for ZMSG routines.
- ZMYVER returns OLD, NEW, or TST based on translation of logical AIPS_VERSION.

The following item is a second level miscellaneous routine; others may be needed in a particular implementation.

- ZERRO2 return system error message for given system error code.

15.10 INCLUDEs

There are several types of INCLUDE file which are distinguished by the first character of their name. Different INCLUDE file types contain different types of Fortran declaration statements as described in the following list.

- Pxxx.INC. These INCLUDE files contain declarations for parameters and the PARAMETER statements.
- Dxxx.INC. These INCLUDE files contain Fortran type (with dimension) declarations, COMMON and EQUIVALENCE statments.
- Vxxx.INC. These contain Fortran DATA statements.
- Zxxx.INC. These INCLUDE files contain declarations which may change from one computer or installation to another.

15.10.1 DDCH.INC

```
C                                                               Include DDCH.
C                                       AIPS system parameters
      CHARACTER SYSNAM*20, VERNAM*4, RLSNAM*8, DEVNAM(10)*48,
     *   MONNAM(8)*48, MAPNAM(12)*48, SYSTYP*4, SYSVER*8
      HOLLERITH NBLANK
      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,
     *   NSNORT, TTYCAR, DEVTAB(50), FTAB(1024)
      COMMON /DCNCHM/ SYSNAM, VERNAM, SYSTYP, SYSVER, RLSNAM,
     *   DEVNAM, MONNAM, MAPNAM
      COMMON /DCNCOM/ 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,
     *   NSNORT, TTYCAR
      COMMON /FTABCM/ FTAB
C                                                               End DDCH.
```

15.10.2 DMSG.INC

```
C                                                               Include DMSG.
C                                       AIPS system message common
      INTEGER   MSGCNT, NPOPS, NLUSER, NACOUN, MSGSUP, MSGREC,
     *   MSGKIL, ISBTCN, DBGAIP, MSGDM1, MSGDM2, MSGDM3
      CHARACTER MSGTXT*80, TSKNAM*6
      COMMON /MSGCOM/ MSGCNT, NPOPS, NLUSER, NACOUN, MSGSUP, MSGREC,
     *   MSGKIL, ISBTCH, DBGAIP, MSGDM1, MSGDM2, MSGDM3
      COMMON /MSGCHR/ MSGTXT, TSKNAM
C                                                               End DMSG.
```

15.11 Routines

15.11.1 SYSTEM

The following describes the first level System "Z" routines as documented in the APLGEN directory.

ZABORT

Will take one of two actions depending on the value of "action". If "action" is zero, it establishes abort handling to clean up for programs in the event of an ABORTASK or otherwise fatal signal. Hangup, interrupt (except for AIPS or BATER) and quit are ignored. If running under the control of a debugger, it simply returns (to avoid affecting the debugger signal handling). If "action" is non-zero, it will issue an illegal instruction in order to induce the abort handler. This is mostly to get a traceback for debugging purposes (e.g., when an invalid argument to a subroutine is detected).

```
ZABORT (TSKNAM, ACTION)
Inputs:
   TSKNAM   C*6   Program name
   ACTION   I     Action indicator code:
                     0 => establish abort handling
                     1 => invoke an illegal instruction
Output:
   none
Generic version - calls ZABOR2 after tests on TSKNAM.
```

ZACTV8

Activate the specified program and return the activated process identification information.

```
ZACTV8 (NAME, INPOPS, RVERSN, PID, IERR)
Inputs:
   NAME     C*6    Program name
   INPOPS   I      POPS # for the task to use
   RVERSN   C*48   Logical name or fully qualified name of the
                   directory from which to get the required
                   executable module
In/Out:
   PID      I(4)   Process identification information used directly
                   by subsequent calls to ZTACTQ where
                   In:  PID(1)   = user number for systems that use
                                   it (= 0 otherwise and on all
                                   AIPSB invocations)
                   Out: PID(2-4) = process ID number(s) as needed
Output:
   IERR     I      Error return code: 0 => no error
                      1 => program name too long
                      2 => activation error
Common: DMSG.INC
   DBGAIP   I      > 10 => start tasks in DEBUG mode if possible
Generic version = stub
```

ZCPU

Determines cumulative cpu usage in seconds for this process: i.e. each time a process calls ZCPU during an execution, TIME is larger.

```
ZCPU (TIME, IOCNT)
Output:
   TIME    R        Current CPU accumulation in seconds
   IOCNT   I        IO count accumulation
Generic version - stub
```

ZDATE

Returns local date

```
ZDATE (LDATE)
Output:
   LDATE   I(3)   local date where
                     LDATE(1)   year since 0.
                     LDATE(2)   month (1-12).
                     LDATE(3)   day   (1-31).
Generic version - stub
```

ZDCHIN

Initialize the device characteristics common and the FCB's (file control blocks) in FTAB(*) for the maximum number of different file types that can be open at the same time. Initialize also other machine-dependent commons and the message common. Note that the task name is not set here.

```
ZDCHIN (DODISK, JOBLK)
Inputs:
   DODISK   L        Get SETPAR-controlled parameters from disk
Inputs from common: DMSG.INC
   TSKNAM   C*6      Task name if known - else ' ' (used in ABORT
                     handler mostly to separate standalones and
                     tasks)
Output:
   JOBLK    I(256)   I/O block - no longer used
Output in commons: DDCM.INC DMSG.INC
   all      ...      All values set to init except TSKNAM

ZDCHIN starts with hard-coded values.  Then, if DODISK is true,
resets those contained in the system parameter file.  The utility
program SETPAR is used to alter the system parameter file values.

Critical system constants (all "words" are local integers, all
"bytes" are AIPS-bytes, i.e., 1/2 a local integer and on 64 bit
architectures, double precision contructs should be preprocessed
into their single precision counterparts):
Generic version - calls ZDCHI2, ZDCHIC (init C codes).
```

ZDELAY

Cause an execution delay for a specified time interval.

```
ZDELAY (SECS, IERR)
Inputs:
   SECS   R   Number of seconds to delay
Output:
```

```
   IERR   I   Error return code: 0 => no error
                 1 => error
Generic version - calls ZDELA2 protecting input argument.
```

ZFREE

Determine the number of free 256-integer blocks that are available on the disks used for AIPS user data and print the information on the user's terminal.

```
ZFREE (MSLEV, IERR)
Inputs:
   MSLEV   I   Message level to use (1 => to terminal only,
                  2-5 => also to message file)
Common /DCHCOM/
   NVOL    I   Number of AIPS disks
Output:
  IERR     I   Error return code: 0 => no error
                  1 => error
Generic version - uses ZFRE2 to get info.
```

ZGNAME

Get the name of the current process.

```
ZGNAME (NAME, IERR)
Output:
   NAME   C*6   Current process name
   IERR   I     Error return code: 0 => no error
                   1 => error
Generic version - stub
```

ZPRIO

Change the execution priority of the current process between that of batch and interactive processes.

```
ZPRIO (OP, IERR)
Inputs:
   OP     C*4   'UPPP' to interactive, 'DOWN' to batch
   IERR   I     Error return code: 0 => no error
                   1 => invalid opcode
                   2 => illegal request
                   3 => other
Generic version - uses ZPRI2 to do the real work.
```

ZSETUP

Performs any needed system level operations which can be performed only after VERNAM, TSKNAM, and NPOPS have been determined.

```
ZSETUP
(no call arguments)
Generic version  - only lowers priority of batch jobs from that of
interactive.  If batch are started at the interactive priority,
then they can usually go back to interactive when needed such as
while they possess a valuable device (e.g., a real AP).
```

ZSTAIP

Performs any operations needed to normalize the local operating system at the conclusion of an interactive AIPS session.

```
ZSTAIP (SCRTCH)
Outputs:
   SCRTCH   I(256)   Scratch buffer

Generic version  - calls ZSTAI2 in case there is something to do
in C or Macro, else a null routine.
```

ZTACTQ

Determines if a specified task is active.

```
ZTACTQ (NAME, PID, ACTIVE, IERR)
Inputs:
   NAME     C*6    Actual task name.
In/out:
   PID      I(4)   Process "ID" code
                   In:  PID(1) user number  0 => any
                        PID(2-4) process #  0 => unknown
                   Out: PID(1) not changed
                        PID(2-4) the ID determined here from NAME
Output:
   ACTIVE   L      Task active indicator (T => active)
   IERR     I      Error return code: 0 => ok.
                      2 => invalid task name.
                      3 => other
Generic version - calls ZTACT2.
```

ZTIME

returns the local time of day

```
ZTIME (ITIME)
Output:
   ITIME   I*2(3)   Local time where: ITIME(1) = hour (0-24)
                                      ITIME(2) = minute (0-60)
                                      ITIME(3) = second (0-60)
Generic version - stub
```

ZTKILL

Will delete/kill the process specified by NAME.

```
ZTKILL (NAME, PID, IERR)
Inputs:
   NAME   C*6    actual task name
   PID    I(4)   Process "ID" code:
                    PID(1) user number  0 => any user
                    PID(2-4) process #  0 => unknown
Output:
   IERR   I      Error number: 0 => ok.
                    1 => error.
Generic version - stubbed pending development.
```

ZTQSPY

Displays information on the user's terminal regarding AIPS account originated processes or all processes running on the system.

```
ZTQSPY (DOALL, TLIST)
Inputs:
   DOALL   R        > 0.0 => display all processes
Output:
   TLIST   I(256)   Scratch buffer (not used in generic)
Generic version - uses ZTQSP2.
```

ZTRLOG

Translate a logical name (i.e., environment variable). NOTE: This routine is ONLY for use by other Z-routines.

```
 ZTRLOG (LLEN, LOGNAM, XLEN, XLATED, XLNB, IERR)
Inputs:
   LLEN     I          Length of LOGNAM (1-relative)
   LOGNAM   C*(LLEN)   Logical name
   XLEN     I          Length of XLATED (1-relative)
Output:
   XLATED   C*(XLEN)   Translation (blank filled)
   XLNB     I          Position of last non-blank in XLATED
                       (1-relative)
   IERR     I          Error return code: 0 => no error
                           1 => error
Generic version - stub
```

ZWHOMI

Determines the actual task name under which the present version of AIPS is running. It uses this information to set the value of NPOPS in the common /MSGCOM/. It then assigns the TV and TK devices setting NTVDEV and NTKDEV in include DDCH.INC.

```
ZWHOMI (IERR)
Output:
   IERR   I   Error return code: 0 => no error
                 1 => process name is not AIPSx
                 2 => illegal POPS number
                 3 => other error
Generic version - uses ZGNAME, ZTRLOG
```

15.11.2 Disk I/O

The following describes the first level Disk I/O "Z" routines as documented in the APLGEN directory.

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
Generic version.
No longer closes TV devices as of the 15OCT87 release.
No longer closes tape devices as of the 15APR87 release.
No longer closes Tektronix devices as of the 15OCT87 release.
No longer closes text files as of the 15OCT87 release.
```

ZCMPRS

Releases unused disk space from the end of an open disk file. "Byte" 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
Generic version - uses ZCMPR2.
```

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)
Generic version - uses ZCREA2
```

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
Generic version.
```

ZEXIST

Determine if a file exists and if so, return its size in AIPS-bytes (1/2 of a local integer).

```
ZEXIST (IVOL, PNAME, ISIZE, SCRTCH, IERR)
Inputs:
   IVOL     I      Disk volume containing file, 1,2,3,...
   PNAME    C*48   Physical file name
Output:
   ISIZE    I      File size in AIPS-bytes (1/2 of a local integer)
   SCRTCH   I(*)   I/O buffer
   IERR     I      Error return code: 0 => file exists
                      1 => file does not exist
                      2 => inputs error
                      3 => other error
Generic version - SCRTCH not used, calls ZEXIS2.
```

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
Generic version - uses ZEXPN2.
```

ZFIO

Transfer one logical record between an I/O buffer and device LUN. For disk devices, the record length is always 256 local 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
Generic version.
No longer performs I/O to TV devices as of the 15MAR84 release.
No longer performs I/O to tape devices as of the 15APR87 release.
Only performs disk and Tektronix device I/O now.
```

ZFULLN

Convert file name from user specified name with or without a logical to a full path and file name with the logical translated.

```
ZFULLN (UNAME, DEFLOG, SUBR, FNAME, IERR)
Inputs:
   UNAME     C*(*)     User specified name - first a logical name,
                       then a colon, then a file name (with or
                       without a "." extension).  Logical optional.
                       ' ' => create temporary file (DEFLOG not ' ')
   DEFLOG    C*(*)     Default logical: ' ' => no default
                       '-1' => UNAME must provide a logical
   SUBR      C*6       Subroutine name to be used in creating a
                       "temporary" file name.  ' ' => no temporary
Output:
   FNAME     C*(*)     Full file name
   IERR      I         Error code: 0 => okay
                          1 => improper combination of inputs
                          2 => no translation for logical
                          3 => FNAME too short
                          4 => logical required, not provided
Generic version - may not need to make any machine-specific versions
unless there is something really special/odd about logical names.
```

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
Generic version - uses ZMI2.
No longer performs I/O to TV devices (15MAR84), to tape devices
(15APR87), and to Tektronix devices (15JUL87) release.
```

ZMKTMP

Form a unique, fully qualified, temporary file name.

```
ZMKTMP (FLEN, FILNAM, IERR)
Inputs:
   FLEN     I       Length of "filnam"
In/Out:
   FILNAM   C*(*)   File name with the extension .XXXXXX, (e.g.,
                    "ZXLPRT.XXXXXX"). The extension will be
                    transformed by 'mktemp' to make "filnam" a
                    unique file name.
Output:
   IERR     I       Error return code: 0 => no error
                       1 => inputs wrong
                       3 => filnam too short
Generic version - stub
```

ZMSGCL

Close a message file associated with LUN removing any exclusive use state and clear the FTAB entry for the LUN (much like ZCLOSE but does not call MSGWRT to avoid recursion).

```
ZMSGCL (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
Generic version - uses ZDACLS and ZTTCLS.
```

ZMSGDK

Transfer one 256-integer record to/from message disk file. Like ZFIO, but does not call MSGWRT to avoid recursion.

```
ZMSGDK (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)
                    In units of 256-integer records.
In/out:
   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
Generic version - calls ZFI2
```

ZMSGOP

Open a message file (much like ZOPEN, but does not call MSGWRT to avoid recursion). Non-map, exclusive, patient assumed on open.

```
ZMSGOP (LUN, FIND, IVOL, PNAME, IERR)
Inputs:
   LUN      I      Logical unit number
   IVOL     I      Disk volume containing file, 1,2,3,...
   PNAME    C*48   Physical file name
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 not found
                      4 => exclusive use denied
                      5 => no room for LUN in FTAB
                      6 => other open errors
Generic version - calls ZDAOPN, ZTTOPN, ZMSGER.
```

ZMSGXP

Increase the size of a message file (special version of ZEXPND that writes error messages to the terminal only in order to avoid recursion).

```
ZMSGXP (LUN, IVOL, PNAME, NREC, IERR)
Inputs:
   LUN      I      LUN of file open message file (must be 12)
   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 => ZEXIS2 error
Generic version - calls ZEXPN2, ZEXIS2.
```

ZOPEN

Open a binary disk file, line printer or terminal 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)
                    For line printers (LUN=1), the output text file
                    to be kept (' ' => use a scratch)
   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
Generic version - uses ZDAOPN, ZLPOPN, ZTTOPN
No longer opens TV devices as of the 15MAR84 release.
No longer opens tape devices as of the 15APR87 release.
No longer opens Tektronix devices as of the 15OCT87 release.
No longer opens text files as of the 15OCT87 release.
```

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

Generic version - the ZOPEN, ZTRLOG, etc. routines interpret the
resulting VMS-like names.
```

ZPHOLV

Construct a physical file name in PNAM from TYPE, IVOL, NSEQ, and IVER - either for public data files or user-specific files. Version of ZPHFIL for UPDAT, in which the Format version codes are passed as arguments.

```
ZPHOLV (VERDAT, VERSYS, TYPE, IVOL, NSEQ, IVER, PNAM, IERR)
Inputs:
   VERDAT  C*1   Data file version code letter
   VERSYS  C*1   System file version code letter
   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

Generic version - the ZOPEN, ZTRLOG, etc. routines interpret the
resulting VMS-like names.
```

ZRENAM

Rename a disk file.

```
ZRENAM (IVOL, OLDNAM, NEWNAM, IERR)
Inputs:
   IVOL     I      Disk volume containing files, 1,2,3,...
   OLDNAM   C*48   Old physical file name
```

```
   NEWNAM   C*48   New physical file name
Output:
   IERR     I      Error return code: 0 => ok
                      2 => old file not found
                      3 => volume not found
                      4 => old file busy
                      6 => new file name already exists
                      7 => other
Generic version.
```

ZTFILL

Fills in initial values in the FTAB file control block.

```
ZTFILL (FIND, MAP)
Inputs:
   FIND   I   Index in FTAB to file control block
   MAP    L   Map file indicator
Common: via DDCH.INC
   FTAB   I(*)   input:  FTAB(FIND) = LUN
                 output: FTAB(FIND+1...) = ? whatever is needed
Generic version - zeros the table.
```

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
Generic version: uses ZWAI2
```

15.11.3 Non-disk I/O routines

The following describes the first level non–disk I/O "Z" routines as documented in the APLGEN directory.

ZBKLD1

Routine to initialize the host environment in preparation for execution of ZBKLD2 under task BAKLD.

```
ZBKLD1 (IERR)
Output:
   IERR   I   error code : 0 => okay
                 1 => can't create listing subdirectory
                 2 => input error - translates fail
Generic version - stub
```

ZBKLD2

Host-dependent routine to process input tape for task BAKLD. The input tape is presumed to have been produced by task BAKTP executing on the same host/OS combination. Data format is the hosts 'backup' utility (BACKUP on VMS, 'tar' on Unix).

```
ZBKLD2 (OP, IERR)
Inputs:
   OP    C*4   'SKIP' skips over a saveset.
               'PRNT' moves over a saveset, listing directory info.
               'LOAD' loads a saveset.
Output:
   IERR  I     Error return
Generic version - stub.
```

ZBKLD3

Routine to clean up host environment after executing task BAKLD.

```
ZBKLD3 (IERR)
Output:
   IERR   I   Error code: 0 => okay
                  1 => can't open command file
Generic version - stub.
```

ZBKTP1

Initialize the host environment in preparation for ZBKTP2. If OPCODE = 'INIT', initializes the tape too. (used only by task BAKTP)

```
ZBKTP1 (ZLUN, ZBKNAM, BAKTXT, NTAPE, ZMTON, LZMTON, IERR)
Inputs:
   NTAPE    I       Tape drive number
   BAKTXT   C*(*)   lowest file name for file listing (not used
                    here, only in VMS now)
Outputs:
   ZLUN     I       LUN to use
   ZBKNAM   C*(*)   Command file name
   ZMTON    C*(*)   Tape name (translation of MTOn) n=NTAPE-1
   LZMTON   I       Actual length used in ZMTON
   IERR     I       Error code: 0 => okay
Generic stub
```

ZBKTP2

Host-dependent subroutine to write a file and its extensions to tape using the hosts "backup" utility.

```
ZBKTP2 (ZLUN, ZBKNAM, BAKTXT, BAKREC, IVOL, ICNO, NTAPE, ZMTON,
  *   LZMTON, IERR)
Inputs:
   ZLUN     I       LUN for command file
   ZBKNAM   C*(*)   Name of command file
   BAKTXT   C*(*)   lowest file name for file listing
   BAKREC   C*60    ?
   IVOL     I       Disk number
```

```
   ICNO     I       Catalog number
   NTAPE    I       Tape drive number
   ZMTON    C*(*)   Tape name (translation of MTOn) n=NTAPE-1
   LZMTON   I       Actual length used in ZMTON
Output:
   IERR     I       Error code: 0 => okay, else ZSHCMD
Generic version - stub
```

ZBKTP3

Clean up host environment after BAKTP execution.

```
ZBKTP3 (ZLUN, ZBKNAM, IERR)
Inputs:
   ZLUH     I       LUN for command file
   ZBKNAM   C*(*)   Name of command file
Output:
   IERR     I       Error code: 0 => okay, else ZSHCMD
Generic version - stub
```

ZDOPRT

Read a bit map such as produced by PRTDRW and convert it into a file that can be spooled to a Versatec printer/plotter.

```
ZDOPRT (IVOL, LUN, NCOPY, PNAME, ISIZE, INBLK, IERR)
Inputs:
   IVOL     I      Disk volume containing file, 1,2,3,...
   LUN      I      Logical unit number
   NCOPY    I      Number of copies of the plot to make
   PNAME    C*48   Physical file name (left justified)
   ISIZE    I      Size of INBLK in words
In/Out:
   INBLK    I(*)   Scratch buffer
Output:
   IERR     I      Error return code: 0 => no error
                      1 => error
Generic version - a stub
```

ZENDPG

Advance the line printer to avoid "burn-out" on electro-static type devices.

```
ZENDPG (LINE)
Inputs:
   LINE     I          # lines printed on page so far
Generic version, does a partial page.
```

ZLASCL

Close and spool a file for printing a plot on a laser printer.

```
ZLASCL (FILNAM, LUN, DELFIL, SYSERR, IERR)
Inputs:
   FILNAM   C*48   print/plot file name
   LUN      I      LUN under which file is open
   DELFIL   I      1 => delete the file after print, 0 => keep it
Output
   SYSERR   I      system error code (on IERR 2)
   IERR     I      error code: 0 => okay, 1 => error in queue,
                      2 => Fortran close error
Generic version - calls ZLASC2 for any special stuff after a Fortran
close.
```

ZLASIO

Open, write to or close and spool a file for printing a plot on a laser printer.

```
ZLASIO (OP, LUN, OUTFIL, NCHAR, CBUFF, IERR)
Inputs:
   OP       C*4    Operation code: 'OPEN', 'POPN', 'WRIT' or 'CLOS'
   LUN      I      Logical unit number for the laser printer
   OUTFIL   C*48   Output file name (used by opcode 'OPEN', 'POPN',
                   'CLOS')
   NCHAR    I      Number of characters to print in CBUFF
                   (used by opcode 'WRIT' only)
   CBUFF    C*(*)  I/O buffer (used by opcode 'WRIT' only)
Output:
   IERR     I      Error return code: 0 => no error
                      1 => invalid opcode
                      2 => trouble translating logical
                      3 => I/O error
Generic version - uses ZFULLN, ZLASOP, ZLASCL
```

ZLPCLS

Close a line printer file, spool it to a printer and delete it.

```
ZLPCLS (LUN, IERR)
Inputs:
   LUN      I   Logical unit number
Output:
   IERR     I   Error return code: 0 => no error
                   1 => close error
Generic version - closes file, calls ZLPCL2
```

ZLPOPN

Open a line printer file.

```
ZLPOPN (LPFILE, IERR)
Inputs:
   LPFIL    C*48   File name to use for line printer file
Output:
   IERR     I      Error return code: 0 => no error
                      1 => error, 2 => LPFILE already exists
Generic version - uses ZFULLN, ZLPOP2.
```

ZLWIO

Open a temporary file for printing a plot on a PostScript printer using the using the name ZLWIO.XXXXXX where XXXXXX is a unique extension (OP = 'OPEN'), write data to the temporary file (OP = 'WRIT'), or close, print and delete the file (OP='CLOS').

```
ZLWIO (OP, LUN, NCHAR, CBUFF, IERR)
Inputs:
   OP      C*4     Operation code ('OPEN', 'WRIT' or 'CLOS')
   LUN     I       Logical unit number
   NCHAR   I       Number of characters to print ('WRIT' only)
   CBUFF   C*(*)   I/O buffer ('WRIT' only)
Output:
   IERR    I       Error return code: 0 => no error
                      1 => input error
                      3 => no such logical device or
                           forming temporary file name
                      6 => I/O error
Generic version - calls ZFULLN, ZLWOP, ZLASCL
```

ZLWOP

Open a file for printing a plot on a QMS Lasergraphix device.

```
ZLWOP (OP, LUN, FILNAM, IERR)
Inputs:
   OP       C*4     Operation code: 'OPEN', 'POPN'
   LUN      I       Logical unit number for the QMS device
   FILNAM   C*(*)   File name
Output
   IERR     I       Error code: 0 => okay
Generic version - simple Fortran open.
```

ZMOUNT

Issue software mount or dismount for a given tape drive.

```
ZMOUNT (MOUNT, IDRIVE, IDENS, IERR)
Inputs:
   MOUNT    L   .TRUE. means mount, .FALSE. means dismount
   IDRIVE   I   Tape drive number
   IDENS    I   Density at which to mount tape (800, 1600 or 6250)
Output:
   IERR     I   Error return code: 0 => no error
                   1 => error
Generic version - calls ZMOUN2 to do real work
```

ZPRMPT

Prompts user on CRT screen and reads a line.

```
ZPRMPT (IPC, BUFF, IERR)
Input:
   IPC     I      prompt character.
Output:
```

```
   BUFF    C*80   line of user input.
   IERR    I      error code:   0 => ok.
                     1 => read/write error.
Generic version - stub
```

ZPRPAS

Prompts the user on his terminal with the prompt string "Password:" and then reads back a 12-character "password" without anything being visible on the screen.

```
ZPRPAS (PASS, BUFF, IERR)
Outputs:
   PASS   C*12     Password - 12 unpacked characters: left
                              justigied and blank filled.
   BUFF   I(256)   Scratch buffer (if needed)
   IERR   I        Error code: 0 => ok
                      ??? => I/O error of some sort
Generic version - stub
```

ZTAPE

Performs standard tape manipulating functions.

```
ZTAPE (OP, LUN, FIND, COUNT, IERR)
Inputs:
    OP      C*4   Operation to be performed.
                  'ADVF' = advance file marks
                  'ADVR' = advance records
                  'BAKF' = backspace file marks.
                  'BAKR' = backspace records.
                  'DMNT' = dismount tape.
                  'MONT' = mount tape.
                  'REWI' = rewind the tape on unit LUN
                  'WEOF' = write end of file on unit LUN: writes 4
                           EOFs, positions tape after first one
                  'MEOF' = write 4 EOF marks on tape, position tape
                           before the first one
    LUN     I     logical unit number
    FIND    I     FTAB pointer.  Drive number for MOUNT/DISMOUNT.
    COUNT   I     Number of records or file marks to skip. On MOUNT
                  this value is the density.
Outputs:
    IERR  I       Error return: 0 => ok
                     1 = File not open
                     2 = Input specification error.
                     3 = I/O error.
                     4 = End Of File
                     5 = Beginning Of Medium
                     6 = End Of Medium
Generic version - uses ZTAP2 and ZMOUNT.
```

ZTAPIO

Tape operations for IMPFIT (compressed FITS transport tape).

```
ZTAPIO (OPER, LN, NAME, FD, BYTREQ, BYTRED, BUF,
  *   SYSERR)
Inputs:
   OPER     C*4    'OPEN', 'READ', 'CLOS'
   LN       I      length of name
   NAME     C*(*)  physical file name
   BYTREQ   I      bytes to be read
In/out:
   FD       I(*)   file descriptor (set on OPEN, else used)
Output:
   BYTRED   I      Bytes read on READ
   BUF      I(*)   Data buffer read
   SYSERR   I      System error code
Generic version - stub
```

ZTKBUF

Will flush the Tektronix output buffer TKBUFF, if necessary, then store the low order 8-bit byte of IN into the proper 8-bit byte of TKBUFF. This is a Z-routine to allow for any required local conversions.

```
ZTKBUF (IN, IT, FIND, IERR)
Inputs:
   IN       I           Word from which to extract the low order
                        byte and store in TKBUF
   IT       I           Data type indicator:
                           1 => control
                           2 => position
                           3 => char
   FIND     I           Index in FTAB to file control block for
                        Tektronix device LUN
Output:
   IERR     I           Error return code:
                           0 => no error
                         > 1 => write error (from TEKFLS)
Common: (DTKS.INC)
   TKSIZE   I           Size of TKBUFF in floating point words
   TKBUFF   R(TKBUFF)   Tektronix output buffer
   TKPOS    I           Current byte position in TKBUFF
Generic version.
```

ZTKCLS

Close a Tektronix device

```
ZTKCLS (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 => Non-zero ZTKCL2 error
                   2 => Non-zero LSERCN error
                   3 => both 1 and 2
                   4 => invalid LUN
Generic version -  calls ZTKCL2 to perform the actual close.
```

ZTKFI2

Read/write "nbytes" of data from/to a Tektronix terminal.

```
ZTKFI2 (OPER, FCB, BUFF, NBYTES, IERR)
Inputs:
   OPER     C*4    Operation code "READ" or "WRIT"
   FCB      I(*)   File control block for opened Tektronix
   NBYTES   I      Number of 8-bit bytes to be transferred
In/out:
   BUFF     I(*)   I/O buffer
Output:
   IERR     I      Error return code: 0 => no error
                      2 => bad opcode
                      3 => I/O error
                      4 => end of file
Generic version - stub
```

ZTKOPN

Open a Tektronix device (calls ZTKOP2 to perform the actual open).

```
ZTKOPN (LUN, FIND, IERR)
Inputs:
   LUN    I   Logical unit number
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 => no such logical device
                 3 => device not found
                 4 => exclusive use denied
                 5 => no room for LUN in FTAB
                 6 => other open errors
Generic version.
```

ZTPCLS

Close the tape drive associated with LUN as well as its disk control file removing any exclusive use state and clear the corresponding FTAB entries. ZTPCL2 actually closes the tape drive and ZDACLS is called to close the disk control file. Also closes sequential type disk files via ZTPCLD.

```
ZTPCLS (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 => non-zero LSERCH error
                 3 => both 1 and 2
                 4 => invalid LUN
Generic version.
```

ZTPMIO

Low level sequential access, large record, double buffered tape device I/O.

```
ZTPMIO (OPER, LUN, FIND, 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
   NBYTES   I      Number of 8-bit bytes to transfer
   BUFF     I(*)   I/O buffer
   IBUFF    I      Buffer number to use (1 or 2)
Output:
   IERR     I      Error return code: 0 => no error
                      1 => file not open
                      2 => input error
                      3 => I/O error
                      4 => end of file (no messages)
Generic version.
```

ZTPOPN

Open a tape drive (as well as its corresponding disk control file) for sequential, "map" (double buffered, asynchronous) I/O or open a pseudo-tape sequential disk file. Exclusive use and wait to open are assumed. Uses a 'TP' disk "lock" file for real tapes.

```
ZTPOPN (LUN, FIND, IVOL, PNAME, OPER, IERR)
Inputs:
   LUN      I       Logical unit number (30 < LUN <= 30 + NTAPED
                       => tape, else disk)
   IVOL     I       Tape drive or disk volume containing file
   PNAME    C*48    tape disk physical file name
   OPER     C*4     'READ' => read only or 'WRIT' => read/write
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 not found
                       4 => exclusive use denied
                       5 => no room for LUN in FTAB
                       6 => other open errors
Generic version.
```

ZTPWAT

Wait until an asynchronous tape or sequential pseudo-tape disk file I/O operation completes.

```
ZTPWAT (LUN, FIND, IBUFF, LBYTES, 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:
   LBYTES   I   Number 8-bit bytes read/written (+1 if tape tape
```

```
                record longer than requested)
   IERR     I   Error return code: 0 => no error
                   1 => LUN not open in FTAB
                   3 => I/O error
                   4 => end of file
                   7 => wait service error
Generic version - calls ZTPWA2, ZTPWAD, ZERROR.
```

ZTTBUF

Dumps the contents (if any) of the input buffer into the array BUF without issuing a prompt or waiting for input. It does the terminal open, a call to some read with timeout set to 0, and the close.

```
ZTTBUF (BUF, BYTCNT, SYSERR, IERR)
Outputs:
   BUF      C*(*)   Input buffer
   BYTCNT   I       Number of bytes (characters) read
   IOSB     I       System error code
   IERR     I       Returned error code:  0 = OK
                       2 = no access to terminal
                       3 = IO error
                       8 = channel not deassigned
This routine was written to simulate trackball buttons on
the keyboard.    Andy Lubenow   22 Nov 1983
Generic version - stub
```

ZTTYIO

Perform I/O to a terminal.

```
ZTTYIO (OPER, LUN, FIND, NCHARS, 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
   NCHARS   I     # characters to transfer (<= 132)
In/out:
   BUFF     C*(*) I/O buffer containing characters (1-256)
Output:
   IERR     I     Error return code: 0 => no error
                     1 => file not open
                     2 => input error
                     3 => I/O error
                     4 => end of file
Generic version: formatted IO does carriage control on output based
on parameter TTYCAR.
```

15.11.4 Data Manipulation

The following describes the first level data manipulation "Z" routines as documented in the APLGEN directory.

ZBYMOV

Moves a string of bytes from INB to OUTB. Byte = 8 bits (NOT an AIPS half-integer byte). This is used with tape input and output only.

```
ZBYMOV (NMOVE, INP, INB, OUTP, OUTB)
Inputs:
   NMOVE   I      Number of 8-bit bytes bytes
   INP     I      First value in INB to move (8-bit byte offset)
   INB     I(*)   Input buffer.
   OUTP    I      Location in OUTB to put first 8-bit byte  (8-bit
                  byte offset)
Output:
   OUTB    I(*)   Output buffer.
Generic version - stub.
```

ZBYTFL

Interchange the low order and high order bytes for all words in the input buffer and put the results in output buffer (which may be the same as the input buffer). For machines that are not byte flipped, the output buffer is identical to the input buffer (see BYTFLP in ZDCHIN). Any required byte swapping is performed via a call to ZBYTF2. NOTE a byte is 8 bits here.

```
ZBYTFL (NWORDS, INBUF, OUTBUF)
Inputs:
   NWORDS   I          Number of 16-BIT words to byte swap
   INBUF    I*2(*)     Input buffer
Output:
   OUTBUF   I*2(*)     Output buffer containing the byte swapped
                       words
Generic version - uses ZBYTF2 and ZADDR, works for 16, 32, 48, and
64-bit machines only
```

ZC8CL

Convert 8-bit ASCII standard characters in a buffer to local character form.

```
ZC8CL (NCHAR, NP, INBUF, OUTBUF)
Inputs:
   NCHAR    I       Number of characters to convert
   NP       I       Starting position in input buffer in units of
                    8-bit characters (1-relative)
   INBUF    R(*)    Input buffer containing 8-bit ASCII characters
Output:
   OUTBUF   C*(*)   Output buffer containing characters in local
                    form beginning in position 1
Generic version - assumes local characters are ASCII.
Requires local development if NBITWD not n*8.
```

ZCLC8

Convert local characters in a buffer to standard 8-bit ASCII character form.

```
ZCLC8 (NCHAR, INBUF, NP, OUTBUF)
Inputs:
   NCHAR    I       Number of characters to convert
   INBUF    C*(*)   Characters in local form
   NP       I       Starting position in output buffer in units of
                    8-bit characters (1-relative)
Output:
   OUTBUF   R(*)    Buffer containing characters in 8-bit ASCII form
Generic version - assumes local characters are ASCII.
Requires local development if NBITWD not n*8.
```

ZDHPRL

Converts from 64 bit Hewitt-Packard floating format to local double precision.

```
HP R*8 format:
          1         2         3         4         5         6
01234567890123456789012345678901234567890123456789012345678901234567890123
smmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmnnnnnnnnnnnnnnnnnnnnnnnnnnnneeeeeeez
The value can be determined as follows:
   man1 = smmmmmmmmmmmmmmmmmmmmmmmmmmmmm is a two's complement
          signed integer.
   man2 = nnnnnnnnnnnnnnnnnnnnnnnn is a POSITIVE unsigned integer.
   exp = zeeeeeee is a two's complement signed integer.
   value = (man1 * 16777216 + man2) * 2.0D0 ** (exp - 55)

   ZDHPRL (NVAL, NP, INB, OUTB)
   Inputs:
      NVAL   I      Number of values to convert
      NP     I      First value in INB to convert
      INB    D(*)   64-bit HP format values
   Output:
      OUTB   D(*)   Local format values.
   Generic version; should work on any machine.
```

ZDM2DL

Convert Modcomp R*6 (zero padded to D) or R*8 data into local double precision.

```
   ZDM2DL (NWORDS, INBUF, OUTBUF)
   Inputs:
      NWORDS   I      Length of the input buffer in words
      INBUF    D(*)   Input buffer containing Modcomp R*6 (zero padded
                      to D) or D*8 data
   Output:
      OUTBUF   D(*)  Output buffer containing local D.P. data
   Notes:
   (1) The INBUF should have its bytes flipped by a call to ZBYTFL
       which will leave the values split between 2 local integers.
       Do not call ZI32IL which may swith 16-bit words, or ZI16I1
       which will expand each 16-bit part to 32-bits.
   (2) First 32 bits:
         Expects, after word flip on VMS only, a sign bit in bit 31
         (1=>negative), bits 22:30 are the  exponent biased by 512,
```

```
        bits 0:21 are the normalized fraction.  Negative values are
        obtained by 2's complement of the whole word.
      Second 32 bits:
        Just extended precision bits.
   (3) Should work inplace.
   Generic version --- stubbed.
```

ZGETCH

Extracts the character in position NCHAR of the HOLLERITH argument WORD and inserts it in the least significant bits of the INTEGER argument ICHAR with zero in the rest. It should also work for INTEGER WORD as long as NCHAR is valid. Characters are numbered from 1 in the order in which they would be printed by a Fortran "A" format specifier. NOTE - we actually get 8 bits here - so this routine works for bytes too.

```
   ZGETCH (ICHAR, WORD, NCHAR)
   Inputs:
      WORD   R   Word from which the character is to be extracted
      NCHAR  I   Position of character to extract
   Output:
      ICHAR  I   Extracted character in LS bits, zero in the rest
   Generic version - stub
```

ZGTBIT

Get the lowest order "nbits" bits of the bit pattern in "word" and return them in the array BITS with the lsb in bits[0]. For example, if

```
word  = 0 0 0 0 0 1 0 1 ... 0 0 0 1 0 0 1 1
        ^                               ^
       MSB                             LSB
and nbits = 3 then bits[0] = 1 , bits[1] = 1 and bits[2] = 0

   ZGTBIT (NBITS, WORD, BITS)
   Inputs:
      NBITS   I      Number of bits
      WORD    I      Word from which to extract bits
   Output:
      BITS    I(*)   Bit array (values 0 or 1)
   Generic version - slow.
```

ZI16IL

Extract 16-bit, 2's complement integers from an input buffer and put them into an output buffer in local integer form. This must work even when the address of the input and output buffers is the same.

```
   ZI16IL (NVAL, NP, INB, OUTB)
   Inputs:
      NVAL   I         Number of 16-bit integers to extract
      NP     I         Starting position in the input buffer counting
                       from 1 in units of 16-bit integers
      INB    I*2(*)    Input buffer
   Output:
      OUTB   I(NVAL)   Output buffer
   Generic version - assumes that the local machine is 32-bit with a
   valid INTEGER*2 Fortran data type
```

ZI32IL

Extract 32-bit, 2's complement integers from an input buffer and put them into an output buffer in local large integer form. This must work even when the address of the input and output buffers is the same. The IBM order applies to the input (i.e., the most significant part of the 32-bit integer is in the lower index of the input buffer and the least significant part is in the higher index.

```
ZI32IL (NVAL, NP, INB, OUTB)
Inputs:
   NVAL   I      # values to extract
   NP     I      Starting position in the input buffer (1-relative)
                 in units of 32-bit integers
   INB    I(*)   Input buffer
Output:
   OUTB   I(*)   Output buffer
Generic version  - does byte order flip in 16-bit words and a flip
of the order of the words.  This assumes local 32-bit integers, but
the capability of INTEGER*2 as well.
```

ZI8IL

Convert 8-bit unsigned binary numbers to local integers. This must work even when the input and output buffers are the same.

```
ZI8IL (NVAL, NP, INB, OUTB)
Inputs:
   NVAL   I         Number of 8-bit values to convert
   NP     I         Starting position in the input buffer counting
                    from 1 in units of 8-bit values
   INB    I(*)      Input buffer
Output:
   OUTB   I(NVAL)   Output buffer
Generic version - for NBITWD = n*8 uses ZGETCH, else is stubbed
```

ZILI16

Convert a buffer of local integers to a buffer of standard 16-bit, 2's complement integers.

```
ZILI16 (NINT, INB, NP, OUTB)
Inputs:
   NINT   I      Number of integers to convert
   INB    I(*)   Input buffer (start at index 1)
   NP     I      Starting index in the output buffer (1-relative)
                 in units of 16-bit integers
Output:
   OUTB   I(*)   Output buffer
Generic version - assumes that the local machine is 32-bit with a
valid INTEGER*2 Fortran data type
```

ZILI32

Convert a buffer of local large integers to a buffer of standard 32-bit, 2's complement integers. This must work even when the address of the input and output buffers is the same. The IBM order applies to the output (i.e., the most significant part of the 32-bit integer is in the lower index of the output buffer and the least significant part is in the higher index).

```
ZILI32 (NVAL, INB, NP, OUTB)
Inputs:
   NVAL   I         # integers to convert
   INB    I(*)      Input buffer (start at index 1)
   NP     I         Starting position in the output buffer
                    (1-relative) in units of 32-bit integers
Output:
   OUTB   I(NVAL)   Output buffer
Generic version  - does byte order flip in 16-bit words and a flip
of the order of the words.  This assumes local 32-bit integers, but
the capability of INTEGER*2 as well.
```

ZMCACL

Convert Modcomp compressed ASCII characters to local uncompressed characters. Successive calls will append data to the end of the output buffer. One compressed record is processed per call.

NOTE: this routine will not work in place.

Modcomp compressed ASCII format for each logical record:

```
BYTE              Use
 0      ASCII ETX (Hex 03)
 1      checksum (optional)
2-3     byte count, negative => end of file (not on tape?)
        (NOTE: may be bytes 1-2)
4-      Compressed ASCII characters.  A NULL (Hex 00) terminates.
        A negative value (most significant bit on) indicates a
        repetition of the previous character the number of times
        indicated by the absolute value of the negative number.
        Example: an ASCII 'C' followed by a byte with the HEX
        value FF (2's complement -1) indicates two 'C's.
ZMCACL (NBYTES, INBUF, OUTBUF, LASTCH)

Inputs:
   INBUF    I(*)        Input buffer of Modcomp compressed ASCII
                        packed characters
In/out:
   NBYTES   I           (In)  Maximum number of bytes to convert
                        (Out) Number of bytes added to the output
                              buffer
   LASTCH   I           Position of the next character in the output
                        buffer (zero before first call)
Output:
   OUTBUF   I(*)        Output buffer containing packed characters
                        (due to the expansion of the data, the size
                        of the output buffer is not strictly
                        predictable)
Generic version (stubbed).
```

ZPTBIT

Build WORD from NBITS bit values contained in the array BITS, where BITS[1] supplies the lsb, BITS[2] the next higher bit, etc. The rest of the bits in WORD are set to zero. For example, if

```
bits[1:*] = 0 1 1 1 0 1 0 1 ... 0 1 1 1 0 0 0 0
            ^                                 ^
          LSB                                 MSB
and NBITS = 4, then WORD = 14 (decimal)
```

```
ZPTBIT (NBITS, WORD, BITS)
Inputs:
   NBITS   I      Number of bits to use from the array "bits"
   BITS    I(*)   Array of bit values (0 or 1)
Output:
   WORD    I      Result containing bit pattern from the first
                  "nbits" values of the array "bits"
Generic version - stub
```

ZPUTCH

Inserts the character contained in the least significant bits of the INTEGER argument ICHAR into the NCHAR position of the HOLLERITH argument WORD. It should also work for INTEGER WORD as long as NCHAR is valid. Characters are numbered from 1 in the order in which they would be printed by a Fortran "A" format specifier.

```
ZPUTCH (ICHAR, WORD, NCHAR)
Inputs:
   ICHAR   I   Character to insert in LS bits
   NCHAR   I   Position in WORD to store character
Output:
   WORD    H   Word into which character is to be inserted
Generic version - stub
```

ZR32RL

Converts from 32 bit IEEE floating format to local single precision.

```
The IEEE format is:

           1         2         3
   01234567890123456789012345678901
   seeeeeeeemmmmmmmmmmmmmmmmmmmmmmm

where sign = -1 ** s, exponent = eee..., mantissa = 1.mmmmm...

The value is given by:

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

Note: these values have a "hidden" bit and must always be normalized The IEEE nan (not a number) values are used to indicate an invalid number; a value with sign and all exponent bits set is recognized as "nan".

The AIPS internal format for an invalid number is the value which has the same bit pattern as 'INDE'.

The IEEE special values (-0., +/- Infty) are not recognized.

A multiplication by a factor of 4.0 converts between VAX F and IEEE 32 bit formats.

```
ZR32RL (NVAL, NP, INB, OUTB)
Inputs:
```

```
   NVAL   I      Number of values to convert
   NP     I      First value in INB to convert
   INB    R(*)   32-bit IEEE format values
Output:
   OUTB   R(*)   Local format values ("nan" values are replaced
                 with AIPS' indefinite value = 'INDE')
Generic version - does IEEE and VAX F formats for 32-bit machines,
is stubbed with STOP for all others.
```

ZR64RL

Converts from 64 bit IEEE floating format data to local double precision (or corresponding 64 bit precision).
The IEEE format is:

```
          1         2         3         4         5         6
01234567890123456789012345678901234567890123456789012345678901234567890123
seeeeeeeeeeemmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm
```

where sign = -1 ** s, exponent = eee..., mantissa = 1.mmmmm...

The value is given by:

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

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

```
ZR64RL (NVAL, NP, INB, OUTB)
Inputs:
   NVAL   I      Number of values to convert
   NP     I      First value in INB to convert
   INB    D(*)   64-bit IEEE format values
Output:
   OUTB   D(*)   Local format values ("nan" values are replaced
                 with AIPS' D.P. blank = 'INDE    ')
Generic version  - does IEEE and VAX G formats for 32-bit machines,
is stubbed with STOP for all others.
```

ZR8P4

Converts between pseudo I*4 and Double Precision. The main use of this is to handle 1*4 integers in buffers with mainly I*2 integers for tape IO - i.e. EXPORT and IBM format headers. The assumption is that the full buffer has been treated with (on read of tape) ZI16IL and needs to be repacked to 32 bits and then changed to local integer and then to local D.P. And the reverse on tape writing.

```
ZR8P4 (OP, INTG, DX)
Inputs:
   OP     C*4    '4IB8' IBM I*4 to D
                 '8IB4' D to IBM I*4
In/out:
```

```
   INTG   I(2)   the I
   DX     D      the D
IBM pseudo-I*4 has the form of a 2's complement, 32-bit integer
with the most significant 16 bits in the I word of lower index and
the least significant 16 bits in the I word of higher index.
Generic version - may work for all
```

ZRDMF

Convert data packed in DEC-Magtape format (DMF) to pairs of local 32-bit integers.
The DMF format is:

```
Track    1    2    3    4    5    6    7    8
Byte
  1      F0   F1   F2   F3   F4   F5   F6   F7
  2      F8   F9   F10  F11  F12  F13  F14  F15
  3      F16  F17  R0   R1   R2   R3   R4   R5
  4      R6   R7   R8   R9   R10  R11  R12  R13
  5       0    0    0    0   R14  R15  R16  R17
```

where Rn refers to the right halfword, Fn to the left halfword.

Since the purpose of this routine is to read Modcomp tapes written with this peculiar format, F16, F17, R16 and R17 (the high order bits) are zero for VLA data, but are used for the word count.

The first word (5 bytes) of a tape block contains the word count of the block. The word count is a 16-bit, twos-complement integer comprised of bits R2-R16. All other words are treated as pairs of 16-bit, twos-complement integers comprising bits F0-F15 and R0-R15.

Input data is assumed to be packed into 1 1/8 integers and output data will be returned in a pair of local integers per DEC-10 word. The first integer of each pair corresponds to the left DEC-10 halfword (zero for the word count) and the second to the right halfword. The sign of each halfword is preserved on expansion to a local integer.

```
ZRDMF (NWORDS, INBUF, OUTBUF, FLAG)

Inputs:
   NWORDS   I          Length of the input buffer in DEC-10
                       words
   INBUF    I(*)       Input buffer containing DMF format data
   FLAG     I          If > 0, the first word word is the
                       beginning of a tape block
Output:
   OUTBUF   I(*)       Output buffer containing two local
                       integers per input DEC-10 word
Generic version (stubbed).
```

ZRHPRL

Converts from 32 bit Hewitt-Packard floating format to local single precision.

```
HP R*4 format:
          1         2         3
01234567890123456789012345678901
smmmmmmmmmmmmmmmmmmmmmmmmeeeeeez
The value can be determined as follows:
man = smmmmmmmmmmmmmmmmmmmmmmmmm is a two's complement signed
```

```
      integer.
exp = zeeeeeee is a two's complement signed integer.
value = man * 2.0 ** (exp - 23)

ZRHPRL (NVAL, NP, INB, OUTB)
Inputs:
   NVAL   I      Number of values to convert
   NP     I      First value in INB to convert
   INB    R(*)   32-bit HP format values
Output:
   OUTB   R(*)   Local format values.
Generic version; should work on any machine.
```

ZRLR32

Converts from local single precision to 32 bit IEEE floating format. See ZR32RL for the description of IEEE 32 bit format. The AIPS internal format for an invalid number is the value which has the same bit pattern as 'INDE'.

A multiplication by a factor of 4.0 converts between VAX F and IEEE 32 bit formats.

```
ZRLR32 (NVAL, NP, INB, OUTB)
Inputs:
   NVAL   I      Number of values to convert
   NP     I      First value in OUTB for result
   INB    R(*)   Local format values
Output:
   OUTB   R(*)   32-bit IEEE FORMAT values ('INDE' values are
                 replaced with "nan")
Generic version - does IEEE and VAX F formats for 32-bit machines,
is stubbed with STOP for all others.
```

ZRLR64

Converts from local double precision (or corresponding 64 bit precision) to 64 bit IEEE floating format. See ZR64RL for a description of the IEEE 64 bit format.

The AIPS internal format for an invalid number is the value which has the same bit pattern as 'INDE '.

A multiplication by a factor of 4.0 converts between VAX G and IEEE 64 bit formats.

```
ZRLR64 (NVAL, NP, INB, OUTB)
Inputs:
   NVAL   I      Number of values to convert
   NP     I      First location in OUTB for results
   INB    D(*)   Local format values
Output:
   OUTB   D(*)   64-bit IEEE format values ('INDE    ' values are
                 replaced with "nan")
Generic version  - does IEEE and VAX G formats for 32-bit machines,
is stubbed with STOP for all others.
```

ZRM2RL

Convert Modcomp single precision floating point data into local single precision floating point.

```
ZRM2RL (NWORDS, INBUF, OUTBUF)
Inputs:
   NWORDS   I      Length of the input buffer in words
   INBUF    R(*)   Input buffer containing Modcomp R*4 data
Output:
   OUTBUF   R      Output buffer containing local REAL data
Notes:
   Before call, input buffer should have its bytes flipped via
   ZI32IL which will leave the values in one local 32-bit integer

   Expects, after word flip, sign bit in bit 31 (1=>negative),
   bits 22:30 are the  exponent biased by 256(?), bits 0:21 are
   the normalized fraction.  Negative values are obtained by 2's
   complement of the whole word.

   Should work inplace.

Generic version - stub
```

ZUVPAK

Routine to pack uv data with magic value blanking. One AIPS logical uv data record is processed at a time. Two values are packed into a single real value.

```
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.
Version for machines with valid integers half the size of reals.
```

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.
Version for machines with valid integers half the size of reals.
```

ZX8XL

Converts a FITS table bit array to an AIPS bit array. An AIPS bit array has the bits in locations of increasing signifigance NBITWD per integer. A FITS bit array has the bits in order of decreasing signifigance with 8 bits per real world byte, zero filled

```
ZX8XL (NBIT, INB, OUTB)
Inputs:
   NBIT   I      Number of bits
   INB    I(*)   Input buffer of FITS bit array data as 8-bit byte
                 stream.
Output:
   OUTB   I(*)   Out buffer of AIPS bit array data.
Generic version: works whenever local integer is a multiple of 16
bits and has no more than 64 bits.
```

ZXLX8

Converts an AIPS table bit array to an FITS bit array. An AIPS bit array has the bits in locations of increasing signifigance NBITWD per integer. A FITS bit array has the bits in order of decreasing signifigance with 8 bits per real world byte, zero filled.

```
ZXLX8 (NBIT, INB, OUTB)
Inputs:
   NBIT   I      Number of bits
   INB    I(*)   Input buffer of AIPS bit array data.
Output:
   OUTB   I(*)   Out buffer, note an integral number of 16-bit
                 integers in OUTB are modified.  OUTB is left in
                 the form of IEEE integers (i.e. ms byte first).
Generic version: works whenever local integer is a multiple of 16
bits and has no more than 64 bits.
```

15.11.5 Directory and Text File

The following describes the first level text file "Z" routines as documented in the APLGEN directory.

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
Generic version.
```

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
Generic version - uses ZDIR, ZFULLN, and Fortran functions INQUIRE
and then calls ZTOPE2.
```

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
Generic version - assumes simple Fortran IO.
```

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
Generic version: uses Fortran CLOSE
```

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
Generic version.
```

ZTXMAT

Open a directory and find a list of member file names whose first NCH characters match the first NCH characters of MNAME plus extension (based on file type).

```
ZTXMAT (IVOL, PNAME, MNAME, NCH, VERSON, NAMES, NNAM, IERR)
Inputs:
   IVOL     I         Disk volume containing file, (ignored)
   PNAME    C*48      Physical file name (only used to determine
                      file type)
   MNAME    C*8       Test file name
   NCH      I         Number of characters to compare (<= 8)
                         0 is okay -> get full directory
   VERSON   C*48      Logical name for directory or version of
                         directory to search (for file type
                         specific directories)
In/out:
   NNAM     I         Number of names in NAME: input = max,
                         output = actual used
Output:
   NAMES    H*8(*)    File names which match the given file spec
                      NOTE HOLLERITH for lower level routines
   IERR     I         Error return code: 0 => no error
                         1 => no matches found
                         2 => error in inputs
                         3 => error opening directory
Generic version - uses ZDIR, ZFULLN, ZTXMA2
```

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
Generic version --- uses ZTXOP2 for actual open.
```

15.11.6 Virtual Devices

The following describes the first level virtual device "Z" routines as documented in the APLGEN directory.

ZVTVCL

Close the virtual TV device channel in the client (VTV) machine to the server (remote, real-TV) machine - uses ZVTVC2.

```
ZVTVCL (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 or flush error
                 2 => file already closed in FTAB
                 3 => both errors
                 4 => erroneous LUN
Generic version - calls ZVTVC2.
```

ZVTVGC

Opens the connection in TVMON to the remote machine which is running the AIPS VTV (Virtual TV) code. This differs from ZVTVO3 in that the socket is already there and an old connection must be closed before a new one can be accepted. Called by ZVTVRX.

```
ZVTVGC (FCB, IERR)
Outputs:
   FCB   I(*)   File descriptor
   IERR  I      Error: 0 => okay
Generic version - stub
```

ZVTVOP

Does whatever is needed to open communication from the current program to a remote machine that has a real TV display attached. This is a generic upper level Z routine.

```
ZVTVOP (LUN, IND, IERR)
Output:
    IERR  I     Error code: 0 => ok
                   1 = LUN already in use
                   2 = file not found
                   3 = volume not found
                   4 = excl requested but not available
                   5 = no room for lun
                   6 = other open errors
Generic version - uses ZVTVO2 and ZPHFIL.
```

ZVTVRC

Close the virtual TV device channel: from receiver point of view

```
ZVTVRC (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 or flush error
                 2 => file already closed in FTAB
                 3 => both errors
                 4 => erroneous LUN
Generic version - uses ZVTVC3.
```

ZVTVRO

Does whatever is needed to enable communication from the current program (TVMON) on a machine that has a real TV display attached to any remote machine needing the display. This is a generic upper level Z routine.

```
ZVTVRO (LUN, IND, IERR)
Inputs:
   LUN    I    An LUN to use (not TVLUN or TVLUN2)
Output:
   IND    I    FTAB location opened
   IERR   I    Error code: 0 => ok
                  1 = LUN already in use
                  2 = file not found
                  3 = volume not found
                  4 = excl requested but not available
                  5 = no room for lun
                  6 = other open errors
```

ZVTVRX

Performs "I/O" over some communication mechanism to a cooperating program with the intention of driving a real TV device controlled by this program.

```
ZVTVRX (FIND, BUFSW, HBUF, IERR)
Inputs:
   FIND    I      FTAB location for socket to remote machine
   BUFSW   I      Number 16-bit integers total data to send
                  If BUFSW > 0, write to client; if <= 0, read
                  from client machine.
In/Out:
   HBUF    I(*)   Buffer: in FITS standard 16 bit
                     (1,2) : READ or WRIT
                     (3,5) : subroutine name packed ASCII
                     (6)   : BUFSW - 8
                     (7)   : BUFSR
                     (8)   : error returned to client
                     (9..) : extra data
Output:
   IERR    I      Error code: 0 => ok.
Generic version - does little except call ZVTVX3
```

ZVTVXF

Performs "I/O" over some communication mechanism to a cooperating program with the intention of driving a real TV device controlled by the cooperating program.

```
ZVTVXF (BUFSW, BUFSR, HBUF, IERR)
Inputs:
   BUFSW  I      Number 16-bit integers extra data to send
   BUFSR  I      # 16-bit integers of extra data to receive
In/Out:
   HBUF   I(*)   Buffer:  in FITS standard 16 bit
                    (1,2) :  READ or WRIT
                    (3,5) :  subroutine name packed ASCII
                    (6)   :  BUFSW
                    (7)   :  BUFSR (TV IERR returned)
                    (8)   : error return
                    (9..) :  extra data
Output:
   IERR   I      Error code:  0 => ok.
Generic version - does little except call ZVTVX2
```

15.11.7 Miscellaneous

The following describes the first level miscellaneous "Z" routines as documented in the APLGEN directory.

ZADDR

Determine if two addresses are the same.

```
ZADDR (ADDR1, ADDR2, IERR)
Inputs:
   addr1   I*2   Address 1
   addr2   I*2   Address 2
Output:
   ierr    I     Error return code:
                    0 => addresses are the same
                    1 => addresses are different
Generic version - stub
```

ZERROR

This routine will attempt to translate the system error code and if appropriate (i.e., FCB(1).NE.-999) print the name of the file or device on which the error occurred as well as the contents of the the file control block (blocks, if map I/O).

```
ZERROR (ZRTNAM, SYSERR, PNAME, FCB, MAP)
Inputs:
   ZRTNAM   C*6    Z-routine where the error occurred
   SYSERR   I      System error code (also stored in FCB)
   PNAME    C*48   Physical file name; ' ' => unknown (omit)
   FCB      I(*)   File control block in FTAB for the file (or
                   device) also containing the system error code
                   FCB(1) = -999 => omit
   MAP      L      Map or non-map I/O involved?
Common: DMSG.INC
   DBGAIP   I      MOD(DBGAIP,10) > 2 => force ZERR02 call
                   MOD(DBGAIP,10) > 2 => dump FCB(s)
Generic version - calls ZERR02.
```

ZHEX

Convert decimal to nc character hexidecimal 'string' - leading blanks are made into 0's, i.e. "illegal" Fortran format Znc.nc

```
  ZHEX (IVAL, NC, HVAL)
Input:
   IVAL   I       Decimal value to convert
   NC     I       Width of receiving field
Output:
   HVAL   C*(*)   String receiving hexidecimal conversion
Generic version - klunky but works
```

ZKDUMP

Dump portions of an array in a variety of formats (e.g., Fortran I, E, A and hexidecimal).

```
ZKDUMP (I1, I2, K, C)
Inputs:
   I1    I      First subscript in integer array to dump
   I2    I      Last subscript in integer array to dump
   K     I(*)   Integer array
   C     R(*)   Real array equivalenced to K in calling routine
Generic version - uses ZHEX, assumes 32-bit integers
```

ZMSGER

This routine will attempt to translate the system error code and if appropriate (i.e., FCB(1).NE.-999) print the name of the file or device on which the error occurred as well as the contents of the the file control block (blocks, if map I/O). For MSGWRT - to avoid recursions.

```
ZMSGER (ZRTNAM, SYSERR, PNAME, FCB, MAP)
Inputs:
   ZRTNAM   C*6    Z-routine where the error occurred
   SYSERR   I      System error code (also stored in FCB)
```

```
   PNAME    C*48   Physical file name; ' ' => unknown (omit)
   FCB      I(*)   File control block in FTAB for the file (or
                   device) also containing the system error code
                   FCB(1) = -999 => omit
   MAP      L      Map or non-map I/O involved?
Common: DMSG.INC
   DBGAIP   I      MOD(DBGAIP,10) > 2 => force ZERRO2 call
                   MOD(DBGAIP,10) > 2 => dump FCB(s)
Generic version - calls ZERRO2.
```

ZMYVER

Determine the default AIPS version (OLD or NEW or TST). Error messages are only written to the terminal since this routine is called before message handling via MSGWRT is established.

```
ZMYVER
 (no call arguments)
Output in DDCH.INC:
   VERNAM   C*4   String = 'OLD:', 'NEW:' or 'TST:'
Generic version.
```

Chapter 16

Calibration and Editing

16.1 Introduction

This chapter will describe the system of routines to edit and calibrate "raw" data multi-source uv data and single dish data files. This chapter is intended to be used both by programmers writing calibration or editing application software and programmers maintaining the more fundamental routines.

The basic design of this system is patterned after the now defunct VLA calibration system implemented on a Dec-10. All data for a single observing band (e.g., C band) with compatable sets of frequencies and polarizations are kept in raw form in a single file. Tables which contain calibration and editing information are manipulated until the user is satisfied with the results (or simply exhausted). The calibration and editing tables may then be applied to the data to produce single-source, calibrated and edited data files in the traditional AIPS form.

There are two parallel and overlapping calibration systems, one for interferometer (also called uv data) and the other for single dish data or other filled aperature measurments. The former system is to make amplitude and phase like corrections and the latter for amplitude and pointing corrections. These calibration systems depend heavily on the use of tables and the reader is encouraged to review the chapter on tables in this manual. Access to the raw data files is through the routines UVGET and SDGET for interferometer and single dish data. These routines can optionally calibrate and edit a selected subset of the data and return it in a specified set of Stokes' parameters.

16.2 Multi-source uv Data Files

The most useful organization of uv data for purposes of editing and calibration is to have all data which is to be calibrated together in the same file in time order. The structure of multi-source files is very similar to single-source files, both types are described in detail in the chapter on disk files. Multi-source files differ from single source files by the presence a source number random parameter, an optional FQ (frequency set[) identifier random parameter, and a number of tables extension files. Because of the similarity with single-source files, the bulk of AIPS software can process multi-source as well as single-source files if no distinction needs to be made between sources. Descriptions of the contents of the calibration and editing tables are given in Appendix C. Access to raw interferometer data is through the routine UVGET (see the detailed description at the end of this chapter). UVGET can select and optionally edit and calibrate the data.

16.2.1 Distinguishing Sources

Each logical record in a multi-source file contains a source identification number as a random parameter; the source information for the individual sources is kept in the source (SU) table. When the name, position, flux density, etc. of a given source is needed, the appropriate record of the source table should be consulted. The position of the source number random parameter is obtained by UVPGET and placed in the DUVH.INC

include common as ILOCSU. See the chapters on the catalog header records and disk files for more details on determining the structure of the logical records.

16.2.2 Time Order

The most convenient order for multi-source files is time-baseline order; that is, strictly in time order. This allows indexing the data, which, in turn, allows rapid access to small portions of the file. For this reason, most of the calibration and editing software require the data to be in time order and require an index (NX) table or, at least, are much more efficient if one is available.

16.2.3 Scans

Observations generally consist of a series of "scans" or sequences of data on the same source. Since the duration of some of these observations may be quite long, the AIPS definition of a scan is more general and a long sequence on the same source may be broken into several "scans". The definition of a scan is the data described by an index (NX) table entry, which must be a contiguous sequence of data all on the same source.

16.2.4 Subarrays

Some data sets may contain observations of several sources at the same time by different members of the array; this is especially common in VLBI data, for which all array elements will not be able to see a given source over the same time range. This complication is dealt with by the concept of subarrays. A further complication is that the AIPS definition of subarrays allows partially or completely disjoint sets of antennas in different subarrays; this allows combining data from distinct synthesis arrays, while keeping track of what data came from where. It may be desirable to calibrate data from separate arrays together to insure that the phases all refer to the same place on the sky.

Each subarray in a data base will have a corresponding antenna (AN) table defining the properties of the array members and the actual observing date and frequency; this is the case even if the contents of the antenna files are identical. More details on the use of subarrays can be found in the chapter on disk files and later in this chapter.

16.2.5 Compressed Data

Data may be kept in "Compressed" form with individual correlation values packed into a single real word. The details of this packing are machine dependent and is done by the routines ZUVPAK and ZUVXPN. These packed values may be blanked by magic value blanking to indicate that they are flagged and each visibility record will have a single weight and scaling random parameters. The use of the compressed format for interferometer data is transparent when UVGETUVGET is used for access as the data is returned in expanded form.

16.2.6 Frequency Sets (FQ id)

A given set of data may have data taken with different sets of frequencies or bandwidths at different times; as for example, in rotation measure experiments or frequency switched spectral line observations. These may be kept in the same data set if they all have the same number of IFs, channels and polarizations. A given set of frequencies and bandwidths are specified by an entry in the FQ table. A random parameter in each visibility record labeled "FQID" specifies the entry in the FQ table that describes the frequencies and bandwidths of that record.

16.2.7 Tables

The manipulation of tables is at the heart of the calibration and editing software and much information about the contents of the associated uv data file is contained in them. Some of the tables, such as the source (SU), antenna (AN) and frequency (FQ) tables, are necessary to interpret values in the uv data file. Other

tables, such as the flag (FG), calibration (CL and CS), and solution (SN) tables, contain information which can be used to edit and/or calibrate the data.

A short description of the major tables follows; a detailed definition of each table is given at the end of this chapter.

- AN table : this table defines the array geometry, the observing date and frequency, and some time invariant properties of each array element such as the instrumental polarization parameters.
- BL table : this table contains the gain corrections which is peculiar to pairs of antennas (baseline) and cannot be factored into antenna based components.
- BP table : this table contains antenna based gain corrections for each spectral channel in a data base.
- CL table : this table contains the calibration information necessary to calibrate the contents of an interferometer data file. Multi-source files only.
- CS table : this table contains the calibration information necessary to calibrate the contents of a single dish data file. Amplitude and pointing corrections are included.
- FG table : this table contains descriptions of data to be rejected from further consideration. Multi-source files only.
- FQ table : this table gives the frequency offset of each IF from the reference frequency. This allows IFs to have arbitrary frequency offsets. Multiple sets of frequencies and/or bandwidths are allowed with the use of a "FQID" random parameter which specifies the entry in this table. The calibration routines currently (1990) only process a single FQ id at a time.
- NX table : the index table contains an index of the uv data file. Multi-source files only.
- SN table : this table contains the solutions obtained from a calibration routine. The contents of these tables are applied to the CL table for multi-source files and to the data for single-source files.
- SU table : the source table contains the information peculiar to a given source such as the name and position.

16.3 Editing Basics

The terms editing and flagging will be used interchangeably in this chapter. The processes of editing is to remove data that is defective or otherwise unwanted. This is done in the context of the calibration software by specifying the unwanted data in terms of time range, subarray, source, antenna or baseline, etc. These descriptions of the data to be removed are kept in the flag (FG) table with a short, optional description of the reason the data is unwanted.

When these flags are applied, bad data is indicated by a non-positive weight associated with the relevant complex value. If all such values in a given logical record are marked bad, then the calibration software will not pass that logical record. Editing is invoked in UVGET or SDGET by setting DSEL.INC variable *DOFLAG* = .TRUE. and *FGVER* to the desired FG table version number.

16.4 Interferometric Calibration Basics

Calibration is the process of removing the instrumental and atmospheric effects from the data and referring the residual phases to the desired position on the sky. Note: the phase of the visibility data is the residual phase after the model phase at the reference position is subtracted. Most of the instrumental and all of the atmospheric effects are a function of antenna rather than baseline. Baseline dependent calibration, if done at all, is only done after the antenna based calibration. The bulk of the calibration software is for antenna based calibration.

Calibration information can be derived from external sources, such as system temperature measurements, atmospheric models, etc., or from internal methods, such as observations of sources with known flux density, position and structure (calibrator sources).

A related means of calibration is self-calibration, in which the current model of a source (e.g., CLEAN components) is iteratively used to determine the instrumental or atmospheric effects directly from the observations of the source of interest. This technique may lead to a considerable increase in the dynamic range of the final image at the cost of losing precise positional and/or flux density scale information.

The information needed to calibrate the data in a multi-source file is kept in the calibration (CL) table. In the case of external calibration, the corrections may be applied directly to the CL table. The process of internal calibration interacts with editing and other uncertainties and so frequently takes a number of iterations.

The calibration information needed is: (1) a complex gain for each correlator value (complex visibility), and, for VLBI data, (2) a residual group delay to correct the derivative of phase with frequency, and (3) a residual phase delay rate (sometimes called fringe rate) to correct the time derivative of the phase. The calibration to be applied to a given visibility measurement is interpolated from the CL table entries surrounding the visibility measurement in time.

16.4.1 Internal Calibration

Internal calibration is a two-step process; the first step is usually to determine calibration values from calibrator source observations; these solutions are kept in a solution (SN) table. The second step is to apply these solutions to the calibration (CL) table. Solutions for different calibrator sources may need to be determined independently and these solutions are put in separate SN tables. Thus, when the SN table are to be applied to the CL tables, there may be several such tables.

The reference times of the SN and CL tables entries will, in general, not be the same; the CL table entries are usually generated when the data is first read into the system. Solutions may be determined from all the data from a given scan or from shorter periods. The SN tables are applied to the CL tables by interpolating between SN table entries to the times of the CL table records and then applying the residual corrections from the SN table to the CL table.

An additional complication is that a given CL table may have been applied to the data before the determination of a set of solutions. In this case, the SN table needs to be applied to the CL table which was used to calibrate the data. In general, it is the responsibility of the user to insure that the correct version of the CL table is used.

16.4.2 Smoothing

Under some circumstances, the user will wish to smooth some aspect of the solutions (e.g., the group delay residual) before applying it to the CL table. This is done by concatenating the SN tables, adjusting them to a common reference antenna, and then smoothing the combined table before applying it to the CL table.

16.4.3 Reference Antennas

An interferometer only measures phase differences between antennas, so internal calibration cannot determine absolute phases. It is usually the practice to refer the phases derived from internal calibration to a given antenna known as the reference antenna. Phase corrections (including delay and rate) cannot be smoothed or interpolated using solutions based on different reference antennas. Solutions using different reference antennas must be re-referenced to the same antenna before smoothing or interpolation. Since all calibration in AIPS involves interpolation, the solutions must always be re-referenced to the same antenna. Routine CALREF rereferences all phase like data in a given SN table to a common antenna. Details of this routine are given at the end of this chapter.

16.5 Observing Model

The phases of the visibility data are actually the residuals from a model phase. The total phase and its cousins (i.e., phase delay, group delay, and their time derivatives) are useful values in their own right. These values can be used to determine the locations of the antennas and the sources, and the orientation of the earth, to high accuracy.

The total phase-like model values are also necessary if high accuracy earth models (precession, UT1 corrections, polar motion, solid earth tides, ocean loading, etc., etc.) are to be used to apply phase corrections to the data. The need for such corrections increases with baseline length and is essential for phase coherent interferometers with baselines of length a thousand km or more.

The total model applied to the data is kept in the calibration (CL) table. The model is expressed in terms of delays and their time derivatives. Because dispersive media are involved, both the group and phase delay must be kept. The total model is arbitrarily divided into three parts: (1) a sinusoidal term, (2) an "atmospheric" term and (3) a "clock" term. The sinusoidal term is intended to contribute the bulk of the delay, being the delay due to the positions of the array elements on the earth or of a satellite in orbit. Since the sinusoidal term is purely geometrical, there is no distinction between group and phase delay. (Note: use of a sinusoidal term may be changed to a polynomial in the future).

The distinction between the "atmospheric" and "clock" terms is, as yet, poorly defined, but both are allowed to have group and phase delay components. Whenever a new residual correction is determined, e.g., when an SN table is applied to the CL table, the "atmospheric" or "clock" total model values need to have the corresponding correction made. This procedure will result in total model values when the calibration process is complete.

16.6 Applying Calibration to Interferometer Data

This section describes the various calibration and spectral smoothing operations that can be invoked using UVGET for interferometer data. These functions are controlled by variables in the commons in DSEL.INC as described below. The values in the DSEL.INC commons are most convienently initialized using routine SELINI. Desired options may then be explicitly set.

16.6.1 Amplitude, Phase, Delay and Rate

Amplitude, phase, delay and fringe rate corrections are invoked by setting DSEL.INC variable *DOCAL*=.TRUE. and *CLUSE* to the version number of the CL (multi-source) or SN (single-source file) table to be applied. This corrections is done in DATCAL using interpolated values in tables maintained by CGASET and CSLGET. An amplitude correction is applied to correct for amplitude loss due to time averaging if a fringe rate correction is made; this corrections is baseline dependent. To make this correction DSEL.INC variable *DXTIME* should be set to the integration time of the data in days.

The weights associated with the data may be calibrated by the amplitude correction. This is necessary if the weights are to be proper statistical weights but this is inconsistent with older useage. The question of calibration of weights is frequently left as a user option and is enabled by setting DSEL.INC variable *DOWTCL*=.TRUE.

16.6.2 Baseline Dependent Calibration

Instrumental errors may be introduced in a mannar which cannot be factored into antenna specific components. The corrections for this type of error is called baseline dependent calibration. Baseline dependent errors may be due to such effects as bandpass mismatches or defects in the correlation process. In principle, these effects can be multiplicative or additive although at present only multiplicative corrections are fully implemented. Baseline dependent corrections, both additive and multiplicative, are contained in the BL table. Baseline dependent calibrations maybe time variable, in which case the applied correction is interpolated in time. Baseline dependent corrections are applied in DATCAL using a table maintained by BLSET. Baseline dependent calibration is invoked by setting DSEL.INC variable *DOBL*=.TRUE. and *BLVER* to the version of the desired BL table.

16.6.3 Bandpass Calibration

Data in which multiple frequency samples are present in the same IF generally require a frequency dependent amplitude and/or phase correction. These are generally needed due to imperfect filters in the signal processing system in the individual antennas and thus may be factored into antenna based components. These corrections may be slowly variable with time. Amplitude and phase corrections may be determined from interferometric observations or amplitude corrections from autocorrelation measurments. These bandpass corrections are kept in the BP table and are applied by DATBND using tables maintained by BPGET, SCLOAD and SCINTP. The call sequence to these routines is described in detail at the end of this chapter. These routines have three modes of operation: 1) a constant set of corrections; 2) using the corrections closest in time; and 3) full interpolation in time of the bandpass corrections. Several scratch files may be used in this process. bandpass calibration is invoked using DSEL.INC variable *DOBAND*=1,2 or 3 for the three option described above and *BPVER* set to the desired BP table.

16.6.4 Spectral Smoothing

Spectral measurments are sometimes smoothed to increase the signal to noise ratio in features wider than the channel separation or as a filter to remove the Gibbs Phenomenon (ringing due to truncation of the lag function). In the later case, a Hanning smoothing is effective. The currently available smoothing methods are Hanning, Gaussian, boxcar and sinc. Spectral smoothing is invoked by setting DSEL.INC variable *DOSMTH* to .TRUE. and specifing the desired smoothing in array *SMOOTH*. Initialization for smoothing is done by SETSM and the smoothing is done by SMOSP.

16.6.5 Polarization Calibration

The instrumental polarization calibration constants for a single FQ id are contained in the subarray AN table. Several possible parameterizations of the instrumental polarization are possible but only a linear approximation for circular or linear (but not mixed) feeds are fully supported. More details are given in the description of the AN table given in Appendix C. If DSEL.INC variable *DOPOL*=.TRUE. then routine DATPOL will correct for the instrumental polarization and remove the effects of parallactic angle rotation for antellas with an alt-az mount. The VLA is recognized as a special case in which all antennas have the same parallactic angle. This requires that no parallactic angle corrections have been made prior to this.

At low frequencies a further corrections for ionospheris Faraday rotation will also be required. This information, if present, is kept in the CL table and is applied if polarization calibration is applied. This correction is applied by DATPOL using tables in DSEL.INC maintained by CSLGET and CGASET using values read from the CL table.

16.7 Data Selection

Data may be selected using a number of criteria specified via commons in DSEL.INC. Sources may be specified by name using character array *SOURCS*, a single qualifier *SELQUA* or calibrator code *SELCOD* (' ' means any, '* ' means any non blank calibrator code, '-CAL' means all but calibrators and anything else requires a match). Source selection is done by routine SOUFIL which indicates the selected sources by source id number in DSEL.INC variables *NSOUWD*, *DOSWNT*, *SOUWAN* and *SOUWTB*.

Selection by time is specified in array *TIMRNG* which gives the start date, hour, minute, second, end day, hour, minute and second. Day numbers are zero relative with respect to the subarray reference date. All zeroes mean all times are selected. A list of antennas whose data is to be included is specified by array *ANTENS*. If all elements of *ANTENS* are positive then selected data are to come from only those antennas. If any value is negative, then data using the antennas specified by the absolute values of the elements in *ANTENS* are to be ignored. The subarray may be selected using *SUBARR* for which zero means all. Data may be specified to be in an annulus in the uv plane by *UVRNG*. The first and second elements of this array are the inner and outer radii of the annulus in 1000's of wavelengths and zeroes mean all. The annulus is defined at the reference frequency of the subarray.

A range of spectral channels can be specified using *BCHAN* and *ECHAN* which are the lowest and highest channel numbers in each IF; zero means all. Similarly, a range of IFs may be specified with *BIF* and *EIF*. The desired Stokes' (polarization) type may be specified by *STOKES*. This will cause the Stokes' type requested to be returned if possible. The setup for this translation is done in DGINIT and the translation is done in routine DGGET. A blank value of *STOKES* means don't translate; other values are 'I', 'Q', 'U', 'V', 'IQU', 'IQUV', 'IV', 'RR', 'LL', 'RL', 'LR' 'HALF' (parallel pol.) and 'FULL' (RR,LL,RL and LR). Setting variable *DOACOR*=.TRUE. causes UVGET to return any auto correlation data found; *DOXCOR*=.TRUE. requests cross correlation data. *DOFQSL*=.TRUE. requests selection by FQ id where the desired FQID is given by *FRQSEL*. The value of *FRQSEL* may be determined directly by the user or specified by Frequency and bandwidth; this is done in routine FQMATC.

16.8 Table Access Routines

Much of the table access is built into the data access routines, but some tables (e.g., the source) may need to be accessed more generally. Most of the commonly used tables will have special access routines which are described in detail in the chapter on tables. The detailed descriptions of the contents of these tables are found in Appendix C.

16.9 Calibration Table Routines

The manipulation of the SN tables and their application to a CL table is done by a single AIPS task, CLCAL using routine CLUPDA. This routine concatenates SN tables and re-references the phases to a common reference antenna, smooths the SN table and applies the combined SN table to specified entries in an input CL table, and finally writes them to an output table. If no CL table exists in a multi-source file, one is created and the SN table copied into it. The details of the call sequence of CLUPDA are at the end of this chapter.

The detailed contents of the calibration related tables are subject to change with time. To minimize the difficulties associated with this there are several routines that check if a given table is of the current format and reformats it if it is an old version. These routines are briefly described in the following and in detail at the end of this chapter.

- BLREFM - Checks existence of BL table, changes format if necessary.
- BPREFM - Checks existence of BP table, changes format if necessary.
- CLREFM - Checks existence of CL table, changes format if necessary.
- SNREFM - Checks existence of SN table, changes format if necessary.

16.10 Data Access Routines

The principal means of accessing multi-source data is through the routine UVGET. This routine will apply a variety of selection criteria, translate polarization types, and, optionally, apply calibration and editing tables. A detailed description of UGVET is given else where in this chapter. A related routine, CALCOP, uses UVGET to create and fill a file with the desired data. UVGET also calls DGHEAD which prepares a catalog header record which describes the output data; this is left in DSEL.INC array *CATBLK*. The catalog header record for the input uv data is in array *CATUV*.

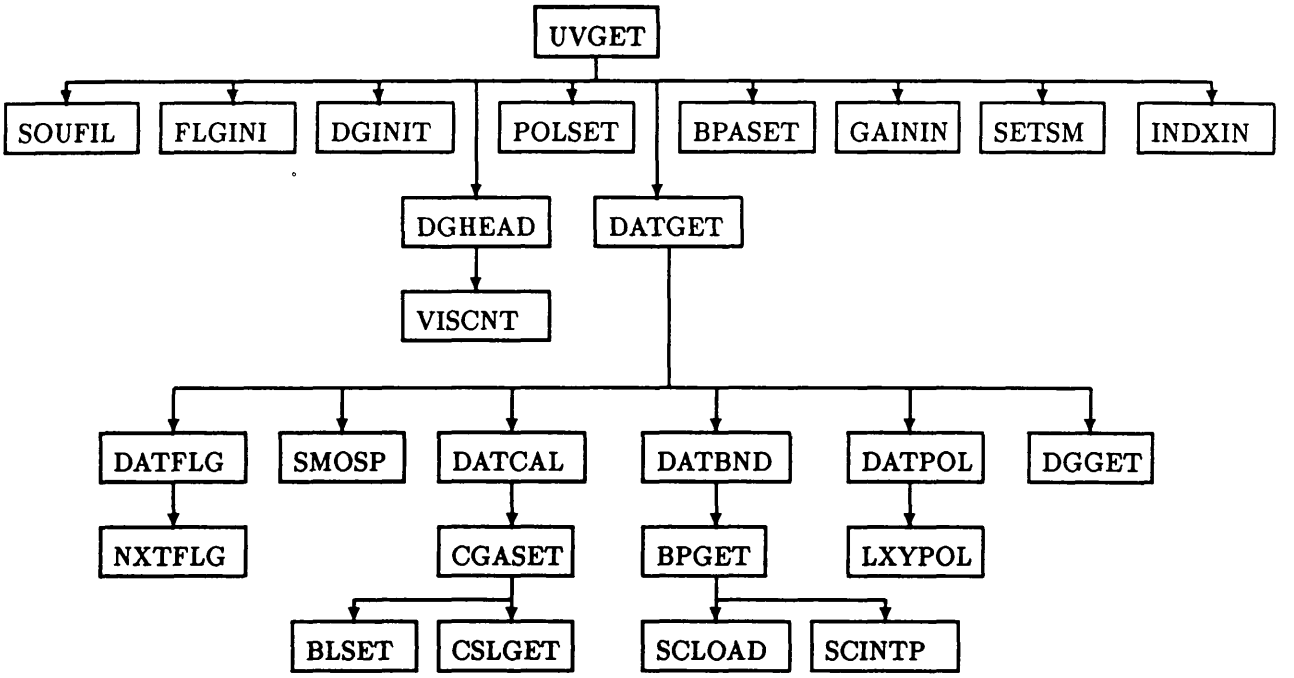
UVGET and CALCOP will work on single-source files as well as multi-source files. This allows calibration software written for multi-source files to work for single-source files whenever possible.

The communication of the selection criteria, flagging and calibration information, etc. is through a common contained in the include DSEL.INC. The commons in these includes contain most of the variables and arrays needed by the inner workings of UVGET.

16.10.1 Structure of the Interferometer Data Access System

UVGET calls a hierarchy of routines to do the necessary functions. While most programmers using UVGET will not need to be familiar with the details of the inner workings of the UVGET system, some applications may require knowledge of, or modifications to, this system.

The relations between the major routines of the UVGET system are shown in the following diagram:



Interferometer Data Calibration

16.10.2 Data Access Routines

In the following, a short description is given for each of the major routines in the data access system; detailed descriptions of the call sequences etc. are given later in this chapter.

1. UVGET : the top level data access routine. UVGET controls the initialization of the various files and arrays, calls DATGET to process the data, and closes the necessary files when done.
2. SOUFIL : this routine converts the list of sources to source numbers, which are filled into a common.
3. GAININ : initializes the gain (CL or SN) table to be applied to the data.
4. DGINIT : sets up to translate data (e.g., RR, LL to IPOL), if necessary.
5. POLSET : initializes common arrays needed for polarization calibration.
6. BPASET : initializes common arrays and possibly scratch files for bandpass calibration.
7. SETSM : initializes common variables for spectral smoothing.
8. INDXIN : initializes the index table I/O and finds the first relevant index record.
9. FLGINI : initializes reading flagging (FG) table.
10. DGHEAD : fills the catalog header record to correspond to the output data.
11. VISCNT : estimates the number of visibility records requested from the index table.

12. DATGET : reads data, applies flagging and calibration, and translates polarization.
13. DATFLG : flags data.
14. SMOSP : smooth spectra.
15. DATCAL : calibrates data.
16. DATPOL : Applies instrumental polarization corrections, parallactic angle corrections and ionospheric Faraday rotation corrections. Converts linear feed data to circular if necessary.
17. DATBND : applies bandpass correction.
18. DGGET : translates polarization.
19. NXTFLG : manages the internal arrays containing the currently active flagging criteria.
20. CGASET : interpolates gain table entries to current time.
21. CSLGET : finds gain table entries on both sides of the current time and reads the values into a common array.
22. BLSET : sets common arrays for baseline dependent calibration.
23. LXYPOL : determines time dependent polarization correction matrix for data observed with linear feeds.
24. BPGET : sets current bandpass corrections in common.
25. SCLOAD : manipulates Bandpass correction scratch files.
26. SCINTP : interpolates bandpass corrections in time.

16.11 Example Using UVGET

The following example illustrates the use of UVGET to access a multisource uv data file with calibration and editing as requested by the user through adverb values passed to the routine. In this example the user specified calibration and flagging options are applied and the selected data written to a scratch file that is created by CALCOP.

```
LOCAL INCLUDE 'PARMS.INC'
C                                       Common with task adverb values
      REAL      DISKIN, SEQIN, XQUAL, XTIME(8), XBCHAN, XECHAN, XBIF,
     *   XEIF, XDOCAL, XDOPOL, XSUBA, XFLAG, XGUSE, XBLVER, XDOBND,
     *   XBPVER, XSMOTH(3), XINT, XBAND, XFREQ, XFQID
      CHARACTER NAMEIN*12, CLAIN*6, XSOUR(30)*8, XCALCO*4, XSTOK*4
      COMMON /PARMS/ DISKIN, SEQIN, XQUAL, XTIME, XBCHAN, XECHAN, XBIF,
     *   XEIF, XDOCAL, XDOPOL, XSUBA, XFLAG, XGUSE, XBLVER, XDOBND,
     *   XBPVER, XSMOTH, XINT, XBAND, XFREQ, XFQID
      COMMON /CPARMS/ NAMEIN, CLAIN, XSOUR, XCALCO, XSTOK
LOCAL END
      SUBROUTINE CALIT (IDISK, ICNO, CATIN, SCRNO, IRET)
C---------------------------------------------------------------------
C   Routine to calibrate and edit a user selected set of data and
C   write it to a scratch file.
C   Inputs:
C      IDISK    I       Input file disk number.
C      ICNO     I       Input file catalog slot number.
C      CATIN    I(256)  Input file catalog header record.
```

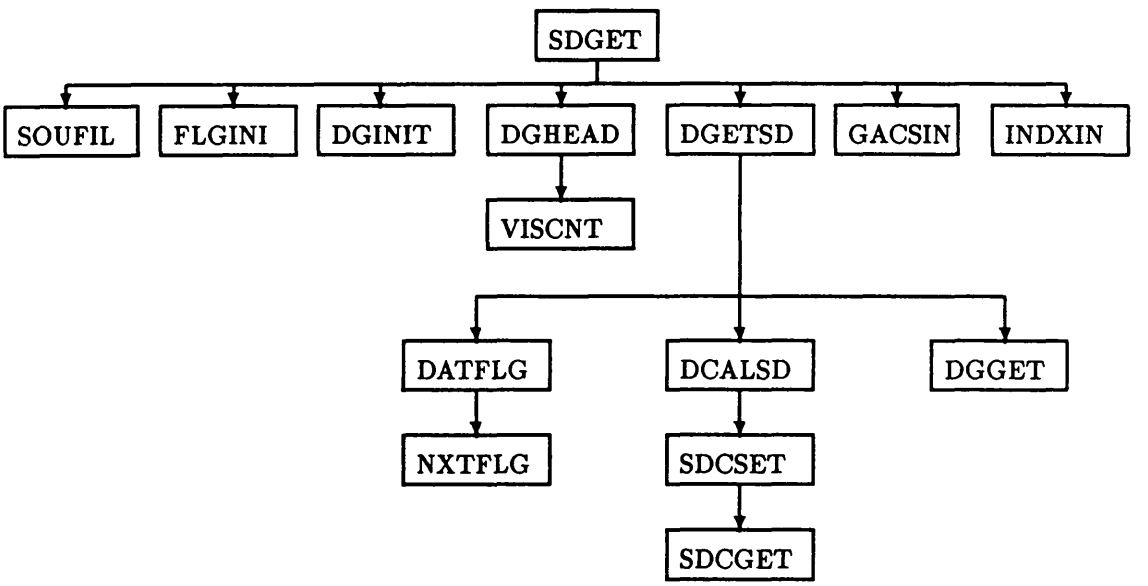
```
C   Inputs from common (PARMS.INC):
C      NAMEIN   C*12     Input uv data file name.
C      CLAIN    C*6      Input uv data file class.
C      DISKIN   R        Input uv data file disk no.
C      SEQIN    R        Input uv data file sequence no.
C      XSOUR    C(30)*8  List of source names
C      XQUAL    R        Qualifier selected
C      XCALCO   C*4      Calibrator code selected.
C      XTIME    R(8)     Time range, start d,h,m,s, end d,h,m,s
C      XSTOK    C*4      Desired Stokes parameter
C      XBCHAN   R        Lowest channel number to select.
C      XECHAN   R        Highest channel number to select.
C      XBIF     R        Lowest IF number to select.
C      XEIF     R        Highest IF number to select.
C      XDOCAL   R        If > 0 then apply calibration.
C      XDOPOL   R        If > 0 then apply polarization calibration.
C      XSUBA    R        Subarray number.
C      XFLAG    R        Flag (FG) table version to apply.
C      XGUSE    R        Calibration (CL or SN) table to apply.
C      XBLVER   R        Baseline dependent (BL) cal table to apply.
C      XDOBND   R        Bandpass calibration code
C      XBPVER   R        Bandpass (BP) table to apply
C      XSMOTH   R(3)     Spectral smoothing parameters.
C      XINT     R        Integration time of data in seconds.
C      XBAND    R        Selected bandwidth (Hz)
C      XFREQ    R        Selected frequency (Hz)
C      XFQID    R        Selected FQ id.
C   Outputs:
C      SCRNO    I        Scratch file number in DFIL.INC common.
C      IRET     I        Return code, 0=>OK else failed.
C-----------------------------------------------------------------------
      INTEGER   IDISK, ICNO, CATIN(256), SCRNO, IRET
C
      INTEGER   XBUFSZ
C                                       XBUFSZ = buffer size (words)
      PARAMETER (XBUFSZ=4096)
      INTEGER    IROUND, BUFSZ, DISK, LUN, I, INVER, OUTVER, LUNI, LUNO
      LOGICAL    MATCH
      REAL       BUFFER(XBUFSZ), RPARM(X), VIS(3,X)
      INCLUDE 'INCS:PUVD.INC'
      INCLUDE 'PARMS.INC'
      INCLUDE 'INCS:DSEL.INC'
      INCLUDE 'INCS:DMSG.INC'
      INCLUDE 'INCS:DHDR.INC'
      INCLUDE 'INCS:DUVH.INC'
C-----------------------------------------------------------------------
C                                       Initialize DSEL.INC
      CALL SELINI
C                                       Put selection criteria into
C                                       DSEL.INC
C                                       Select file
      UNAME = NAMEIN
      UCLAS = CLAIN
      UDISK = DISKIN
```

```
      USEQ = SEQIN
C                                        Select data
      DO 10 I = 1,30
         SOURCS(I) = XSOUR(I)
 10      CONTINUE
      SELQUA = IROUND (XQUAL)
      SELCOD = XCALCO
      CALL RCOPY (8, XTIME, TIMRNG)
      STOKES = XSTOK
      BCHAN = IROUND (XBCHAN)
      ECHAN = IROUND (XECHAN)
      BIF = IROUND (XBIF)
      EIF = IROUND (XEIF)
C                                        Select flagging
      FGVER = IROUND (XFLAG)
C                                        Select calibration
      DOCAL = XDOCAL .GT. 0.0
      DOPOL = XDOPOL .GT. 0.0
      SUBARR = IROUND (XSUBA)
      CLUSE = IROUND (XGUSE)
      BLVER = IROUND (XBLVER)
      DOBAND = IROUND (XDOBND)
      BPVER = IROUND (XBPVER)
C                                         Spectral smoothing
      DO 20 I = 1, 3
         SMOOTH(I) = IROUND (XSMOTH(I))
 20      CONTINUE
      DOSMTH = SMOOTH(1) .GT. 0
C
      DXTIME = XINT / 86400.0
C                                        Freq id
      IF (XBAND.GT.0.0) SELBAN = XBAND
      IF (XFREQ.GT.0.0) SELFRQ = XFREQ
      FRQSEL = IROUND (XFQID)
      IF (FRQSEL.EQ.0) FRQSEL = -1
      LUN = 28
      CALL FQMATC (IDISK, ICNO, CATIN, LUN, SELBAN, SELFRQ, MATCH,
     *   FRQSEL, IRET)
      IF (.NOT.MATCH) THEN
         IRET = 5
         MSGTXT = 'NO MATCH TO SELBAND/SELFREQ ADVERBS - CHECK INPUTS'
         GO TO 990
         END IF
      IF (IRET.GT.0) GO TO 999
C                                        Init call to UVGET, note that
C                                        RPARM and VIS are not used here.
      CALL UVGET ('INIT', RPARM, VIS, IRET)
      IF (IRET.NE.0) GO TO 999
C                                        Call CALCOP to process file.
      DISK = 0
      SCRNO = 0
      BUFSZ = XBUFSZ * 2
      CALL CALCOP (DISK, SCRNO, BUFFER, BUFSZ, IRET)
      IF (IRET.NE.0) GO TO 999
```

```
C                                       Copy FQ table
      INVER = 1
      OUTVER = 1
      LUNO = 28
      LUNI = 29
      CALL CHNCOPL, INVER, OUTVER, LUNO, LUNI, DISKIN, DISK, IUCNO,
     *     SCRNO, CATUV, CATBLK, BIF, EIF, FRQSEL, BUFFER, UBUFF, PBUFF,
     *     IREF
      GO TO 999
C                                       Error message
 990  CALL MSGWRT (8)
C
 999  RETURN
C--------------------------------------------------------------------
      END
```

16.12 Single Dish Data

"Single Dish" data consists of sky brightness measurments or spectra made at arbitrary positions on the sky. The format of this data is very similar to that of interferometry data with sky brightness and the baseline subtracted constituting the "real" and "imaginary" parts. A detailed description of the single dish format is given in the chapter on disk I/O. Access to single dish data is through routine SDGET which can optionally calibrate amplitudes and/or positions and remove baselines. Many of the single dish calibration and editing routines are the same as used for interferometery data. This is especially true for flagging (FG table) and indexing (NX table). For purposes of selection and editing the "Feed Number" takes the place of "baseline" code in interferometer data. The discussion of editing for interferometry data also applied to single dish data. Compression of single dish data is not supported. The structure of the single dish calibration routines is shown in the following diagram:



Single Dish Data Calibration

The functions of the single dish calibration routines is briefly described in the following and detailed descriptions are given at the end of this chapter.

1. SDGET : the top level data access routine. SDGET controls the initialization of the various files and arrays, calls DGETSD to process the data, and closes the necessary files when done.
2. SOUFIL : this routine converts the list of sources to source numbers, which are filled into a common.
3. GACSIN : initializes the calibration (CS) table to be applied to the data.
4. DGINIT : sets up to translate data (e.g., RR, LL to IPOL), if necessary.
5. FLGINI : initializes reading flagging (FG) table.
6. INDXIN : initializes the index table I/O and finds the first relevant index record.
7. DGHEAD : fills the catalog header record to correspond to the output data.
8. DGETSD : reads data, applies flagging and calibration, and translates polarization.
9. VISCNT : estimates the number of visibility records requested from the index table.
10. DATFLG : flags data.
11. NXTFLG : manages the internal arrays containing the currently active flagging criteria.
12. DCALSD : calibrates data.
13. SDCSET : interpolates calibration table entries to current time.
14. SDCGET : finds calibration table entries on both sides of the current time and reads the values into a common array.
15. DGGET : translates polarization.

16.13 Text of INCLUDE files

There are several types of INCLUDE file which are distinguished by the first character of their name. Different INCLUDE file types contain different types of Fortran declaration statements as described in the following list.

- Pxxx.INC. These INCLUDE files contain declarations for parameters and the PARAMETER statements.
- Dxxx.INC. These INCLUDE files contain Fortran type (with dimension) declarations, COMMON and EQUIVALENCE statments.
- Vxxx.INC. These contain Fortran DATA statements.
- Zxxx.INC. These INCLUDE files contain declarations which may change from one computer or installation to another.

16.13.1 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, INTFM, 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.
```

16.13.2 PUVD.INC

```
C                                                       Include PUVD
C                                        Parameters for uv data
      INTEGER   MAXANT,  MXBASE,  MAXIF, MAXFLG, MAXFLD, MAXCHA
C                                        MAXANT = Max. no. antennas.
      PARAMETER (MAXANT=45)
C                                        MXBASE = max. no. baselines
      PARAMETER (MXBASE= ((MAXANT*(MAXANT+1))/2))
C                                        MAXIF=max. no. IFs.
      PARAMETER (MAXIF=15)
C                                        MAXFLG= max. no. flags active
      PARAMETER (MAXFLG=1000)
C                                        MAXFLD=max. no fields
      PARAMETER (MAXFLD=16)
C                                        MAXCHA=max. no. freq. channels.
      PARAMETER (MAXCHA=512)
C                                        Parameters for tables
      INTEGER MAXCLC, MAXSNC, MAXANC, MAXFGC, MAXNXC, MAXSUC,
     *   MAXBPC, MAXBLC, MAXFQC
C                                        MAXCLC=max no. cols in CL table
      PARAMETER (MAXCLC=41)
C                                        MAXSNC=max no. cols in SN table
      PARAMETER (MAXSNC=20)
C                                        MAXANC=max no. cols in AN table
      PARAMETER (MAXANC=12)
C                                        MAXFGC=max no. cols in FG table
      PARAMETER (MAXFGC=8)
C                                        MAXNXC=max no. cols in NX table
      PARAMETER (MAXNXC=7)
C                                        MAXSUC=max no. cols in SU table
      PARAMETER (MAXSUC=21)
C                                        MAXBPC=max no. cols in BP table
      PARAMETER (MAXBPC=14)
```

```
C                                       MAXBLC=max no. cols in BL table
      PARAMETER (MAXBLC=14)
C                                       MAXFQC=max no. cols in FQ table
      PARAMETER (MAXFQC=5)
C                                                       End PUVD.
```

16.14 Routines

16.14.1 BLREFM

Routine to change the format of the BL table from one containing 13/9 columns to the 14/10 columns needed by the addition of the FREQID column. NOTE: routine uses LUN 45 as a temporary logical unit number.

```
BLREFM (DISK, CNO, VER, CATBLK, LUN, IRET)
Inputs:
   DISK            I       Volume number
   CNO             I       Catalogue number
   VER             I       Version to check/modify
   CATBLK(256)     I       Catalogue header
   LUN             I       LUN to use
Output:
   IRET            I       Error, 0 => OK

Note, routine will leave no trace of its operation, i.e. BL table
will be closed on output and will have same number as one specified.
Difference will be only that number of columns has changed if that
is required.
```

16.14.2 BLSET

Fills current baseline correction table (BLFAC) by interpolation.

```
BLSET (TIME, IERR)
Inputs:
   TIME         R    Current time (of data) in days.
Inputs from DSEL.INC:
   GMMOD        R    Mean gain modulus correction, 0=>none.
   BLTAB(*,2)   R    Baseline table from BL table file.
                     The second dimension corresponds to the time
                     before and after the current data time.
                     Indexing scheme: an entry defined by ant1<ant2
                     starts in element:
      lentry * (((ant1-1)*numant-((ant1+1)*ant1)/2 + ant2) - 1) + 1
                     where lentry = 2 * NUMPOL * (EIF-BIF+1)
                     An entry contains the values in order:
                     By IF (NUMIF)
                        By Polarization (NUMPOL)
                            Real part, imaginary part.
   BLTIM(3)     R    Time of two cal. entries; third value is
                      time of current values.
   IBLP1        I    Pointer in BLTAB, BLTIM to previous time.
   IBLP2        I    Pointer in BLTAB, BLTIM to next time.
   DOBL         L    If true then use baseline correction table,
                     else initialize BLFAC.
```

```
Output:
   IERR          I    Return error code, 0=>OK else error.
Output to DSEL.INC:
   BLFAC(*)      R   Baseline dependent factors as (real, imag)
                     Includes GMMOD correction if necessary.
                     Addressing is like BLTAB.
                     Initialized to 1/GMMOD,0 if DOBL=.FALSE.
```

16.14.3 BPASET

Sets up the bandpass table array for use by DATBND.

```
BPASET (IERR)
    Inputs from DSEL.INC:
     BPBUFF(*)     I    BP table I/O TABIO buffer
     NBPINR        I    Number of BP records in file.
     NANTBP        I    Number of antennas
     NPOLBP        I    Number of IFs per group (polarizations)
     NIFBP         I    Number of IFs.
     NCHNBP        I    Number of channels
  Output:
     IERR          I    Return error code 0=>OK, else failed.
  Output to DSEL.INC:
     TIMENT(MAXBP)          R    An index array recording the times of
                                 BP records in the scratch file
     ANTENT(MAXBP,MAXANT)   I    The corresponding antenna index.
     MXANUM                 I    The maximum antenna number
```

16.14.4 BPGET

Gets next set of bandpass data in array PBUFF. Depending on the value of DOBAND will

1. extract data for antenna from scratch file containing averaged data.
2. extract bandpass data closest in time to visibility data
3. will do a linear interpolation in time between the entries.

If options (2) or (3) are selected the I/O rate will be very high and will slow the program down tremendously.

```
BPGET (TIME, IA1, IA2, IERR)
Inputs:
   TIME           R       Current time of data (days)
   IA1            I       First antenna to be selected
   IA2            I       Second antenna to be selected
Inputs from common /...../
   DOBAND         I       Bandpass selection option
   TIMENT
   ANTENT
Output:
   IERR           I       Return error code. 0=>OK, else error
Output to common /...../
   BPANT1         R(*)    Array containing bandpass spectrum
                          for IA1
   BPANT2         R(*)    Array containing bandpass spectrum
```

```
                        for IA2
   LBPTIM        R      Time of current calibration
   BPGOT(2)      I      Antenna numbers of present calibration.
```

16.14.5 BPREFM

Routine to change the format of the BP table from one containing 13/10 columns to the 14/11 columns needed by the addition of the FREQID column. NOTE: routine uses LUN 45 as a temporary logical unit number.

```
BPREFM (DISK, CNO, VER, CATBLK, LUN, IRET)
Inputs:
   DISK           I      Volume number
   CNO            I      Catalogue number
   VER            I      Version to check/modify
   CATBLK(256)    I      Catalogue header
   LUN            I      LUN to use
Output:
   IRET           I      Error, 0 => OK

Note, routine will leave no trace of its operation, i.e. BP table
will be closed on output and will have same number as one specified.
Difference will be only that number of columns has changed if that
is required.
```

16.14.6 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
```

16.14.7 CALREF

Subroutine to adjust the reference antenna in a uv data file. The Table is first read to find all data relating ANT and REFAN. These data are then smoothed and the resulting ANT-REFAN are used in a second pass through the table to adjust data using ANT as a reference antenna to REFAN as the reference antenna. Several work arrays are passed which are used for storing, smoothing and interpolating data. The table should already be open and BUFFER should be the buffer used by TABINI (or other table opening routines).

```
CALREF (ANT, REFAN, SUB, KOLS, BUFFER, SMOTIM, MAXTIM,
  *   FREQ, WRKTIM, WORK1, WORK2, WORK3, WORK4, WORK5, IRET)
Input:
   ANT      I     Old reference antenna
   REFAN    I     New reference antenna
   SUB      I     Subarray desired
   KOLS     I(9)  Array of TABIO column pointers in order:
                  antenna, ref. antenna, subarray, weight, time,
                  real, imag, delay, rate.
   BUFFER   I(*)  Buffer for TABIO use; table must already be
                  open
   SMOTIM   R(3)  Boxcar averaging times (days) 1=phase, 2=delay
                  3 = rate
   MAXTIM   I     Maximum number of times (dim of WRKTIM etc)
   FREQ     R     Frequency of observation (Hz)
Output:
   WRKTIM   R(*)  Large work array.
   WORK1    R(*)  Large work array, same size as WRKTIM
   WORK2    R(*)  Large work array, same size as WRKTIM
   WORK3    R(*)  Large work array, same size as WRKTIM
   WORK4    R(*)  Large work array, same size as WRKTIM
   WORK5    R(*)  Large work array, same size as WRKTIM
   IRET     I     Return code 0=OK, else failed.
```

16.14.8 CGASET

Gets next set of calibration data in CURCAL, does linear interpolation in time between time entries in CALTAB. Calls BLSET to fill or initialize BLFAC and apply and baseline dependent calibration then enters any rate corrections. If the preceeding or following entry is for the current source then only entries for that source is used.

```
CGASET (TIME, IERR)
Inputs:
   TIME     R         Current time (of data) in days.
Inputs from DSEL.INC:
   CALTAB   R(*,2)    Cal. table from gain table file
                      Values in order:
                      By antenna (NUMANT)
                         By IF (NUMIF)
```

```
                              By Polarization (NUMPOL)
                                  Real part, imaginary part,
                                  group delay, phase rate, ref. ant.
   LCLTAB  I        Number of values in CALTAB per entry (5)
   IFRTAB  R(*,2)   Ionospheric Faraday rotation from cal table,
                    listed by antenna number
   CALTIM  R(3)     Time of two cal. entries; third value is
                    time of current values.
   CIDSOU  I(2)     Previous/next source ID number using ICALPn as
                    a pointer.
   CURSOU  I        Current source ID number.
   ICALP1  I        Pointer in CALTAB, CALTIM to previous time.
   ICALP2  I        Pointer in CALTAB, CALTIM to next time.
   IBLP1   I        Pointer in BLTAB, if < 0 then BLFAC needs
                    to be initialized by BLSET.
   DOCAL   L        If true then apply antenna based calibation.
   DOBL    L        If true then apply baseline based calibation.
   RATFAC  R(*)     IF scaling factor to convert s/s to rad/day
   DELFAC  R(*)     IF scaling factor to convert s to rad/channel
   DXTIME  R        Integration time of the data in days.
Output:
   IERR    I        Return error code, 0=>OK else error.
Output to DSEL.INC:
   LCALTM  R        Time of current calibration.  If gain file
                    is exhausted then 1.0E20 is returned.
   CURCAL  R(*)     Current calibration information.
                    Values in order:
                    By antenna (NUMANT)
                       By IF (EIF-BIF+1)
                          By Polarization (NUMPOL)
                              Real part, imaginary part,
                              cos(delta), sin(delta), rate
                    Where delta is the phase change between
                    channels and rate is the fringe rate in
                    radians/day
   IFR     R(*)     Current ionospheric rotation measure for each
                    antenna
   BLFAC   R(*)     Baseline dependent factors.
                    Initialized to 1/GMMOD,0 if DOBL=.FALSE.
                    Indexing scheme: an entry defined by ant1<ant2
                    starts in element:
      lentry * (((ant1-1)*numant-((ant1+1)*ant1)/2 + ant2) - 1) + 1
                    where lentry = 2 * NUMPOL * (EIF-BIF+1)
                    An entry contains the values in order:
                    By IF (NUMIF)
                       By Polarization (NUMPOL)
                           Real part, imaginary part.
```

16.14.9 CLREFM

Routine to change the format of the CL table from one containing 39/24 columns to the 41/26 columns needed by the addition of the FREQID and IFR columns. NOTE: routine uses LUN 45 as a temporary logical unit number.

```
CLREFM (DISK, CNO, VER, CATBLK, LUN, IRET)
Inputs:
   DISK     I       Volume number
   CNO      I       Catalogue number
   VER      I       Version to check/modify
   CATBLK   I(256)  Catalog header
   LUN      I       LUN to use
Output:
   IRET     I       Error, 0 => OK

Note, routine will leave no trace of its operation, i.e. CL table
will be closed on output and will have same number as one specified.
Difference will be only that number of columns has changed if that
is required.
```

16.14.10 CLUPDA

Concatanates all SN tables and rereferences to the same reference antenna. Then if SNSMTH is true the SN table will be smoothed. If DOAPPL (in common) is true then the SN table is applied to the specified CL table. Leaves the output table sorted in time-antenna order.

```
CLUPDA (SINGLE, SNSMTH, REFA, IERR)
Inputs:
  SINGLE    L    If true then the uv data is a single source file
                 and only SN tables will be processed.
  SNSMTH    L    If true then smooth SN tables.
  REFA      I    The desired reference antenna, 0=most used.
Inputs from DSEL.INC
   DOAPPL       L    If true then apply SN tables to the CL table.
   CLVER        I    Input Cal (CL) file version number.
                     For single source files the input SN table.
   CLUSE        I    Cal file version number to put smoothed gains
                     into and use for calibration.
                     For single source files the output SN table.
   TSTART       R    First time to process (days) (no default)
   TEND         R    Last time to process (days) (no default)
Output:
   IERR         I    Return code, 0=>OK, otherwise failed.
Useage notes:
   1) Uses CLBUFF, BLFAC, CALTAB, BLTAB and UBUFF from /DSEL.INC/.
   2) Sorts the relevant tables.
```

16.14.11 CSLGET

Sets up for interpolation in cal (CL or SN) table, reads values from cal table. Assumes only valid, selected data in open cal table.

```
CSLGET (TIME, IERR)
Inputs:
   TIME    R         Current data time.
Inputs from DSEL.INC:
   CLBUFF  I(*)      Cal table I/O TABIO buffer
   ICLRNO  I         Current cal record number
   NCLINR  I         Number of cal records in file.
```

```
   NUMANT  I         Number of antennas
   NUMPOL  I         Number of IFs per group (polarizations)
   NUMIF   I         Number of IFs.
   GMMOD   R         Mean cal modulus
Output:
   IERR    I         Return error code 0=>OK, else failed.
Output to DSEL.INC
   CALTAB  R(*,2)    Cal. table from cal table file
                     Values in order:
                     By antenna (NUMANT)
                        By IF (NUMIF)
                           By Polarization (NUMPOL)
                               Real part, imaginary part,
                               group delay, phase rate, ref. ant.
   LCLTAB  I         Number of values in CALTAB per entry (5)
   IFRTAB  R(*, 2)   Ionospheric Faraday rotation measure from
                     cal table. Values listed by antenna.
   CALTIM  R(3)      Time of two cal. entries; third value is
                     time of current values.
   CIDSOU  I(2)      Previous/next source ID number using ICALPn as
                     a pointer.
   ICALP1  I         Pointer in CALTAB, CALTIM to previous time.
   ICALP2  I         Pointer in CALTAB, CALTIM to next time.
```

16.14.12 DATBND

Routine which applies the bandpass correction.

```
DATBND (TIME, IA1, IA2, VIS, IERR)
Inputs:
   TIME     R      Time of visibility data (in days)
   IA1      I      Antenna number 1
   IA2      I      Antenna number 2
   VIS      R(*)   Array of visibility data
Inputs from common:
   PBUFF    R(*)   Large array containing bandpass spectra for
                   several antennas
   ANTPNT   I(2)   Pointer giving the start address of the specified
                   antennas bandpass spectra within PBUFF
Outputs:
   VIS      R(*)   Array of corrected visibility data
   IERR     I      If = 0, all OK,
                   If > 0, error returned from BPGET
Output to common:
   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)

NOTE: This routine applies the bandpass correction for formulae:

(1) Cross-power:   Scorr   =          Sobs
                                 --------------------
                             SQRT (Sant_1 * Sant_2)
```

```
(2) Total-power:   Scorr        (Son / Soff) - 1.0
```

16.14.13 DATCAL

Applies calibration to data.

```
DATCAL (IA1, IA2, TIME, VIS, DROP, IERR)
Inputs:
   IA1    I      First antenna number
   IA2    I      Second antenna number
   TIME   R      Time of record (days)
   VIS    R(*,*) Input visibility array (not yet converted to
                 output form.
Inputs from DSEL.INC:
   DOCAL   L     If true do antenna calibration.
   DOBL    L     If true do baseline calibration.
   DOWTCL  L     If true calibrate weights.
   CURCAL  R(*)  Current calibration information.
                  Values in order:
                  By antenna (NUMANT)
                     By IF (EIF-BIF+1)
                        By Polarization (NUMPOL)
                            Real part, imaginary part,
                            cos(delta), sin(delta), rate
                  Where delta is the phase change between
                  channels and rate is the fringe rate in
                  radians/day
   LCUCAL   I     Number of values in CURCAL per entry (5)
   POLOFF   I(4,2)Offsets from the beginning of an IF entry in
                  CURCAL for a given polarization.  The first
                  dimension is the polarization pixel number and
                  the second is the antenna number of a baseline
                  (e.g. first or second = 1 or 2).
   CALTIM   R(3)  Time of two cal. entries; third value is
                  time of current values.
   LCALTM    R    Time of current calibration.
   BLFAC     R(*) Baseline dependent factors including GMMOD.
                  Indexing scheme: an entry defined by ant1<ant2
                  starts in element:
      lentry * (((ant1-1)*numant-((ant1+1)*ant1)/2 + ant2) - 1) + 1
                    where lentry = 2 * NUMPOL * (EIF-BIF+1)
                    An entry contains the values in order:
                    By IF (NUMIF)
                       By Polarization (NUMPOL)
                           Real part, imaginary part.
                     Applied only to cross corelation data.
  Output:
   DROP    L      True if data all flagged.
   IERR    I      Return code, 0=OK, else CGASET error number.
  Output to common:
   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)
```

16.14.14 DATFLG

Flags data specified in flagging table

```
DATFLG (RPARM, VIS, DROP, IERR)
Inputs:
   RPARM(*)    R    Random parameter array
   VIS(3,*)    R    Visibility array
Inputs from common /CFMINF/:
   CURSOU      I    Current source number
   NUMFLG      I    Number of flagging entries.
   TMFLST      R    Time of last visibility for which flagging
                    was checked.
   FLGSOU(*)   I    Source id numbers to flag, 0=all.
   FLGANT(*)   I    Antenna numbers to flag, 0=all.
   FLGBAS(*)   I    Baseline (A1*256+A2) numbers to flag, 0=all.
   FLGSUB(*)   I    Subarray numbers to flag, 0=all.
                    Following should have defaults filled in.
   FLGBIF(*)   I    First IF to flag.
   FLGEIF(*)   I    Highest IF to flag.
   FLGBCH(*)   I    First channel to flag.
   FLGECH(*)   I    Highest channel to flag.
   FLGPOL(4,*)L     Flags for the polarizations, should correspond
                    to selected polarization types.
Output:
   RPARM(*)    R    Random parameter array
   VIS(3,*)    R    Visibility array
   DROP        L    True if data all flagged.
   IERR        I    Return code, 0=OK, else NXTFLG error number.
```

16.14.15 DATGET

Reads next selected data record. Applies calibration and editing.

```
DATGET (RPARM, VIS, TIMLST, IERR)
Inputs from DSEL.INC:
   INXRNO       I    Current INDEX file record number.
                     If .LT. 0 then there is no index file.
   NINDEX       I    Number of entries in the index table
   NXKOLS(7)    I    Pointer array for index records.
   ANTENS(50)   I    List of antennas selected, 0=>all,
                     any negative => all except those specified
   NANTSL       I    Number of antennas selected/excluded in ANTENS
                     0 = All included.
   DOAWNT       L    If .TRUE. then antennas in ANTENS included.
                     If .FALSE. then excluded.
   SUBARR       I    Subarray number desired, 0=>any.
   NSOUWD       I    Number of sources specified.
   DOSWNT       L    If true sources specified are included
                     else excluded.
   SOUWAN(30)   I    List or source numbers from source file.
   DOACOR       L    If true, pass autocorrelations.
```

```
    DOXCOR       L    If true, pass crosscorrelations.
    TRANSL       L    If true, translate data to requested stokes.
    JADR(2,*)    I    Table to translate input data to output vis.
    TSTART       R    Start time in days.
    TEND         R    End time in days.
    UVRA(2)      R    UV range (wavelength squared)
    NPRMIN       I    No. random parameters in the input data
    NRPARM       I    No. random parameters in output data.
    LRECIN       I    Length of input record in words.
    KLOCSU       I    Source number pointer in input data.
    KLOCWT       I    Weight pointer for compressed data.
    NDECMP       I    Number of entries in DECMP
    DECMP    I(2,*)   (1,*) = number of packed correlator values
                      (2,*) = 0-rel offset in vis data.
                      (from beginning of vis data NOT ran. parms.)
    DOFQSL       L    If true, FREQSEL random parameter is present
    FRQSEL       I    FQ entry to pass.
  Input/output:
    TIMLST       R    Time of last record
  Output:
    RPARM(*)     R    Random parameter array
    VIS(3,*)     R    Visibility array
    IERR         I    Return code, 0=OK, else UVDISK error number.
 Output to common /DSEL.INC/:
    FSTVIS       I    First word pointer of current buffer.
    LSTVIS       I    Last word pointer of current buffer.
    CURSOU       I    Current source number.
```

16.14.16 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.
```

16.14.17 DGHEAD

Corrects CATBLK in common /MAPHDR/ to correspond to UVGET output data. If only one output source is specified then the information about that source is filled in from the source file (if any).

```
DGHEAD
Inputs from include DSEL.INC:
   BCHAN   I       First channel desired.
   ECHAN   I       Last channel desired.
   BIF     I       First IF desired.
   EIF     I       Last IF desired.
   PMODE   I       Polarizarion mode (see DGINIT for codes)
                   0 => same Stokes' as in input.
   NSOUWD  I       Number of sources specified.
   DOSWNT  L       If true sources specified are included
                   else excluded.
   SOUWAN  I(30)   List or source numbers from source file.
Input/Output in common /MAPHDR/:
   CATBLK  I(256)  Uvdata catalog header record.
```

16.14.18 DGINIT

Sets up tables for selecting data from vis. record. Checks if requested data in data base. Requires catalog header record in common /MAPHDR/ and setup of common /UVHDR/ 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.
```

16.14.19 GACSIN

Single dish calibration routine - Initializes CS file, and prepares table to be applied. If there is no CS file DOCAL is set to .FALSE.

```
GACSIN (IERR)
Inputs from DSEL.INC:
   CLUSE    I   CS file version number to initialize.
Output:
   IERR     I   Return code, 0 => ok, otherwise CS table exists but
                   cannot be read.
Output to DSEL.INC:
   ICLRNO   I   Current CS record number
   NCLINR   I   Number of gain records in file.
   NUMPOL   I   Number of polarizations
   NUMIF    I   Number of IFs.
```

16.14.20 GAININ

Initializes Cal file, and prepares gain table to be applied. If there is no CL file DOCAL is set to .FALSE. For single source data files an SN table will be used rather than a CL table. Opens gain (CL or SN) and baseline (BL) tables if necessary.

```
GAININ (IERR)
Inputs from common /DSEL.INC/
   CLUSE        I    Cal file version number (CL or SN) to
                     initialize.
Output:
   IERR         I    Return code, 0=>OK, otherwise CL table
                     exists but cannot be read.
Output to common /DSEL.INC/:
   RATFAC(*)    R    IF scaling factor to convert s/s to rad/day
   DELFAC(*)    R    IF scaling factor to convert s to rad/channel
   LAMSQ(*, *)  R    Table of wavelength squared (in meters squared)
                     for each channel (first axis) and IF (second
                     axis)
   ICLRNO       I    Current cal record number
   NCLINR       I    Number of gain records in file.
   NBLINR       I    Number of BL records in file.
   NUMANT       I    Number of antennas
   NUMPOL       I    Number of polarizations
   NUMIF        I    Number of IFs.
   GMMOD        R    Mean gain modulus
```

16.14.21 LXYPOL

Fills polarization correction table from info in AN table for Linear polarization feeds (XY).

```
LXYPOL (PANGLE, IERR)
Inputs:
```

```
   PANGLE  R(*)      Parallactic angles of the antennas (Rad)
Output:
   IERR    I         Return error code, 0=>OK else error.
                     1=table too small, 2=multiple subarrays,
                     10 = unknown polarization parameterization,
                     otherwise GETANT error.
Output to common /DSEL.INC/:
   PARTIM  R         Time of current parallactic angles. (-1.0E10)
   PARSOU  I         Source ID for current parallactic angles. (-10)
   POLCAL  R(2,*)    Polarization correction
                     Values in order:
                     By baseline
                        By IF (EIF-BIF+1)
                           A 4x4 complex matrix to be multiplied by
                               the observed polarization vector
                               (RR,LL,RL,LR) to produce the
                               corrected data.
                     Indexing scheme: an entry defined by ant1<ant2
                     starts in element:
      (((ant1-1)*numant-((ant1+1)*ant1)/2 + ant2) - 1) + 1
```

16.14.22 NXTFLG

Updates flagging tables in common fron an FG table.

```
NXTFLG (TIME, IERR)
Inputs:
   TIME          R    Current time (days) for flag entries
Inputs from common /CFMINF/(INCLUDEs /DSEL.INC):
   NUMFLG        I    number of current FLAG entries.
   FGKOLS(8)     I    The column pointer array in order, SOURCE,
                      SUBARRAY, ANTS, TIMERANG, IFS, CHANS, PFLAGS,
                      REASON
   FGNUMV(8)     I    Element count for each column
   IFGRNO        I    Current FLAG file record.
Output to common /DSEL.INC/:
   NUMFLG     I    Number of flagging entries.
   TMFLST     R    Time of last visibility for which flagging
                   was checked.
   FLGSOU(*)  I    Source id numbers to flag, 0=all.
   FLGANT(*)  I    Antenna numbers to flag, 0=all.
   FLGBAS(*)  I    Baseline (A1*256+A2) numbers to flag, 0=all.
   FLGSUB(*)  I    Subarray numbers to flag, 0=all.
                   Following should have defaults filled in.
   FLGBIF(*)  I    First IF to flag.
   FLGEIF(*)  I    Highest IF to flag.
   FLGBCH(*)  I    First channel to flag.
   FLGECH(*)  I    Highest channel to flag.
   FLGPOL(4,*)L    Flags for the polarizations, should correspond
                   to selected polarization types.
   FLGTMD(*)  R    End time of flag.
Output:
   IERR       I    Return code, 0=OK, else TABIO error number.
```

16.14.23 POLSET

Fills polarization correction table from info in AN table.

```
POLSET (IERR)
Inputs from common:
   STMEPL   R(2,*)   Feed real/elipticity (poln, IF)
   STMORI   R(2,*)   Feed imag/orientation (poln, IF)
   STMPST   C*8      Feed solution type:
                        'APPROX  ' => linear approximation
                        'ORI-ELP ' => orientation-ellipticity
                        'X-Y LIN ' => lin. approx. for lin.
                           polarized (X-Y) data.
  Output:
   IERR     I        Return error code, 0=>OK else error.
                     1=table too small, 2=multiple subarrays,
                     10 = unknown polarization parameterization,
                     otherwise GETANT error.
  Output to DSEL.INC:
   PARTIM   R        Time of current parallactic angles. (-1.0E10)
   PARSOU   I        Source ID for current parallactic angles. (-10)
   POLCAL   R(2,*)   Polarization correction
                     Values in order:
                     By baseline
                        By IF (EIF-BIF+1)
                           A 4x4 complex matrix to be multiplied by
                              the observed polarization vector
                              (RR,LL,RL,LR) to produce the
                              corrected data.
                    Indexing scheme: an entry defined by ant1<ant2
                    starts in element:
      (((ant1-1)*numant-((ant1+1)*ant1)/2 + ant2) - 1) + 1
```

16.14.24 SCLOAD

Copies part of a 'BP' scratch file to a second scratch file for more efficient I/O.

```
SCLOAD (TIME1, TIME2, LUNOP, FINDOP, CREATE, IERR)
Input:
   TIME1        R    Time label of first section of data to be
                     transferred.
   TIME2        R    Time label of second section of data to be
                     transferred. If < 0 is ignored.
   LUNOP        I    LUN of secondary scratch file.
   CREATE       L    If true must create new scratch file.
Output:
   FINDOP       I    FTAB pointer for secondary scratch file.
   IERR         I    Return error code 0=>OK, else failed.
```

16.14.25 SCINTP

When the interpolation mode of bandpass calibration is specified this routine takes the secondary scratch file created by SCLOAD and interpolates in time between the two entries and writes a scratch file containing the interpolated data. To save on I/O and interpolation time the third scratch file is updated at 0.2 of the interval between the two entries.

```
SCINTP (TIME, LUNIN, FINDIN, LUNOP, FINDOP, IERR)
Input:
   TIME      R   Current time of data (days)
   LUNIN     I   LUN of secondary scratch file.
   FINDIN    I   FTAB pointer for second scratch file.
   LUNOP     I   LUN of third (interpolated) scratch file.
Output:
   IERR      I   Return error code 0=>OK, else failed.
```

16.14.26 SDCGET

Single dish calibration routine. Sets up for interpolation in cal (CS) table, reads values from cal table. Assumes only valid, selected data in open cal table. Uses calls to TABIO directly for efficiency.

```
SDCGET (TIME, IERR)
Inputs:
   TIME          R    Current data time.
Inputs from common /DSEL.INC/:
   CLBUFF(*)     I    Cal table I/O TABIO buffer
   ICLRNO        I    Current cal record number
   NCLINR        I    Number of cal records in file.
   NUMANT        I    Number of beams
   NUMPOL        I    Number of IFs per group (polarizations)
   NUMIF         I    Number of IFs.
Output:
   IERR          I    Return error code 0=>OK, else failed.
Output to common /DSEL.INC/:
   CALTAB(*,2)   R    Cal. table from cal table file
                      Values in order:
                      By beam (NUMANT)
                         By IF (NUMIF)
                            By Polarization (NUMPOL)
                                Amplitude factor,
                                offset (before factor)
                                RA correction
                                Dec correction.
   LCLTAB        I   Number of values in CALTAB per entry (4)
   CALTIM(3)     R    Time of two cal. entries; third value is
                      time of current values.
   ICALP1        I    Pointer in CALTAB, CALTIM to previous time.
   ICALP2        I    Pointer in CALTAB, CALTIM to next time.
```

16.14.27 SDCSET

Single dish calibration routine: Gets next set of calibration data in CURCAL, does linear interpolation in time between time entries in CALTAB.

```
SDCSET (TIME, IERR)
Inputs:
   TIME      R         Current time (of data) in days.
Inputs from common /DSEL.INC/:
   CALTAB    R(*,2)    Cal. table from gain table file
                       Values in order:
                       By beam (NUMANT)
```

```
                        By IF (NUMIF)
                           By Polarization (NUMPOL)
                               Amplitude factor,
                               offset (before factor)
                               RA correction
                               Dec correction.
   LCLTAB   I          Number of values in CALTAB per entry (4)
   CALTIM   R(3)       Time of two cal. entries; third value is
                       time of current values.
   ICALP1   I          Pointer in CALTAB, CALTIM to previous time.
   ICALP2   I          Pointer in CALTAB, CALTIM to next time.
Output:
   IERR     I          Return error code, 0=>OK else error.
Output to common /DSEL.INC/:
   LCALTM   R          Time of current calibration.  If gain file
                       is exhausted then 1.0E20 is returned.
   CURCAL   R(*)       Current calibration information.
                       Values in order:
                       By beam (NUMANT)
                          By IF (EIF-BIF+1)
                             By Polarization (NUMPOL)
                                 Amplitude factor,
                                 offset (before factor)
                                 RA correction
                                 Dec correction.
```

16.14.28 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/ common 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 DSEL.INC  (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 /DSEL.INC/: 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 catalgue 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.
```

16.14.29 SETSM

SETSM determines the type of spectral smoothing to be applied and sets up the look up table to do it. The actual smoothing is done in routine SMOSP

```
SETSM (IRET)
Inputs: (via common)
   SMOOTH  R(3)   Array containing smoothing parms
                  SMOOTH(1) = type of function
                        (2) = width of function
                              in channels
                        (3) = support of function
                              in channels
                  Type of function supported are:
                         0 => no smoothing
                         1 => hanning
                         2 => gaussian
                         3 => boxcar
                         4 => sin(x)/x
Output:
  IRET      I     Return error code, 0=>OK, otherwise abort.
```

16.14.30 SELINI

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

```
SELINI
Outputs via DSEL.INC  (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)
```

16.14.31 SNREFM

Routine to change the format of the SN table from one containing 18/12 columns to the 20/14 columns needed by the addition of the FREQID and IFR columns. NOTE: routine uses LUN 45 as a temporary logical unit number.

```
SNREFM (DISK, CNO, VER, CATBLK, LUN, IRET)
Inputs:
   DISK     I       Volume number
   CNO      I       Catalogue number
   VER      I       Version to check/modify
   CATBLK   I(256)  Catalogue header
   LUN      I       LUN to use
Output:
   IRET     I       Error, 0 => OK

Note, routine will leave no trace of its operation, i.e. SN table
will be closed on output and will have same number as one specified.
Difference will be only that number of columns has changed if that
is required.
```

16.14.32 SOUFIL

Fills in arrays of source numbers to be included or excluded; also checks antennas to be selected.

```
SOUFIL (IERR)
Inputs from include DSEL.INC:
   SOURCS(30)   C*16 Names of up to 30 sources, *=>all
                     First character of name '-' => all except those
                     specified.
   CALSOU(30)   C*16 Names of up to 30 calibrators,
                     '*' or blank =>all, first character of name '-'
                     => all except those specified.
   SELQUA       I    Source qualifiers to be selected, -1=>any.
                     Applied to both SOURCS and CALSOU.
   SELCOD       C*4  Calibrator codes to select.
                     '    '  => any,
                     '*   ' => any non blank calibrator code.
                     '-CAL' => blank only (no calibrators)
```

```
                    anything else => matching CALcodes.
                    Applied to SOURCS or CALSOU as controlled by
                    DOAPPL
   DOAPPL       L   If true then selection of the sources in
                    CALSOU is conditioned on SELCOD else
                    selection of SOURCS is conditioned on SELCOD.
   ANTENS(50)   I   List of antennas selected, 0=>all,
                    any negative => all except those specified
Output:
   IERR         I   Return code, 0=>OK, otherwise source file
                    exists but cannot be read.
                    1=TABIO problem, 2=no sources or calibrators
Output to DSEL.INC:
   NSOUWD       I   Number of sources included or excluded; if
                    0 all sources are included.
   DOSWNT       L   If .TRUE. then sources in SOUWAN are included
                    If .FALSE. then excluded.
   SOUWAN(30)   I   The source numbers of sources included or
                    excluded.
   SOUWTB(30)   I   The SoUrce table row numbers corresponding
                    to SOUWAN.
   NCALWD       I   Number of calibrators included or excluded.
   DOCWNT       L   If .TRUE. then calibrators in CALWAN are
                    included, if .FALSE. then excluded.
   CALWAN(30)   I   The source numbers of calibrators included or
                    excluded.
   CALWTB(30)   I   The SoUrce table row numbers corresponding
                    to CALWAN.
   NANTSL       I   Number of antennas selected/excluded in ANTENS
                    0 = All included.
   DOAWNT       L   If .TRUE. then antennas in ANTENS included.
                    If .FALSE. then excluded.
Note: also uses FGBUFF and UBUFF from /SELCAL/
```

16.14.33 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 DSEL.INC  (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.
```

16.14.34 VISCNT

Counts the number of visibility records for the sources selected with the given time range in DSEL.INC. The index (NX) file should already be open.

```
VISCNT (IERR)
Inputs from commons in DSEL.INC
   TSTART        R    Start time (days)
   TEND          R    End time (days)
   INXRNO        I    Current INDEX file record number.
                      If .LT. 0 then there is no index file.
   NINDEX        I    Number of entries in the index table
   NXBUFF(*)     I    TABIO buffer for INDEX table.
   NSOUWD        I    Number of sources specified.
   DOSWNT        L    If true sources specified are included
                      else excluded.
   SOUWAN(30)    I    List or source numbers from source file.
   SUBARR        I    Selected subarray, 0=>all
   FRQSEL        I    Selected freq id, 0=>all
Output:
   IERR          I    Return code, 0=>OK, 1=>no data, 2=>error.
Output to commons in DSEL.INC
   CATBLK(KIGCN)  I    Visibility count. (CATBLK)
```

Appendix C

Details of AIPS files

C.1 Introduction

This appendix contains the detailed descriptions of the various files used in AIPS both for user data and AIPS system functions. The details of the structure, contents and usage are given for each of these file types. These files are organized into 4 categories in this appendix; 1) AIPS system files, 2) user data files, 3) user tables and 4) task specific tables.

C.2 AIPS System files

In this section the files that the AIPS system uses are described.

C.2.1 Accounting (AC) file

Overview

The account file is used to accumulate information on the use of the AIPS system.

Details: All user programs in the AIPS system are required to place information on task name, user number, start date/time, elapsed time, and CPU time for each execution in the accounting file. This information is used by AIPS System Managers for a variety of purposes including the identification of problem programs and the maintaining of financial and usage accounting.

Name: The file name is ACr00000 where r is the system format revision number. It is a permanent file and must be created at AIPS system installation. A good initial size is about 100 blocks, but the file will expand itself one granule at a time as necessary.

File Structure

The file consists of logical records each 9 words long. There are 28 logical records in each 256-word physical record. The first logical record contains

FIELD	TYPE	DESCRIPTION
1	I	Maximum logical record # now recorded
2	I	Max logical record # which will fit in file
3	I	number of words / logical record
4	I	Number of logical records / physical record
5	I(5)	Reserved

where fields 3 and 4 are clearly just for convenience. All other logical records have the structure:

FIELD	TYPE	DESCRIPTION
1	H(2)	Task root name (4 characters/word)
2	I	POPS number + 100*i (i=1,2,3 for OLD,NEW,TST)
3	I	Logon user number
4	I(2)	Start time (packed YY/MM/DD, HH/MM/SS)
5	I	Total I/O count (when possible)
6	R	Total real time in seconds
7	R	Total CPU time in seconds

When a program initializes its entry in the file, field 6 is set to zero and field 7 is set to the current CPU time. When a program closes its entry, field 6 is reset to the difference in the current time and that in field 4 and field 7 is reset to the difference between the current CPU time and that previously in field 7. Thus, system overhead times are not fully charged to the program and programs which abort are identifiable by field 6 equal to zero. I/O count is handled like CPU time.

Usage Notes

No normal program other than those listed below should access this file. Of course, AIPS System Managers may need to construct additional, special account management programs.

Routines and commons for writing the account task data file

The primary routine for accessing the file is ACOUNT (IOP, buffer), where IOP = 2 means the entry is being closed and all other values of IOP mean the entry is being opened. The AIPS, AIPSC, AIPSB, and BATER programs call ACOUNT directly at appropriate times. Tasks must call GTPARM to obtain their full identity and their adverb values. Therefore, as soon as the identity is known, GTPARM calls ACOUNT. When tasks end, they must issue a close message, resume the initiating AIPSxx, and close the account entry. The subroutine DIETSK (RETCOD, RQUICK, buffer) has been constructed to perform all of these operations. The last executable statement in every task must be a call to DIETSK. Subroutines DIE and TSKEND perform this call for some tasks.

The common /MSGCOM/ (DMSG.INC) now carries a parameter, NACOUN, which gives the position in the accounting file being used by the current program. No routine, but ACOUNT, should change this parameter.

Routines and commons for reading the task data file

A service program, PRTAC, has been written for use by AIPS System Managers and other interested users. It can be run either as an AIPS task or as a stand-alone program. It prints the contents of the account file in the forms: (1) sequential listing, (2) totals by POPS number, (3) totals by user number, and (4) totals by program name (sorted by CPU and sorted by number of occurrences). PRTAC accumulates tasks and AIPS-like programs separately for types (2) and (3). The user may limit which of these displays he receives. He may also limit the summing and displays to a specific program name, a range of POPS numbers, a range of user numbers, only programs using more than x seconds of CPU, and/or only programs starting after a specific date. The displays may be done separately for each AIPS version, or the versions may be lumped together into a single display. PRTAC has a separate operation code which will clear the file for AIPS Managers only.

C.2.2 Batch text (BA) file

Overview

Batch text files contain a list of 80-character lines to be used as input to the batch versions of AIPS. There are logically two types of batch text files: the work files used by AIPS and BATER to prepare jobs for submission and the files actually queued to AIPSBn.

Names: The former have names BAr00n0m where r is the AIPS format revision code, n is the queue number (1 <= n <= NBATQS) and m = 1 through NINTRN for the interactive AIPSs and m = NINTRN + 1 for BATER. These are permanent files which can grow as needed. The latter have names BAr0nn0m where nn is the lower two digits of the job number (01 <= nn <= 64) and m is the value of NPOPS used by the AIPSBm program (i.e. m = BATQUE + NINTRN + 1). These files are created by the checker version of AIPS and are destroyed by AIPSBm when the batch job terminates.

Data structures

The physical records in these files are each 256 words. The first four words contain special information (which is only used in record 1) and the rest of the record contains logical records. There are 11 logical records in each physical record. The two kinds of files use identical data and pointer structures.

The first four words of the first physical record contain:

```
FIELD    TYPE       DESCRIPTION

 1        I    User number given in the logon
 2        I    Next available logical record number
 3        I    Logical record number of last line in file
 4        I     Number of 256-word records now in file
```

The first four words of all other physical records are unused. Each logical record has the structure:

```
FIELD    TYPE       DESCRIPTION

  1      I     Logical record number of next line (0 if none)
  2      I     Logical record number of previous line (0 if none)
  3      H(20) Text of line in HOLLERITH form.
```

Usage Notes

The first line in the file is regarded as line 0 and is always blank and always located at logical record number 1. Its sole purpose is to point to the first real line of text. If this convention were not adopted, then it would not be possible to insert text in front of the existing text in a work file nor to delete the first line in the work file. The linked list structure of these files is not necessary for the files processed by AIPSBn. However, it is maintained in order to retain a single, somewhat simpler reading program. In particular, it is used to make BATLIST and JOBLIST essentially identical.

Routines and commons for reading batch text files

The subroutine PREAD performs all standard reads for POPS language processors. If the variable IUNIT in common /IO/ is set to 3, then the read is done from a batch text file. The operation is controlled by the common /BATCH/ as :

```
BATLUN   I       logical unit number of batch text file
BATIND   I       pointer to FTAB for the open file
BATREC   I       logical record number of the last line read
                   (<= 0 if none)
BATDAT   I(256)  contents of last physical record read
```

The program which uses PREAD for reading batch files must open the file setting appropriate values in BATLUN, BATIND, and BATREC. If BATREC is set > 0, PREAD will assume that the contents of BATDAT are valid and contain the logical record referenced by BATREC. PREAD will use the pointer in the current BATREC to update BATREC and to obtain the next line, performing a read operation only when required. If the pointer in the current logical record (BATREC) indicates that there are no more lines, then, effectively, an error (end of file) condition has arisen. PREAD signals this by returning a text line of 'EXIT' or, for UNQUE in AIPS and BATER, 'ENDBATCH'.

The application subroutines AUB and CUB also read batch text files. The verb UNQUE uses the common /BATCH/ together with PREAD to read the text file being unqueued. The verbs BATEDIT, BATLIST, and JOBLIST use the common /BWTCH/ (see below) to read a file to locate the line to be editted and to locate the lines to be listed.

Routines and commons for writing batch text files

The subroutine PREAD can perform a write to a batch text file of the line which has just been read. It does this under control of the common /BWTCH/ as

```
BWTLUN   I        logical unit number of output text file
BWTIND   I        pointer to FTAB for the opened file
BWTREC   I        logical record number of the last line
                  added to the file (0 if none)
WASERR   L        .FALSE. => do the writing
BWTDAT   I(256)   contents of physical record containing
                  logical record BWTREC
```

Programs using PREAD and not wishing to write a batch text file must set WASERR = .TRUE.. As for reading, the text file must be opened and the common initialized by the program which uses PREAD. The program must also write the last record in BWTDAT to, and close, the file. It should also update the global pointers in record 1.

At present, the only program which uses PREAD for writing batch text files is AIPSCm. Because of certain problems with deletion and insertion as well as the need to check for the "magic" character strings 'RUN' and 'ENDBATCH', the batch preparation routines use the subroutine BBUILD to write into batch work files. BBUILD uses the common /BWTCH/ (except for WASERR) in a way which is similar to, but slightly messier than the way used by PREAD.

C.2.3 Batch queing (BQ) file

Overview

The batch queueing file is used to queue jobs to the batch versions of AIPS.

Name: The file is named BQr00000 where r is the system format revision code. It is a permanent file in area DA00 and must be large enough to hold NBATQS 256-word records.

Data structure

There is one 256-word record in the file for each batch processor: record n corresponding to BATQUE = n processed by AIPSB(m) (i.e., with NPOPS = m = n + NINTRN + 1).

Each 256-word block contains structures describing 64 jobs. These structures have the form:

```
FIELD    TYPE      DESCRIPTION

  1      I     User number given in the logon
  2      I     Submission time: 256 * 256 * (year-1900)
                                + 256 * month + day
  3      I     Submission time: 256 * 256 * hours
                                + 256 * minutes + seconds
  4      I     Desired version: 1-3 => OLD, NEW, TST resp.
```

In addition, the signs and contents of the first two words are used to convey the status of the job as:

```
word 1  = 0                    no job (vacant slot)
word 1  > 0  word 2  > 0       job waiting to run
```

```
word 1  > 0  word 2  < 0      job ran and failed
word 1  < 0  word 2  = 0      job is being submitted
word 1  < 0  word 2  > 0      job is running
word 1  < 0  word 2  < 0      job finished - may be reassigned
```

Usage Notes

Job numbers are formed as 100 * n + m , where m is the entry in queue n. It is simply the m'th 4-word block in record n. The queueing algorithm is first-in-first-out with a bias toward new users. Thus, the algorithm first looks for the oldest job submitted under any user number other than the previous three users of that queue. If it finds no jobs to run, it repeats the search excluding only the previous two user numbers. And so forth.

Routines and commons

The principal program to access this file is the subroutine BATQ which performs all normal operations on the file. It can "OPEN" a job by finding an available job number and setting the values in the slot to show that the job is being submitted. It takes the lowest numbered vacant slot or, if there is none, the lowest numbered job finished slot. BATQ can "RUNN" (queue) a job by converting the user number to a positive value and inserting the current time. It can "FIND" the next job to run using the algorithm mentioned above and setting the entry to job running. It can also "CLOS" a job by clearing the entry or, if it was running, by marking it finished. BATQ uses no commons and requires the calling program to maintain the list of previous users. BATQ does update the list on FIND, however. Other operations include "WHOO" to find the ID of the current job and "FAIL" to mark all currently running jobs as having failed.

The other subroutine which accesses this file directly is AUB (and the BATER version of AUB called CUB). These routines perform on the BQ file the user-oriented functions:

- QUEUES list job numbers and times in the queue
- JOBLIST list contents of submitted job
- UNQUE remove job from queue and transfer text of job back to a batch work file

C.2.4 GRIPE (GR) files

Overview

The gripe file contains gripes about the AIPS system; either bug reports or suggestions for improvements. versions of AIPS.

Name: The file is named GRr00000 where r is the system format revision code. It is a permanent file in area DA00.

Data structure

The first record of the GR file is a header record:

FIELD	TYPE	DESCRIPTION
1	I	Current file size in blocks
3	I	Current record number
4	I	Current character position

The following records contain a series of arbitrary length hollerith entries. An entry is delimiter by the characters "{" and "}". Each gripe consists of 10 entries; these are:

1. Date and time in form {dd-mmm-yyyy hh:mm:ss}

2. system name {AIPS system name}
3. User number and release date {nnn 15JUL90}
4. User name {User Name}
5. Address { User address}
6. Phone number { User phone number}
7. Text of gripe { I can't get anything to work...}
8. unused entry { }
9. unused entry { }
10. unused entry { }

Usage Notes

Gripes are entered by users and may be accessed by users. Gripes are periodically copied to the AIPS gripes managment system.

Routines and commons

The AIPS GR file access is in routine AUC which create entries and give users access to the entered gripes. Gripes are removed from the GR file on a periodic basis and entered into the AIPS gripes managment system. This is currently done with the standalone utility routine GR2TEX. Routine AIPSUB:CHGRIP reads and writes entries in the GR file. CHGRIP accepts and returns character entries without the enclosing "{" and "}". AIPS verbs which allow user access to the GR file are:

- GRIPE enter a gripe.
- GRINDEX Index the gripes currently in the GR file.
- GRLIST Displays a sepected gripe.
- GRDROP Delete a specified gripe.

C.2.5 Help (HE) file

Overview

Help files are text files used to describe things to the user at run time.

Details: Help files are text files stored in a logical area called HLPFIL. A portion of the text in each help file is displayed on the user's terminal upon request.

Names: Each Help file is named with the name of the verb, task, adverb, pseudoverb, or procedure which it describes.

File Structure and contents

The Help files are text files prepared by the local text editor. Each Help file is a separate physical file in directory with logical name HLPFIL with the first part of the name being the name of the adverb, verb, task etc to which the file applies. The file name ends with ".HLP". E.g. the HELP file for help is HLPFIL:HELP.HLP in VMS notation or $HLPFIL/HELP.HLP in Unix notation.

Since help files are used only to provide run-time assistance to users, they may contain anything the programmer feels is relevant. Three general guidelines should be followed in order to obtain pleasing displays:

1. Text should occupy only card columns 1 - 64.

2. Text should be typed in lower case.

3. The file should not contain too many lines. 23 lines will fit on almost every terminal.

Help files are typed following a general pattern. To illustrate this pattern, consider the example:

```
; PRTMSG
;-------------------------------------------------------------
;! prints selected contents of the user's message file
;# Verb General
;  This software is the subject of a User agreement and is
;  confidential in nature. It shall not be sold or otherwise
;  made available or disclosed to third parties.
;-------------------------------------------------------------
PRTMSG    LLLLLLLLLLLLUUUUUUUUUUUU CCCCCCCCCCCCCCCCCCCCCCCCCCCC
PRTMSG:  Verb to print the message log file
PRIORITY           0.0        10.0 Print messages at or above
                                   this priority.
PRNUMBER          -1.0        15.0 AIPS number to be printed
                                      0 -> current, -1 -> all
PRTASK                             Restrict to task name(s)
PRTIME             0.0       999.9 Only messages younger than
                                   PRTIME (in days) printed
DOCRT             -1.0       132.0 > 0 => output on terminal
                                   > 72 => width of terminal
OUTPRINT
                                   Printer disk file to save
-------------------------------------------------------------
PRTMSG
Type: verb
Use:  Prints contents of message file including log of input
      commands and messages produced by AIPS and by tasks shed
      by AIPS.  This verb no longer deletes the messages.
      Please use CLRMSG to keep your message files small.
Adverbs:
  PRIORITY....Limit print to message having priority >= PRIORITY
              Message level 0 (user inputs) is also printed when
              PRIORITY <= 5.
  PRNUMBER....Selects the AIPS number of the messages to be
              printed.  0 => your current one, -1 => all.  See
              HELP PRNUMBER for some details.
  PRTASK......Selects the programs whose messages are to be
              printed.  Any program name whose first characters
              match the first non-blank characters of PRTASK
              will be selected.  Thus, PRTASK = 'UV' will cause
              the messages of UVMAP, UVSUB, UVCOP, etc. to be
              printed.  ' ' => all tasks by this rule.
              NOTE: messages from GO, INPUTS, and all other
              verbs are messages from AIPS not from any other
              associated task.
  PRTIME......Only messages younger than PRTIME days will be
              printed.   <= 0 => all times printed.
  DOCRT.......<= 0 -> print the messages on the line printer
              with times, priorities, etc. listed.  > 0 -> show
              the messages in slightly abbreviated form on Bthe
```

```
            terminal.  If you have a wide terminal, set DOCRT
            to its width in characters for a more complete
            display.
 OUTPRINT....Disk file name in which to save to line printer
            output (after printing).   ' ' => use scratch.
------------------------------------------------------------
```

There are four parts to the help file.

1. Precursor lines to give the file name, a one line description, list of categories, and the user agreement statment.
2. The INPUTS section used by AIPS verbs INPUTS, GO, TPUT, etc. This has a fairly strict format. Adverb names begin in column 1, lower limits to numeric adverbs are free format in columns 11-22, upper limits to numeric adverbs are free format in columns 23-34, and a description of the adverb uses columns 36-64.
3. Separated from the INPUTS section by a row of 64 minus signs is the HELP section. This is displayed on the user terminal with verb HELP. The standard form is illustrated above.
4. Separated from the HELP section by another row of 64 minus signs is the EXPLAIN section. This provides more detailed user information, usually on the printer, with AIPS verb EXPLAIN.

The HELP file ends with a row of 64 minus signs. Some adverbs are used both for starting a task (GO) and for TELLing it things later. This is indicated by placing an astrisk ("*") in column 10 of the appropriate line of the inputs section. If an adverb is not used by GO but is used by TELL place a "?" in column 10. A line in the help file beginning with a semi-colon is a comment line, never displayed to the user.

Usage Notes

Help files are an important part of AIPS. They are the means by which users can find out how to use AIPS without leaving the terminal or leafing through thick manuals. Every task, verb, adverb, and pseudoverb must have a help file. Additional help files may also be created to explain special details. The existence of such files should be mentioned in the appropriate normal help file.

Routines and commons for reading Help files

Help files are opened and the member is located by the subroutine ZTOPEN. They are read sequentially by ZTREAD and closed by ZTCLOS. By convention, the logical unit number used for this is 11. These text file I/O routines are described elsewhere in this volume.

The subroutine which displays these files is called HELPS. The routine looks for special names which cause the symbol table to be examined and printed. Any other name is assumed to be a help file name and the file is opened (ZTOPEN), read (ZTREAD), printed (MSGWRT), and closed (ZTCLOS).

Routines and commons for writing Help files

Help files are prepared by programmers using their local source editors. No routines are to write such files at run time.

C.2.6 Image catalog (IC) file

Overview

The image catalog contains data for images stored on the TV device that identify the images, refer them back to their original map files, and specify scaling of the X-Y and intensity coordinates. There is a separate image catalog which performs the same functions for graphics devices (e.g. TEK4012 storage screens).

Name and Location: There is one image catalog file for each television device whose physical name corresponds to ICr0000n, where r = the system format revision number n = the device number (0 for graphics, 1 - n for TVs). They reside in area DA00 and must be created at AIPS installation.

Data Structures

General: For each grey-scale image plane of the TV device, the IC file contains N 1-block (256-word) records for cataloging up to N subimages, plus a (N-1)/51+1 block directory. The directory immediately precedes the catalog blocks for each image plane. For each TV graphics overlay plane there is one, undirectoried, catalog block. These blocks follow immediately after the last grey-scale block.

The IC for pure graphics devices (called TK devices) has one image catalog block for each device in the system including all "local" TK devices followed by all remote-entry devices. Record number n in this file is associated with TK device number n (NTKDEV in /DCHCOM/).

The image catalog blocks themselves are essentially duplicates of the map catalog blocks except that scaling information replaces the extension file index of the map catalog.

Record Formats:

Directory Block (Grey-scale image)

OFFSET	LENGTH	TYPE	DESCRIPTION
0	2	I	Sequence number of last sub-image cataloged on this plane
2	2	I	Seq. no. of sub-image in slot 1; 0 if slot empty
4	8	I(4)	TV pixel positions of corners of 1st sub-image, x1,y1,x2,y2
12	2	I	Seq. no. of sub-image in slot 2; 0 if empty
14	8	I(4)	TV pixel positions of corners of 2nd sub-image
.	.		
.	.		
.	.		

Catalog Block for each image or subimage:

Most of the Image Catalog block is identical to the map catalog block of the source of the image. (See section on CB files.) Pointers for the Image catalog are found in the DHDR.INC common. The information on antenna pointing, alternate frequency/velocity axis descriptions, and extension files is replaced in the IC by:

OFFSET	LENGTH	TYPE	POINTER	DESCRIPTION
472	8	R(2)	IRRAN	Map values displayed as min & max brightness (units are those of file, not the physical ones)
508	4	I	IIVOL	Disk volume from which map came
512	4	I	IICNO	Catalog slot number of orig. map
516	16	I(4)	IIWIN	Map pixel positions of corners of displayed image (rel. to orig. map)
532	20	I(5)	IIDEP	Depth of displayed image in 7 - dimensional map (axes 3 - 7)
552	16	I(4)	IICOR	TV pixel positions of corners of image on screen
568	4	I	IITRA	2-char code for transfer function used to compute TV brightness from map intensity values.
572	4	I	IIPLT	Code for type of plot.
576	100	I(25)	IIOTH	Misc. plot type dependent info. (at the moment no more than 20 used)

Usage Notes

We assume that single images only are stored on graphics planes; they are not directoried.

When a grey-image plane is cleared, its directory is zeroed. As images are added to the plane, their coordinates are written into an open directory slot for that plane, along with the current value of the plane sequence number. The sequence number is then incremented. If an old image is completely overwritten by a new one, its directory slot is cleared. For partially overlapping images, the sequence number allows the user to select the one most recently loaded into a given part of the plane.

Subroutines:

- YCINIT clears the Image Catalog for a given plane
- YCOVER are there any overlapped images in each on quadrant?
- YCWRIT adds a new block to the catalog
- YCREAD returns the block corresponding to a given TV pixel
- YFIND determines desired image, asks user if > 1 visible

These routines expect the "plane number" as an argument. TV gray scale planes are numbered 1 - NGRAY, TV graphics overlay planes are numbered NGRAY+1 - NGRAY+NGRAPH, and TK devices are referenced by any plane number > NGRAY+NGRAPH.

Commons:

The COMMON /TVCHAR/ referenced by 'DTVC.INC' contains TV device characteristics such as:

```
NGRAY  =  # of grey-scale planes on this device
NGRAPH =  # of graphics planes
MAXXTV(2) Maximum number of pixels in x,y directions in image
```

The common /DCHCOM/ (DDCH.INC) contains two important parameters in this regard: NTVDEV and NTKDEV. The subroutine ZDCHIN sets these to the actual number of such devices present locally. Then, the routines ZWHOMI (in AIPS only) and GTPARM (in all tasks) reset them to the device number assigned to the current user. ZWHOMI determines these assignments.

C.2.7 TV lock (ID) file

Overview

This file is to establish current ownership of a TV display device. To becode the current "owner" of the device, a process opens the ID file with exclusive use. This prevents other users from attempting simultaneous access to the display.

Name and Location: There is one ID file for each television device whose physical name corresponds to IDr0000n, where r is the system format revision number, and n = the device number (0 for graphics, 1 - n for TVs). They reside in area DA00 and must be created at AIPS installation.

Data Structures

There is no meaningful data in the ID file.

Usage Notes

The ID file is opened by YTVOPN using ZOPEN and closed by YTVCLS using ZCLOSE to extablish exclusive use of the display device.

C.2.8 POPS memory (ME) file

Overview

POPS memory files are used to store copies of the POPS "environment" (procedure source code, symbol table, procedure executable code, adverbs values, various pointers).

Details: The files are permanent files and have a size which depends on the size of the LISTF and K arrays in the version of POPS being used. The formula is 5 times the size of LISTF plus 4 times the size of K.

Names: POPS memory files are named MEr0000n , where r is the system format revision number and n is NPOPS (the POPS identification number).

File Structure

The structure of the K and LISTF arrays is discussed elsewhere. Here we will show only how they are stored on disk. The files consist of sequential 256-word records. Using the sizes of the K and LISTF arrays in the 15APR90 version of AIPS, the files are laid out as

```
FIELD   TYPE          DESCRIPTION

  1     I(4096)    LISTF in use by the current user
  2     I(18944)   K array for initial values and RESTART
  3     I(4096)    LISTF array for initial values and RESTART
  4     I(18944)   K array for STORE 1 and RESTORE 1
  5     I(4096)    LISTF array for STORE 1 and RESTORE 1
  6     I(18944)   K array for STORE 2 and RESTORE 2
  7     I(4096)    LISTF array for STORE 2 and RESTORE 2
  8     I(18944)   K array for STORE 3 and RESTORE 3
  9     I(4096)    LISTF array for STORE 3 and RESTORE 3
```

User notes

It should not be necessary for users to read or write this file under normal circumstances. Standard POPS subroutines do all the operations which are required.

Routines which read and write POPS memory files

The memory files are given their initial values by the program POPSGN. This program does a special compilation followed by an AIPS-like compilation on text files called POPSDAT for interactive and batch AIPSs. POPSGN sets all areas of the memory file to the "virgin" K and LISTF arrays which it has computed.

In the POPS programs, the subroutine INIT moves field 3 ("virgin" LISTF) into field 1 (working LISTF) and moves field 2 into the core copy of the K array. The subroutine STORES updates the working copy of LISTF (field 1 on disk) and performs the pseudoverbs LIST, STORE, RESTORE, SAVE, GET, CORE, and SCRATCH. The variables in the common /POPS/ which aid this process are

```
LPAGE       number of 256-word blocks in LISTF (= 16)
MPAGE       number of 256-word blocks in K plus LISTF (= 90)
```

C.2.9 Message (MS) file

Overview

Message log files are used to record the input to, and messages from, all programs.

Details: Message files are user-owned files which record all messages except those with no time-variable information content (e.g. HELPs, interactive instructions) and those explicitly directed to the line printer

(e.g. verbs PRTMSG and PRTHI, tasks PRTIM and PRTPL). All programs at all POPS numbers use the message file of the logon user.

Names: The message file is called MSfuuu00.uuu, where uuu is the user number in hexadecimal notation and f is the data format version code letter.

File Structure

Message files are sequential files consisting of 256-word records. Each of these physical records begins with two words of special information (which are used only in record 1). The remainder of each physical record is devoted to 10 logical records.

The special information in word 1 of the first record is simply the current number of messages recorded in the file and the number of physical records currently in the file. Each logical record has the structure

FIELD	TYPE	DESCRIPTION
1	I	message priority (0 - 10) + 16 * POPS number of task
2	I	message date: YY/MM/DD packed
3	I	message time: HH/MM/SS packed
4	H(2)	name of task generating message (5 HOLLERITH characters)
5	H(20)	80-character message (HOLLERITH characters)

User notes

All programs should use this file for message display. They do this with the subroutine MSGWRT described below.

Routines and commons for writing message files

The subroutine MSGWRT is the only routine which should be used to write on these files. It is a somewhat complicated routine and very fundamental to the AIPS system. It uses the commons in include DMSG.INC as :

MSGCNT	I	number of messages in file (< 0 implies don't know and file is closed)
MSGREC	I	number of 256-word records in file at present
TSKNAM	C*6	name of current task
MPOPS	I	POPS identification number
MLUSER	I	user number from logon
MSGSUP	I	if = 32000, suppress level 6 & 7 messages
ISBTCH	I	if = 32000, treat this job as a batch job
MACOUN	I	position in the accounting file this job
MSGTXT	C*80	80-character message to be logged in file and/or displayed on terminal

MSGWRT opens the message file and initializes MSGCNT, puts the message into the file (unless inhibited), displays the message on the terminal (LUN=6, unless inhibited), and closes the file updating the recorded message count and resetting MSGCNT. MSGWRT will expand the file with no limit as needed. It will complain to interactive users when the number of messages is greater than 750.

Routines and commons for reading message files

The subroutine PRTMSG is used to print, and compress message files. It takes 7 input and 3 output arguments. The first is an opcode of 'PRIN' for print, 'DELE' for compress, or anything else for a typed

summary. The next 6 arguments define the user number, the POPS number (0 => all), the priority lower limit for printing, the task name, a time cutoff for printing or deleting, and a logical to request the printing to be on the CRT terminal. The outputs are the number of messages printed/deleted, the number left in the file, and an error code. See the precursor remarks for the details. The physical file is compressed after messages are deleted.

C.2.10 Password (PW) file

Overview

The PW file contains AIPS user passwords. These are AIPS internal passwords and are not related to system passwords.

Details: AIPS user passwords are up to 12 characters kept as 3 hollerith variables. The first of these words in the (256 integer) record given by:

```
RECNO = (user_number - 1) / (256/3)) + 1
```

in word number:

```
WORD = MOD ((user_number - 1), (256/3)) * 3 + 1
```

Names: The password file is called PWf00000., where f is the system format version code letter.

File Structure

Password files are sequential files consisting of 256-word records. User passwords are stored in blocks of 3 hollerith variables (all in a given record) as described above.

Useage notes

All access to the PW file is through ZFIO. AIPS usage of the password file is limited to routine AUC. Several standalone programs also use the password file.

C.2.11 POPS Save-Get (SG) file

Overview

POPS memory files are used to store copies of the POPS "environment" (procedure source code, symbol table, procedure executable code, adverbs values, various pointers). Save/get files are "user-owned", rather than the "public" ones described in the ME portion of this appendix.

Details: The files are semi-permanent files and have a size which depends on the size of the LISTF and K arrays in the version of POPS being used. The file must be larger than one K plus one LISTF array. The files are at least 91 256-word blocks long. Each user having one or more SG files also has an SG directory file. On ordinary computers this file only requires 8 blocks.

Names: POPS save/get files are named SGfuuujj.uuu , where uuu is the user logon identification number in hex and jj is the sequence number in hex of the file in the SG directory. Sequence number 0 is reserved for the directory file. The data format revision letter (A, B, ...) is f.

File Structure

The SG directory file consists of 256 logical records, numbered 0 through 255. Each logical record consists of 7 words. There are 256 / 7 logical records per physical record (36). The first logical record (called no. 0) is reserved for control information and currently includes:

FIELD	TYPE	DESCRIPTION
1	I	Maximum version number now in use (called max)

```
2     I          Number of vacant entries with version # < max
3     I(2)       Last GET time (packed YY/MM/DD, HH/MM/SS)
4     I(2)       Last SAVE time (packed YY/MM/DD, HH/MM/SS)
5     I          Reserved
```

Logical records 1 to 255 (correspond to file version numbers 1 - 255) contain

```
FIELD   TYPE          DESCRIPTION

  1     I           Number characters in user-supplied name ( <= 0
                    means empty) + 32 * SG format version number
  2     I(2)        Last SAVE time (packed YY/MM/DD, HH/MM/SS)
  3     H(4)        User supplied name, blank filled, 16 char.
```

The SG data files are described below:

The structure of the K and LISTF arrays is discussed elsewhere. Here we will show only how they are stored on disk. The files consist of sequential 256-word records. Using the sizes of the K and LISTF arrays in the Charlottesville version of AIPS, the files are laid out as

```
FIELD   TYPE          DESCRIPTION

  1     I(256)      Header record (see below)
  2     I(18944)    K array
  3     I(4096)     LISTF array
```

The header record at the present contains only 6 useful words:

```
FIELD    TYPE          DESCRIPTION

  1      I(3)      Date: year since 0, month, day
  2      I(3)      Time: hour, min, sec of the last SAVE op
```

User notes

It should not be necessary for users to read or write these files under normal circumstances. Standard POPS subroutines do all the operations which are required. They are created at run time by the user as required. They may be deleted by the user at run time or by the AIPS manager using FILINI or other utilities.

Routines which read and write POPS memory files

The subroutine STORES creates, writes, and reads the save/get files. The subroutine AU3A reports on disk usage including such files and can destroy all such files belonging to the logon user. The variables in the common /POPS/ which aid in this process are

```
LPAGE       number of 256-word blocks in LISTF (=16)
MPAGE       number of 256-word blocks in K plus LISTF (=90)
```

The subroutine SGLOCA, called by STORES, performs normal operations on SG directory files including creation. The verb SGDESTR performed by the subroutine AU2A handles destruction of individual SG data files and will destroy an empty directory. AU2A also performs the directory listing function (verb SGINDEX). Subroutine AU3A will destroy all the user's SG files at once (verb SAVDEST).

C.2.12 System parameter (SP) file

Overview

The system parameter file provides programs with information about the system resources that are available to AIPS.

Details: All AIPS programs read this file as one of the first steps of program execution. This is a permanent file created at AIPS installation time either with FILINI or FILAIP or as part of an automatic installation procedure that is available on a limited number of computers. The size needed is one 256 word block, rounded up to one granule.

Name: The file name is SPr00000. where r is the system format revision code letter.

File Structure

The file consists of one 256 word logical record.

```
FIELD   TYPE   DESCRIPTION

  1     I      Number of disk drives available.
  2     I      Number of tape drives available.
  3     I      Number of lines per CRT page.
  4     I      Number of lines per print page.
  5     I      Number of batch queues.
  6     I      Plotter no. of X dots per page.
  7     I      Plotter no. of Y dots per page.
  8     I      Plotter no. of X dots per character.
  9     I      Plotter no. of Y dots per character.
 10     I      Maximum no. of interactive users.
 11     I      Number of words in AP (in 1024 sections).
 12     I      Number of TV devices available.
 13     I      Number of graphics devices (like TEK 4012) available.
14-63   I(50)  Device table (see below).
 64     I      No. of users allowed access to TVs.
 65     I      No. of users allowed access to graphics devices.
 66     I      No. of entries in private catalogs. 0=> public catalog.
 67     I      Max. user number.
 68     I      Width of line printer in characters.
 69     R      # 1024s words of secondary AP memory.
 70     I      Shortest vector length to vectorize
 71     R      No. or dots per mm on printer
 72     R      No. of dots per mm on graphics screen.
 73     H(5)   System name or ID (4 characters/floating point).
 78     R(15)  Min. TIMDEST time for each disk (days)
 93     R      Min. TIMDEST time for SAVE/GET files (days)
 94     R      Min. automatic destruction time for messages
 95     R      Min. automatic destruction time for scratch
 96     R      Min. destruction time for empty catalogs.
 97     R(4)   Times during which AP Batch jobs cannot start.
               1, 2 start, stop times (hrs) on weekends
               3, 4 start, stop times (hrs) on weekdays
101     R(3)   1 => time between rolls (min)
               2,3 polynomial terms for determining how long
               a job must wait before grabbing the AP.
104     R(120) Lists of allowed users, 8 per disk for up to 15
               disks. 0=all, -1=scratch only otherwise user numbers.
               Ordered by disk.
```

```
224     R(2)  Graphics screen size x,y
226     R(2)  Graphics character size x,y

 REST OF BLOCK  Free
```

The device table is used to describe the characteristics of a file that can be opened on a specific AIPS I/O logical unit number. The logical unit numbers run from 1 to 50 and are described by its corresponding device table value. The codes for the device table values are listed below.

```
DEVICE TABLE        TYPE OF FILE OPENED ON THE CORRESPONDING LUN
   VALUE
     0              File manager file with FTAB
     1              Fortran device (no FTAB)
     2              Non-file manager with FTAB
     3              Other, no FTAB
     4              television device with FTAB
```

If this isn't clear, don't be alarmed. The device table need not be changed unless significant alterations or additions to AIPS are made.

Routines and commons for reading the system parameter file.

The subroutine ZDCHIN (called by every program in the initial stages of execution) will open, read and then close the system parameter file. The information is then transfered by ZDCHIN to commons DCHCOM and FTABCM which are found in include file DDCH.INC.

Routine for initializing and updating the system parameter file.

Stand alone program SETPAR allows the AIPS manager to initialize the system parameter file during AIPS installation or to modify the file when conditions warrant (for example when another disk is made available). SETPAR is an interactive program with prompts. The first prompt that appears when SETPAR is run is:

```
Enter:  1=Init, 2=Change vals, 3=Change DEVTAB, 4=Quit
```

Option 1 (INIT). This option is normally chosen during AIPS installation. SETPAR will initialize the values in the SP file with the default values but SETPAR will not create an SP file. This must be done with program FILINI or FILAIP. After initializing the values SETPAR will immediately proceed to option 2 described below.

Option 2 (CHANGE VALS) This option allows the AIPS manager to change the current SP file values. When this option is chosen the current values and a description is printed on the CRT. Then the following prompt is printed.

```
Enter number to change or  0 = print,  -1 = Return
```

The "number to change" in the prompt refers to the number on the far left of the description of the value. For example if the user wants to change the number of disks available to AIPS from 2 to 3, then the user finds the following line on the CRT:

```
1  no. of AIPS data disks            2
```

In this case the 1 is the "number to change" and 2 is the current number of disks. Thus, the user enters 1 in free format. The next prompt is:

```
1  no. of AIPS data disks            2
```

The user then enters a 3 in free format. The program then reprints the prompt:

```
Enter number to change or  0 = print,  -1 = Return
```

At this point, the user can change another value by entering the proper positive number, or reprint the latest values by entering a 0, or return to the very first prompt (from which the quit command can be issued) by entering a -1. If values are changed SETPAR will ask for the password before changing the file.

Option 3 (CHG DEVTAB) This option allows the user to change the device table values described above. The details of operation are very similiar to option 2.

Option 4 (QUIT) This option will save the latest SP file values and stop the program.

C.2.13 Task Show and Tell (TC) file

Overview

The Show and Tell file is used to pass parameters to tasks which are already running and which can make some use of parameters during execution.

Details: The use of this file is for interactive, but asynchronous, control of tasks. Thus, the only program to write new parameters into the file is AIPS. Those tasks which can change parameters during execution (e.g. number of iterations), or which can sensibly be told to quit early may read the file at any time. If information intended for them is present, then the task should fetch that data and remove the indication of the communication to it.

Name: The file is named TCr00000 where r is the system format code. It is a permanent file and must contain at least (4 + 8 * NINTRN) 256-word records. It is created during system installation.

File Structure

The file consists of 4 directory records followed by up to 120 data records. The directory records contain 64-word logical records one for each value of NPOPS, 4 per physical record. Each 64-word record contain 8 8-word logical records, allowing up to 8 TELL operations to be queued for each POPS number at any one time. The structure of these 8-word records is

WORD	TYPE	DESCRIPTION
1	H(2)	Task name (Hollerith)
3	I(2)	Time (YY/MM/DD, HH/MM/SS)
5	I	User number
6	I	Operation
7	I(2)	Reserved

The data records follow, one for each operation, 8 for each POPS number. They contain adverb values specified in the Inputs section of the HELP file by am "*" or "?" in column 10, preceded by the value of the adverb OPTELL.

C.2.14 Task data (TD) file

Overview

The task data file is used to pass data to spawned tasks and to return a "return code" to the initiating tasks.

Details: The only programs in the AIPS system which are allowed to spawn tasks are AIPS, BATER, AIPSCn, and AIPSBm. These tasks use the task data file to pass binary parameter data to spawned tasks. This file and the TC file are the only mechanism for direct intertask communication. Other forms of intertask communication are accomplished indirectly via the changes produced in the various catalog, map, history, plot, *et. al.* files by the normal functioning of verbs and tasks. SHOW and TELL use file type TC to pass parameters to running tasks in a manner similar to that used for the TD file.

Name: The file is named TDr00000 where r is the system format version number. It is a permanent file and must contain at least 46 256-word records. It must be created during system installation.

File Structure

The file consists of 46 256-word physical records. They are ordered logically in the file as

FIELD	TYPE	DESCRIPTION
1	I(256)	Control record
2	I(768)	Data area: NPOPS = 1
3	I(768)	Data area: NPOPS = 2
4	I(768)	Data area: NPOPS = 3
5	I(768)	Data area: NPOPS = 4
6	I(768)	Data area: NPOPS = 5
7	I(768)	Data area: NPOPS = 6
8	I(768)	Data area: NPOPS = 7
9	I(768)	Data area: NPOPS = 8
10	I(768)	Data area: NPOPS = 9
11	I(768)	Data area: NPOPS = A
12	I(768)	Data area: NPOPS = B
13	I(768)	Data area: NPOPS = C
14	I(768)	Data area: NPOPS = D
15	I(768)	Data area: NPOPS = E
16	I(768)	Data area: NPOPS = F

The control record is divided into fifteen five-word entries and the remaining 181 words are ignored. Each entry has the structure:

WORD	TYPE	DESCRIPTION
1	H(2)	Task name: (Hollerith)
3	I	Return code
4	I(2)	Reserved

The data areas are simply 768 words of binary data. Their structures are almost totally determined by the task being activated. The only standards are that all task-dependent values are in floating point and that the first N integer values convey

1	I	The logon user number
2	I	The assigned TV device number
3	I	The assigned TK device number
4	I	The MSGKILL parameter
5	I	The ISBATCH parameter
6	I	The DBGAIP parameter (controls print levels, debugger use...)
7-8	I	reserved
9	R	DOWAIT value
10	H	Version string (4 characters)

Words 11 through 768, as needed, contain the adverb values required by the task in binary form. Note: character adverbs are sent as Hollerith strings.

Usage Notes

This file is clearly important and somewhat dangerous. It must be handled carefully and kept available to the full system as much as possible. There are standard routines for handling the file. There should not be a need for new routines and/or new uses of this file.

Routines and commons for writing the task data file

There are two standard subroutines which write on the task data file: AU2 which spawns tasks for AIPS and AIPSBn and RELPOP which resumes the initiating programs. The subroutines AUA (called CUA in BATER) which does the verb SUBMIT, CHSTOP which can start up batch queues for AIPSCn, and GTPARM, which is described in the next section, also write on this file.

Except for the batch submission and activation process, task activation is done by the following process. AU2 writes the standard values (listed above) into the first 8 integers and 2 reals of the data area corresponding to the current value of NPOPS (physical records 3*NPOPS - 1 through 3*NPOPS + 1). The remainder is filled by AU2 with the adverb values for the task. Then AU2 places the root task name in record one in the entry corresponding to the current value of NPOPS (i.e. words 5*NPOPS - 4 through 5*NPOPS - 3) and the initial return code (-999) in the return code entry (5*NPOPS-2). Finally, AU2 releases control of the task data file, activates the task, and waits for a resumption signal from the spawned task using subroutine TASKWT. That signal is now given by changing the -999 return code to some other value (or by aborting). AU2 determines the order of the adverbs required by the task by reading adverb names from the Inputs section of the help text file associated with that task. Using the POPS language subroutines, AU2 locates the adverbs in the symbol table, determines from the symbol table the number of words of data corresponding to the adverb name and the location of those words in the K array, and moves those words to the data area. Note that this makes tasks impervious to changes in POPS and the structure of the K array.

RELPOP is the subroutine used by spawned tasks to resume AIPS, AIPSBn, and BATER. A one word "return code" is written on record 1 in the entry corresponding to the current value of NPOPS. At present, this return code has only a binary meaning. A value of zero means that the spawned task was happy, at least when it called RELPOP. Any other value implies problems and causes batch jobs to abort.

The activation of AIPSCm and AIPSBn is less complicated. AIPSC requires three adverbs: BATQUE, the desired batch queue number, DETIME, the desired delay time before starting the job, and VERSION, the desired release for the AIPSB to be used. AIPSB requires only the job number that QMNGR wants to be run. Thus, Inputs files and language routines are not needed. For reasons of speed and simplicity in dealing with the message file, the AIPSs now run their batch checkers (AIPSC) their own value of NPOPS. BATER and its checker both use NPOPS = NINTRN+1. The checker programs will activate the batch control program QMNGR if it is not already running. To do this, Checker uses the data area appropriate to the value of NPOPS under which the QMNGR is supposed to run (NINTRN+2). Other than these differences, the activation of AIPSCn and AIPSBm procedes by the same process as for other tasks. (Note: NINTRN is the maximum number of simultaneous interactive AIPSs allowed.)

Routines and commons for reading the task data file

The subroutine GTPARM is used by all spawned tasks to obtain the data passed by the initiating program. GTPARM reads the first record in the task data file and locates the entry containing the root task name. This location implies the value of NPOPS which is to be used and the location in the task data file of the binary data. GTPARM then moves the requested number of words from the disk area to a buffer provided by the calling program and initializes MSGCNT, NPOPS, NLUSER, TSKNAM, MSGCNT, and DBGAIP in the include DMSG.INC. Finally, GTPARM zeros the entry containing the task name and updates the control record (no. 1) in the task data file. This last operation is needed in order to prevent interference between AIPSm's and AIPSBm's in initiating tasks with the same root name. GTPARM also returns a parameter (RQUICK) which instructs the task to resume the initiating program as soon as possible or only after the task has done its thing. It is this parameter which forces batch processes to be synchronous, but allows asynchronous processes for interactive users. The formula is simply

```
RQUICK  =  (.NOT. DOWAIT)  .AND.  (NPOPS .LE. NINTRN)
             .AND.  (ISBTCH.NE.32000)
```

GTPARM also picks up the assigned TV and TK device numbers and the code to suppress messages (if the user asserted MSGKILL TRUE).

C.2.15 Tape lock (TP) file

Overview

The tape dummy file is used as a means to take exclusive use of a tape drive.

Details: Many systems do not allow a sub-process to take exclusive use of a tape drive. The parent process (i.e. AIPS) or others can still perform a rewind in the middle of a job which is writing on the tape! In order to avoid this, all open operations on tapes also open exclusive a disk (TP) file.

Name: The files are named TPr0010n, where r is the format revision number and n is the tape drive number. They are permanent files, but need not contain any data. They must be created during system installation.

File Structure

The file contains no data and need not even have any records.

Usage Notes

These files are accessed by ZTPOPN and ZTPCLS and should not be accessed by any other programs.

C.2.16 Task Adverb Save (TS) file

Overview

The Task Save file stores the adverb values for the most recent execution of each task for each user.

Details: When AIPS, AIPSB, and AIPSC "spawn" tasks they pass adverb values to the task via the Task Data file. The values are also saved in a directoried Task Save file. The file contains only one area per task root name, but can hold areas for a large number of tasks. The adverb values used for the last execution of a given task may be retrieved from this file using the task's Inputs file and a process similar to GO. The adverb values may also be put into the file with the verb TPUT without actually spawning the task.

Names: The file is named TSfvvvnn.uuu, where, for all interactive AIPS, uuu and vvv are the user number in hex and nn is 0. For batch AIPS, vvv is 0 and nn is NPOPS. For batch, these files are temporary and last only through the execution of the batch (or Checker) job. For interactive however, they are created when first needed and remain until explicitly destroyed. f is the data format revision code letter.

File Structure

The first six 256-word blocks in the file consist of 5-word logical records, 51 per block. The first logical record (called logical record no. 0) contains

FIELD	TYPE	DESCRIPTION
1	I	Number of 256-word blocks in file
2	I	Number of tasks currently in directory
3	I(2)	Most recent write access (packed YY/MM/DD, HH/MM/SS)
4	I	Reserved

Logical records 1 through 305 contain task information:

FIELD	TYPE	DESCRIPTION
1	H(2)	Task name (Hollerith)
2	I(2)	Last write access time (packed YY/MM/DD, HH/MM/SS)
3	I	Version code: 1-8 => OLD, NEW, TST, OLDPSAP, NEWPSAP, TSTPSAP, LOCAL, PRIVATE

Note that we allow 8-character names here. This permits a user to save the adverb values for verbs (via GO *verb-name*) if so desired. The fact that a verb was specified will be trapped and the sequence TPUT *verb-name* ; *verb-name* will be substituted. Of course, the user can simply save the verb parameters with TPUT *verb-name* .

The remainder of the file contains task data in the form transmitted to the task in the TD file (see description of the TD file in this appendix). Only three records per task are allowed. If the task name occurs in the directory logical record IN, then the task data occur in physical blocks 3 * IN + 4 and, if needed, up to 3 * IN + 6.

Usage Notes

There should be no need for routines other than those listed below to access this file.

Routines which read and write the Task Save file

The subroutine AU2 performs the operations of GO and TPUT (see description of the TD file in this chapter). As it does so, it also opens the TS file (creating one if needed) and locates the task name in the directory (creating an entry and expanding the file, if needed). When it writes a record to the TD file, it writes the same data to the appropriate record of the TS file.

The subroutine AU2A performs the verb TGINDEX which lists the task names and last write times found in the TS file directory. AU2A also performs the verb-like portions of the pseudoverb TGET. Using code nearly identical to that of AU2, it locates the relevant Inputs file and locates the task name in the TS file directory. It then parses the Inputs file identifying the adverb names and properties via POPS language processing routines and transfers the adverb values from the TS file to the in-core K array.

C.3 User data files

This section describes files that are used for user data files other than tables.

C.3.1 Catalog directory (CA) file

Overview

Catalog files contain directory information for the AIPS files stored on a disk.

Names and Locations: There is a catalog file on each disk on which user data can be stored. The catalog refers only to maps, uv data, and scratch files on its own disk volume. The catalogs have physical names corresponding to "CAf00000", where f is the current fomat code letter (A, B, ...) and can be user owned (.uuu = user number in hex) or public (.uuu omitted).

Data Structures and usage notes

File Structure: Each catalog file contains a one block (256-word) header and a number of catalog directory blocks. The header block contains principally the number of catalog blocks in the file; this is set when the file is created or expanded. The directory blocks contain a reference to each catalog entry. The directory is used to speed catalog searches and also contains the status words that register file activity. A catalog to store N entries must have enough space for 1 + CEIL[N/NLPR] blocks (i.e. catalog blocks + directory), where NLPR is defined below and is 25 on normal machines.

Record Formats:

Header Block:

OFFSET	LENGTH	TYPE	DESCRIPTION
0	1	I	Volume number of disk containing this catalog
1	1	I	Unused
2	1	I	Number of catalog blocks in this file
3	3	I	Date (YYYY, MM, DD) create

```
  6      3    I   Time (HH, MM, SS) create
  9      3    I   Date (YYYY, MM, DD) last access
 12      3    I   Time (HH, MM, SS) last access
```

Directory Block:

The Mth directory block contains NLPR entries, each NWPL words, indexing the NLPR*(M-1)+1 to the NLPR*M-1 catalog blocks. In a file with N catalog blocks, the first directory block is the 2nd block in the file. The parameters are given by NWPL = 10, NLPR=256 / NWPL.

```
OFFSET LENGTH TYPE         DESCRIPTION
  0      1    I     User ID number; or -1 if slot is empty
  1      1    I     Map file activity status
  2      2    I(2)  Date/Time file was last accessed
  4      1    I     User defined sequence number 1 to 9999
  5      5    H(5)  Image ID as:
                        User defined name, 12 characters
                        User defined class, 6 characters
                        Program defined type, 2 characters
   ...
```

Directory Usage Notes:

Directory: Map name and class are user defined character strings of 12 and 6 characters that can be used to identify and locate a specific map. The strings are stored as Hollerith characters together with the 2-character string which identifies the "physical" map type, in their slots in the directory. The sequence number is similarly an arbitrary I reference number. The Map Status is an I number registering the activity of the map file itself.

```
STATUS = 0    => no programs are accessing the map file
       = n>0  => n programs are reading the map
       = -1   => one program is writing into the file
       = n<0  => 1 + n programs are reading the map, one
                 program is writing into the file.
```

File Type: This word describes file type. At present only 'MA' => map and 'UV' => single- or multi-source uv and 'SC'=> scratch files are allowed.

Usage protocols:

Maintaining the integrity of the catalog entries is essential to insure reliable access to the map files. Thus certain rules should be followed when using the catalog. These rules are coded in to the utility routines described below; these routines should be used when at all possible to access the catalog.

Rules:

1. Take exclusive use of the catalog whenever you access it. The required operation should be done quickly and then the catalog file should be closed and released.

2. The status word must be monitered to see if an intended catalog or map operation will disturb an (asynchronous) operation already in progress.

Specifically: Do not modify a catalog block, nor write into a map file which is not in a rest state (STATUS = 0).

If you intend to write into a map and STATUS = 0, change the status to "WRITE" (STATUS = -1) before releasing exclusive use of the catalog.

If you intend to read a map file or catalog block, check to see if someone else is writing on it (STATUS < 0). If so decide whether this is acceptable to your program. If so modify the status to indicate use;

```
STATUS = 1 + STATUS if STATUS > 0
STATUS =-1 + STATUS if STATUS < 0.
```

Clear status when you have finished your operation. If you were reading, reverse the process just described. If you were writing; STATUS = - (1 + STATUS)

Subroutines

- CATDIR searches, lists, and modifies the catalog directory
- CATIO reads and writes catalog blocks and can modify status
- CATOPN opens the catalog file on a given volume
- MCREAT, MAPOPN, MAPCLS, and MDESTR handle most of the catalog bookkeeping while creating, opening, closing, or destroying map files.

C.3.2 Catalog header (CB) file

Overview

Image header files contain descriptions of the format and contents of standard image files.

Names and Locations: There is an image header file for every image stored on disk (MA, UV, SC, or whatever) and cataloged in the corresponding CA file. The header files have physical names corresponding to CBfccc01.uuu where f is the format revision code, ccc is the catalog number of the image file in the CA directory file, and uuu is the user number. Note that ccc and uuu are in hex.

Data Structures and usage notes

File Structure: Each catalog header file contains a one block (256-word) binary header and a number of keyword = value entries in as many following blocks as are required. A catalog file must be at least two blocks long.

Record Formats:

Catalog Blocks:

```
TYPE    POINTER      DESCRIPTION

H(2)    KHOBJ        Source name
H(2)    KHTEL        Telescope, i.e., 'VLA'
H(2)    KHINS        e.g., receiver or correlator
H(2)    KHOBS        Observer name
H(2)    KHDOB        Observation date in format 'DD/MM/YY'
H(2)    KHDMP        Date map created in format 'DD/MM/YY'
H(2)    KHBUN        Map units, i.e., 'JY/BEAM '
H(2)(7) KHPTP        Random Parameter types
        KIPTPN= 14   Max. number of labeled random paramaters
H(2)(7) KHCTP        Coordinate type, i.e., 'RA---SIN'
        KICTPN= 7    Max. number of axes
D(7)    KDCRV        Coordinate value at reference pixel
R(7)    KRCIC        Coordinate value increment along axis
R(7)    KRCRP        Coordinate Reference Pixel
R(7)    KRCRT        Coordinate Rotation Angles
R       KREPO        Epoch of coordinates (years)
R       KRDMX        Real value of data maximum
R       KRDMN        Real value of data minimum
R       KRBLK        Value of indeterminate pixel (real
                      maps only)
```

```
I       KIGCN       Number of random par. groups.
                    This is the number of uv data
                    records.
I       KIPCN       Number of random parameters
I       KIDIM       Number of coordinate axes
I(7)    KINAX       Number of pixels on each axis
I       KIIMS       Image sequence no.
H(3)    KHIMN       Image name (12 characters)
        KHIMNO= 1   Character offset in HOLLERITH string
H(2)    KHIMC       Image class (6 characters)
        KHIMCO= 13  Character offset in HOLLERITH string
H       KHPTY       Map physical type (i.e., 'MA','UV') (2 char)
        KHPTYO= 19  Character offset in HOLLERITH
I       KIIMU       Image user ID number
I       KINIT       # clean iterations
R       KRBMJ       Beam major axis in degrees
R       KRBMN       Beam minor axis in degrees
R       KRBPA       Beam position angle in degrees
I       KITYP       Clean map type: 1-4 => normal,
                    components, residual, points.
                    For uv data this word contains a
                    two character sort order code.
I       KIALT       Velocity reference frame: 1-3
                    => LSR, Helio, Observer +
                    256 if radio definition.
D       KDORA       Antenna pointing Right Ascension
D       KDODE       Antenna pointing Declination
D       KDRST       Rest frequency of line (Hz)
D       KDARV       Alternate ref pixel value
                    (frequency or velocity)
R       KRARP       Alternate ref pixel location
                    (frequency or velocity)
R       KRXSH       Offset in X (rotated RA) of phase center
R       KRYSH       Offset in Y (rotated Dec) from tangent pt.
H(20)   KHEXT       Names of extension file types (2 char)
        KHEXTN= 20  Max number of extension files
I(20)   KIVER       Number of versions of corresponding
                    extension file.
```

Comments

General

1. Standard names are given for the pointer variables. The values for the pointers are computed by the subroutine VHDRIN and are machine-dependent. The values are found in the common /HDRVAL/ via include DHDR.INC. The characters of each H*8 variable are packed separately (and left-justified) in as many hollerith variables as required. The image name, class, and physical type are as a 20-character string in as many Hollerith variables as required.

2. The header contains 256 words and should be contained in the arrays CATBLK(256), CATH(256), CATR(256) and CATD(128) which are INTEGER, HOLLERITH, REAL, and DOUBLE PRECISION resp. and are equivalenced. Pointers of the type KI... should refer to CATBLK locations, KH... to CATH locations, KR... to CATR locations and KD... to CATD locations. E.g. CATD(KDORA) contains the D pointing right ascension. The variable names used here are now standards and should be used wherever possible.

3. When used in the image catalog (IC file) instead of the CB file some parameters are given new meanings. See the IC file description for details.

Specific

- KHINS: Any special equipment etc. used during observations
- KHPTP: Random parameters are those associated with an irregularly gridded "array".
- KHCTP: Seven coordinates!!! Four will commonly be used; RA, DEC, FREQ and STOKES.
- KDCRV: In keeping with the FITS format convention, angles are expressed in degrees.
- KREPO: Somewhat astronomically specific. 1950.0 or 2000.0 are used.
- KRBLK: The value used to specify that a pixel is undefined; usually 'INDE'. If there are no blanked pixels, set to 0.0.
- KHEXT: The types of subsidiary files associated with the map are given by a two letter designation; eg.'HI' for history files, 'PL' plot file.
- KIVER: The current highest version number of the associated file type listed in the same relative array position in the previous type listing.

Keywords

The keyword section contains one or more records beginning in record two of the file. The first keyword record uses words one through six as

```
WORD      TYPE      USE
  1         I       Number of 256-integer records in file
  2         I       Number of keywords in file
3-6         I(4)    Reserved
```

Keyword N is found in disk record N/51 + 2 at word 5 * MOD (N,5) + 2. In other words, each keyword logical record requires 5 words, the first logical record in disk record 2 is logical record 0 used for the control data listed above, and the first word of all disk records (except the control) is ignored. A logical record contains:

```
WORD      TYPE      USE
1-2         H(2)    Keyword (8 HOLLERITH characters)
3-4          *        Keyword value (word 4 used for
                         type 3 and, if NWDPDP = 2,
                         for type 1)
 5           I        Keyword type:
                         1 double precision floating
                         2 single precision floating
                         3 character (8 HOLLERITH chars)
                         4 integer
                         5 logical
```

Subroutines

- CATCR creates CA files
- CATDIR searches and modifies the catalog directory, creates CB
- CATIO reads and writes catalog blocks and can modify status

- CATKEY reads/writes keywords in the CB file
- CATOPN opens the catalog file (CA) on a given volume
- MCREAT, MAPOPN, MAPCLS, and MDESTR handle most of the catalog bookkeeping while creating, opening, closing, or destroying map files.

C.3.3 Gain (GA) file

Overview

This extension file for a uv data set contains the gains resulting from ASCAL.

Details: GA files use the EXTINI-EXTIO file structure. With the rescaling factor in the file header. The logical record length is 256 words

Names: The file name is GArsssvv where r is the format revision code, sss=catalog number and vv = version number.

File structure.

The file header record contains the following:

```
    Location
Location  Type            Description
  255       R      GM=mean gain modulus.
```

Logical record structure.

```
   Location
  Addr  Type   Name          Description
    1    CX    GAIN(28,2)    IF gains
  113    I     IFLG(2)       Packed logical IF flags 1st 28=IF 1,
                             2nd 28=IF 2.
  115    I     BFLG(24)      Packed logical baseline flags,
                             1st 378 = IF 1, next 378 = IF 2.
                             The order number for baseline i-j
                             is j - 28 + i(55 - i)/2
  139    I     KV            Last vis number in the time range.
  140    R     T1            Start time of solution interval.
                             NOTE: times are in seconds.
  141    R     T2            End time of solution interval.
  142    I     IREF(2)       Reference antenna for the 2 IFs.
  144    I     GACAL(28,2,2) (ant, if, 1) = Tsys (used by VBANT)
                             (ant, if, 2) = Tant
```

User notes.

When calling EXTINI use LREC=256.

Routines to write GA files:

EXTINI and EXTIO, GA files are currently written by ASCAL.

Routines to access GA files:

EXTINI and EXTIO are used to access GA files.

C.3.4 History (HI) file

Overview

History files are used to record, in character form, the processing history of an image.

Details: Although they are in fact separate physical files, they are treated as if they were extensions of the file containing the image data. There must be one and only one history file for each image. The files are normally created with relatively modest size and extended when necessary.

Names: The file names are HIfsss01 , where f is the format version code and sss is the sequential position of the image header in the catalog file for that disk in hex. With user owned file systems, the names are HIfsss01.uuu, where uuu is the user number in hex.

File Structure

History files consist of sequential 256-word records. Each of these physical records begins with 4 words of special information (used only in the first record). The rest of each physical record consists of 72-character logical records in AIPS HOLLERITH form (4 characters per integer or floating). There are NL such logical records per physical record where NL = 252/(72/4) = 14.

The special data in the first record has the form

FIELD	TYPE	DESCRIPTION
1	I	Number of logical records currently in file
2	I	Number of logical records which can fit in file before expansion is required.
3	I(2)	Reserved.

The logical records consist of 72 HOLLERITH characters with no required format. However, the following FITS-like usages are recommended strongly:

1. The first 5 characters should be the left-justified name of the program causing this history record and the sixth character should be blank.
2. The rest of the card should show parameter values in the form <keyword> = <value> where more than one keyword may appear on a "card" (logical record). Character string values must be enclosed in single quotes (').
3. Information which should never be parsed (i.e. comments) should follow any parsable (i.e. keyword=value) information and should be preceded by the slash (/) character.

Thus,

```
IMLOD / Run at 14:23:06 on October 15, 1983
IMLOD INNAME='CASA' INCLASS='IPOL' INSEQ=3 / Source select
```

are good logical records for history files.

User notes

History files are important. Any program which produces an output image should copy the history file(s) of the input image(s) to the history file of the output image. It should then record in that file all parameters relevant to the function of the program. Parameters which were defaulted by the user should have the actually-used values inserted in the history file. History files need to be readable both to humans and to computers. Thus comments and parameter names should not be overly terse or crowded onto the cards. However, one should also try to avoid having the history files grow without bounds. Thus, excessive commenting and one keyword per card are poor practices. The parsing rules should be those of ANSII-standard Fortran 77 list- directed I/O.

Routines and commons for writing history files

There is a collection of basic routines to create, open, add to, read, write, and close history files. They all use the basic common called /HICOM/ in DHIS.INC:

```
NHIFIL  I          max number of history files open at once
NHIWRD  I          number of words/entry in HITAB
NHIWPL  I          # words / logical record
NHILPR  I          # logical records / physical record
HITAB   I(NHIFIL*NHIWRD)  table describing open history files
```

If HIPNT is the pointer to an entry in HITAB, then the structure is

```
HITAB(NIPNT+0)     logical unit number of file
HITAB(HIPNT+1)     pointer to FTAB for file
NITAB(HIPNT+2)     number of logical records now in file
HITAB(HIPNT+3)     number logical records which will fit in file
HITAB(HIPNT+4)     disk number for file
HITAB(HIPNT+5)     sequential catalog position for image
HITAB(HIPNT+6)     physical record # now in user-supplied buffer
```

The user of the standard routines must cause the HITAB to receive an appropriate size and to be initialized. This may be done by

```
INCLUDE 'DHIS.IHC'
....

CALL NIINIT (n)
```

where n is the number of history files around and is less than 20. The include 'DHIS.INC' declares HITAB(140). If you really must have more than 20 simultaneous HI files, you will need to declare the COMMON and its variables in your own initialization routine. Do follow the example of DHIS.INC and HIINIT, however.

The user may create and open a history file with subroutine HICREA or open an old file with HIOPEN. Among the arguments to these routines is a 256-word buffer which is used as a working buffer by all of the standard history routines. The programmer should not modify the contents of this buffer between the call to HIOPEN (or HICREA) and the corresponding call to HICLOS. Separate buffers must be provided for each open history file. History records are normally added to history files via HIADD. This routine calls HIIO to perform the actual input/output operations (when required only) including expanding the file as needed. HICLOS is used to complete the writing of the file including updating the pointers in record 1 and to close the file.

There are a number of less basic routines which provide history- related services. HISCOP copies the contents of one open history file into another. HENCO1, HENCO2, and HENCOO encode and write to files some of the basic parameters used by most tasks.

Routines and commons for reading history files

At this writing, there are no special routines for reading history files. Normally, one uses HIINIT, HIOPEN, and HICLOS and the common /HICOM/ as described above. HILOCT may be used to return the location of the entry in HITAB (called HIPNT above). With this pointer, one may use HIIO (or simply ZFIO) to read the file one physical record at a time.

C.3.5 Image (MA) file

Overview

Map data files contain the binary pixel values for the n- dimensional "map" array ("image").

Details: Each map file holds only one image. The header which describes that image is stored in the catalog file for the disk on which it resides. (See CB file description in this section.)

Names: Map file physical names are MArsss01, where r is the format revision code and sss is the sequential position of the header in the catalog file for that disk. The logical names for map files, used by users, have no relationship to this physical name.

Record formats

The pixel values are stored in binary form in map files, normally as real values. The image array data are ordered in the standard Fortran order in which the first axis counter varies fastest. Each "plane" of the array (one full cycle of the first two axis counters) must begin on a sector boundary and occupy one or more consecutive sectors as required. Thus, depending on row length and the number of bytes per sector on the local machine, some disk space may be wasted. This waste is tolerated in order to obtain improved IO performance on (the normal) row reads and writes. The full map occupies NGPMAP = CEIL ((NSROW * NAXIS2 * NAXIS3 ...) / NSPG) granules, where NBPS and NSPG are in the common /DCHCOM/ (DDCH.INC) described in Chapter 6 on Disk I/O.

Usage Notes

Map files are always read with the fast IO routines MINIT, MDISK, ZMIO, ZWAIT, et al..

Routines and commons for reading and writing map files

These routines and commons are described in the chapter on Disk I/O.

C.3.6 Plot (PL) file

Overview

Plot files are a generalized representation of a graphics display. They contain scaling information and commands for drawing lines, pixels, and characters. Plot files for a certain type of plot (contour maps, grey scales, etc.) are written by one routine and then another routine must read the plot file and write to a particular graphics device (plotter, graphics terminal, etc.). In AIPS these functions are performed by separate tasks. This two step approach offers several advantages. A plot file may exist for an extended period of time allowing plots to be written to different devices and copies to be generated at later times without duplicating the calculations needed in making the plot. Also only one program for a particular plot need be written instead of one program for each graphics device.

I/O to the plot files consists of reading and writing 256-word blocks. The logical records are of 9 types and vary in length. With the exception of the 'draw pixels' record, logical records do not span the 256-word blocks. Unused space at the end of a block consists of integer zeros. All values in the plot file are I variables or Hollerith characters. This aids in exporting plot files to other computers via tape. Plot files have names of the format PLrsssvv, where r is the format revision code, sss is the Catalog slot number of the associated map, and vv is the version number. Plot files are usually extension files associated with a cataloged map.

Record layouts

(0) The first physical record contains information about the task which created the file. It is not logically part of the "plot file", but is there to provide documentation of the file's origins should it be needed. The first logical record (see below) starts in word 1 of the second physical record in the file. The contents of the first physical record are task-dependent and have the form:

```
FIELD   TYPE        DESCRIPTION
  1.    H(3)        Task name
  2.    I(6)        Date/time of file creation YYYY,MM,DD,HH,MM,SS
  3.    I           Number of words of task parameter data
  4.    R(*)        Task parameter block as transmitted from AIPS
                      (preferably with defaults replaced by the values
                       used).
```

(1) Initialize plot record.
The first logical record in a plot file must be of this type.

```
FIELD  TYPE     DESCRIPTION
  1.   I        Opcode (equal to 1 for this record type).
  2.   I        User number.
  3.   I(3)     Date:  yyyy, mm, dd
  4.   I        Type of plot:  1 = miscellaneous
                                2 = contour
                                3 = grey scale
                                4 = 3D profile
                                5 = slice
                                6 = contour plus polarization lines
                                7 = histogram
```

(2) Initialize for line drawing record. This record provides scaling information needed for a plot. The plot consists of a 'plot window' in which all lines are drawn and a border (defined in terms of character size) in which labeling may be written. The second record in a plot file must be of this type.

```
FIELD  TYPE     DESCRIPTION
  1.   I        Opcode (equal to 2 for this record type).
  2.   I        X Y ratio * 100.  The Ratio between units on the X axis
                and units on the Y axis (X / Y).  For example if the
                decrement between pixels in the X direction on a map
                is twice the decrement in the Y direction the X Y ratio
                can be set to 2 to provide proper scaling.  Some
                programs may ignore this field.  For example IISPL when
                writing grey scale plots to the IIS.
  3.   I        Scale factor (currently 16383 in most applications).
                This number is used in scaling graph positions before
                they are written to disk.  BLC values in field 4 are
                represented on disk by zero and TRC values are
                represented by integers equal to the scale factor).
  4.   I(4)     The bottom left hand corner X and Y values and the top
                right hand X and Y values respectively in the plot
                window (in pixels).
  5.   I(4)     1000 * the fractional part of a pixel allowed to occur
                outside the (integer) range of BLC and TRC (field 4
                above) in line drawing commands
  6.   I(4)     10 * the number of character positions outside the plot
                window on the left, bottom, right, and top respectively
  7.   I(5)     Location of the X Y plane on axes 3,4,5,6,7.  This
                  field is valid only for plots associated with a map.
```

(3) Initialize for grey scale record. This record if needed must follow the 'init for line drawing' record. This record allows proper interpretation of pixels for raster type display devices. Programs that write to line drawing type devices (e.g. the TEKTRONIX 4012) ignore this record.

```
FIELD  TYPE     DESCRIPTION
  1.   I        Opcode (equals 3 for this record type).
  2.   I        Lowest allowed pixel intensity.
  3.   I        Highest allowed pixel intensity.
  4.   I        Number of pixels on the X axis.
  5.   I        Number of pixels on the Y axis.
```

(4) Position record. This record tells a device where to start drawing a line, row/column of pixels or character string.

```
FIELD  TYPE     DESCRIPTION
  1.   I        Opcode (equals 4 for this record type).
  2.   I        scaled x position i.e. a value of 0 represents the
                BLC values defined in the 'init for line drawing'
                record, and a value equal to the scale factor
                represents the TRC value.
  3.   I        Scaled Y position.
```

(5) Draw vector record. This record tells a device to draw a line from the current position to the final position specified by this record.

```
FIELD  TYPE     DESCRIPTION
  1.   I        Opcode (equals 5 for this record type).
  2.   I        Scaled final X position.
  3.   I        Scaled final Y position.
```

(6) Write character string record. This record tells a device to write a character string starting at the current position. .

```
FIELD  TYPE     DESCRIPTION
  1.   I        Opcode (equals 6 for this record type).
  2.   I        Number of characters.
  3.   I        Angle code: 0 = write characters horizontally.
                           1 = write characters vertically.
  4.   I        X offset from current position in characters * 100
  5.   I        Y offset from current position in characters * 100
                  (net position refers to lower left corner of 1st char)
  6.   H(n)     Hollerith characters (n = INT((field2 + 3) / 4))
```

(7) Write pixels record. This record tells a raster type device to write an n-tuple of pixel values starting at the current position. Programs that write to line drawing type devices ignore records of this type.

```
FIELD  TYPE     DESCRIPTION
  1.   I        Opcode (equals 7 for this record type).
  2.   I        Number of pixel values.
  3.   I        Angle code: 0 = write pixels horizontally.
                           1 = write pixels vertically (up).
  4.   I        X offset in characters * 100.
  5.   I        Y offset in characters * 100.
  6.   I(n)     n (equal to field 2) pixel values.
```

(8) Write misc. info to image catalog record. This record tells the programs that write to interactive devices (TEKPL, IISPL) to put up to 20 words of miscellaneous information in the image catalog starting at word I2TRA + 2. This information is interperted by routines such as AU9A (TKSKYPOS, TKMAPPOS, etc.). Routines that write to non- interactive graphics devices (PRTPL) ignore this record.

```
FIELD  TYPE      DESCRIPTION
  1.   I        Opcode (equals 8 for this record type).
  2.   I        Number of words of information.
  3.   I(n)     Miscellaneous info (n=value of field 2).
```

(9) End of plot record. This record marks the end of a plot file.

```
FIELD  TYPE     DESCRIPTION
  1.   I       Opcode (equals 32767 for this record type).
```

Subroutines and Commons for Writing

- GINIT Creates and opens a plot file, initializes the graphics common GPHCOM and writes an 'Initialize plot' record.
- GINITL Writes an 'init for line drawing' record to the plot file.
- GINITG Writes an 'init for grey scale' record to a plot file.
- GPOS Writes a 'position' record to the plot file.
- GVEC Writes a 'draw vector' record to the plot file.
- GCHAR Writes a 'write character string' record to the plot file.
- GRAYPX Writes a 'write pixels' record to the plot file.
- GFINIS Writes an 'end of plot' record to the plot file and closes the file.
- GPHWRT This routine is called by the above routines if I/O is needed. It writes the current buffer to the plot file and zeros the buffer. Then the record count and buffer position pointer are updated in common GPHCOM.
- GPHCOM (common) contains variables used for inter-subroutine communication. It is declared by including DGPH.INC and has the structure:

```
GPHSIZ  I       File size in 256-word blocks
GPHLUN  I       Logical unit number
GPHIND  I       Pointer to FTAB
GPHPOS  I       Position in work buffer of last word used
GPHRRN  I       Number of records written
GPHVOL  I       Disk volume number
GPHNAM  I(12)   Physical file name
GPHX1   R       Leftmost (col) pixel position
                  (as passed in file logical record type 2,
                   field 4 above; i.e. integer part)
GPHX2   R       Rightmost (col) pixel position
GPHY1   R       Lowest (row) pixel position
GPHY2   R       Highest (row) pixel position
GPHTLO  I       Lowest allowed pixel value
GPHTHI  I       Highest allowed pixel value
```

This common is handled fully by the routines listed here. A user of these routines only needs to declare the common in his main program to insure that adequate space is reserved for it.

Subroutines and Commons for Reading.

- TVPL Reads a plot file and writes the corresponding commands to the tv device.
- TVSPCL (common) for inter-subroutine communication for IISDRW and its subroutines.
- PRTDRW Reads a plot file and builds a bit map representation on disk. This bit map is then written to a printer/plotter.
- CPRT (common) allows inter-subroutine communication for PRTDRW and its subroutines.
- TEKDRW Reads a plot file and writes the corresponding commands to the TEKTRONIX 4012 graphics screen.
- TVCHAR (common) for inter-subroutine communication for TEKDRW and its subroutines.
- IMAHDR (common) plot file catalog header information.used in IISDRW, PRTDRW, and TEKDRW.

C.3.7 Slice (SL) file

Overview

Slice files are usually extension files associated with a map file. The slice file contains interpolated data points that lie along a vector from one map pixel to another. End points may be fractions of a pixel. The slice file may also contain the parameters for a number of models of the slice file. Each model may contain up to 4 gaussian components.

Details: SL files use a modified EXTINI-EXTIO file structure. The logical record length is 256 words. The number of data points in the slice is determined by the following algorithm:

```
If the length of the vector is greater than 1024 then 4096.
If the length of the vector is greater than 512 then 2048.
If the length of the vector is greater than 256 then 1024.
otherwise the no. of points is 512.
```

Names: The system name for a slice file is SLrsssvv where r is the release code, sss is the catalog slot number of the associated map file, and vv is the version number (starting at 1) of the slice file.

Record formats

Slice files are sequential files consisting of 256 word records. The first record in the file is of the type created by EXTINI and contains useful miscellaneous information. The 2nd record contains the inputs the user specified when creating the slice file. The slice data points start in record three and continue for as many records as necessary to use up the data. Models, if extant, immediately follow the data using one record per model.

WORD	TYPE	DESCRIPTION
1	I	# 512-byte records in the existing file
2	I	# logical records to extend the file when req.
3	I	max. # of logical records
4	I	current number of logical records
6	I	# values per logical record.
7	I	# of logical records per physical record, if neg then the # of physical records per logical record.
8	H*6	Creation task name
11	I(6)	Creation date, time
17	H*48	File name
29	I	Volume number on which file resides.
30	H*6	Last write-access task

33	I(6)	Last write-access time,date
39	I(18)	reserved. (53-56 used by EXTIO.
57	I	Number of slice data points.
58	I	Number of model records currently in the file.
59	I	The record that may contain the first slice model. Includes header record and numbers from one.

(1) The second physical record
All default values for the fields have been filled in.

WORD	TYPE	DESCRIPTION
1	H*6	Program name (slice).
4	I(3)	Date (year, month, day).
7	I(3)	Time (hour, min, sec).
10	I	Number of words in the following inputs section of the record (25 at this time).
11	R	User number of the map file.
12	H*12	Name of the associated map file.
15	H*6	Class of the associated map file.
17	R	Sequence number of the associated map file.
18	R	Disk volume number of slice and map files.
19	R	Type of the map file.
20	R(7)	Starting pixel of the slice vector.
27	R(7)	Ending pixel of the slice vector.
34	R	Maximum value of slice pixels.
35	R	Minimum value of slice pixels.

(1) Slice data points

WORD	TYPE	DESCRIPTION
1	R(256)	Slice data points in floating point.

(2) Slice models.

WORD	TYPE	DESCRIPTION
1	R(12)	Parameters for up to 4 gaussian components in the order maximum amplitude (physical units), position (slice point), half width (slice points), maximum amplitude ...
12	R(12)	Errors for corresponding parameters.
25	I	Slice points not fitted from the beginning of the slice.
26	I	Slice points not fitted from the end of the slice.
27	I	Number of Gaussians.
28	I(3)	Date as year, month, day.
31	I(3)	Time as hours, minutes, seconds.
34	I(4)	Minus ones indicate initial guess for position of corresponding gaussian component held constant.
38	I(4)	Minus ones indicate initial guess for maximum amplitudes held constant.
42	I(4)	Minus ones indicate initial guess for half widths held constant.

User notes.

When calling EXTINI use LREC=256.

Routines to write SL files:

EXTINI and ZFIO, SL files are currently written by SLICE and SLFIT.

Routines to access SL files:

EXTINI and ZFIO are used to access SL files.

C.3.8 Uv data (UV) file

Overview

UV files contain interferometric data or single dish data randomly sampled on the sky. The files consists of a series of records each with a set of "random" parameters and a rectangular data array. The "random" parameters are descriptions of the associated data array and give such information as u, v, and w or feed number. The structure of these records is flexible and depends on the particular application.

Details: A UV file contains an arbitrary number of records in an arbitrary order. A description of the structure, number and order of records is given in the associated catalog header (CB) record. A number of extention tables may be need to fully interprete the data.

Names: UV file physical names are UVrsss01, where r is the format revision code and sss is the sequential position of the header in the catalog file for that disk. The logical names for UV files, used by users, have no relationship to this physical name.

Record formats

All data are stored as single precision values. The exception to this is "compressed" format data in which "visibility" data may be stored in a machine dependent form. Data are stored with random parameters first followed by the data array in Fortran order. There is no unused space between records and records may span disk sectors.

Usage Notes

UV files are always read with the fast IO routines UVINIT, UVDISK, ZMIO, ZWAIT, et al..

Routines and commons for reading and writing UV files

These routines and commons are described in the chapters on Disk I/O and Calibration.

C.4 Table details

This section describes the standard AIPS tables files. These are user data files using the conventions described in the chapter on Tables in this volume. These are used heavily in the calibration package.

C.4.1 Antenna (AN) table

Overview

This extension table for a uv data set contains information about the antennas and the array geometry including conversion from atomic time to sidereal time.

Name: The file name is ANrsssvv where r is the revision code (A, B...), sss=catalog number and vv = version number.

File Structure

Logical records consist of the information for a single antenna. In the case of an orbiting antenna the elements of the orbit are given. For each antenna two polarization feeds are assumed (A and B) For orbiting antennas the orbital elements are given by ORBPARM. The file header record contains the following KEYWORDs:

Keyword	type	Description
ARRAYX	D	Array center X coordinate (meters, earth center)
ARRAYY	D	Array center Y coordinate
ARRAYZ	D	Array center Z coordinate
GSTIA0	D	GST at IAT=0 (degrees) on ref. date
DEGPDY	D	Earth rotation rate (deg/IAT day)
FREQ	D	Obs. Reference Frequency for subarray(Hz)
RDATE	A	Reference date as 'DD/MM/YY'
POLARX	E	Polar position X (meters) on ref. date
POLARY	E	Polar position Y (meters) "
UT1UTC	E	UT1-UTC (time sec.) "
IATUTC	E	IAT-UTC (time sec.) "
ARRNAM	A	Array name
NUMORB	I	Number of orbital parameters
NOPCAL	I	Number of polarization calibration constants.
POLTYPE	A	Feed polarization parameterization, only if the feed parameters have been entered. 'APPROX ' => linear approximation 'ORI-ELP ' => orientation-ellipticity 'X-Y LIN ' => lin. approx. for lin. polarized (X-Y) data.

Table entries:

Title	Units	Code	Description
ANNAME		8A	Station name
STABXYZ	meters	3D	X,Y,Z offset from array center
ORBPARM		*D	Orbital parameters (see note 1)
NOSTA		1I	Station number
MNTSTA		1I	Mount type, 0=altaz, 1=equatorial, 2=orbiting
STAXOF	meters	1E	Axis offset
POLTYA		1A	Feed A feed poln. type 'R','L','X','Y'
POLAA	degrees	1E	Feed A feed position angle.
POLCALA		*E	Feed A poln. cal parameter. (note 2)
POLTYB		1A	Feed B feed poln. type 'R','L','X','Y'
POLAB	degrees	1E	Feed B feed position angle.
POLCALB		*E	Feed B poln. cal parameters.

User Notes

The "code" column is element_count + basic type code. basic type codes: D=Double precision, E=single precision, A=character, I=integer, L=logical, X=bit.

1. ORBPARM is an array whose dimension is given by the header keyword NUMORB
2. POLCALA and POLCALB are arrays whose dimension is given by the header keyword NOPCAL.

Routines to access AN files

ANTINI and TABAN, AN files are currently written by UVLOD, VLBIN, MK3IN and SETAN.

C.4.2 Baseline dependent calibration (BL) table

Overview

This table contains information for baseline dependent (antenna pair) calibration. The complex gains for each baseline necessary to correct for non-antenna based errors. These errors are assumed to consist of a multiplicative and an additive portion.

Names: The file name is BLrsssvv where r is the format revision code, sss=catalog number and vv = version number.

File structure

Logical records consist of the information for a single baseline at a given time for all IF and polarizations. The file header record contains the following KEYWORDs:

Keyword	Code	Description
NO_ANT	I	The number of antennas for which there is information (actually highest antenna number).
NO_POL	I	The number of polarizations
NO_IF	I	The number of IFs.

Table entries:

Title	Units	code	Description
TIME	Days	1E	Time of center of interval since 0h on reference day.
SOURCE ID		1I	Identification number of the source used.
SUBARRAY		1I	Subarray number
ANTENNA1		1I	First antenna number.
ANTENNA2		1I	Second antenna number.
FREQ ID		1I	Frequency group id.
REAL M1		*E	Real part of multiplicative factor for first polarization. (see note 1)
IMAG M1		*E	Imag. part of multiplicative factor for first polarization.
REAL A1		*E	Real part of additive correction for first polarization.
IMAG A1		*E	Imag. part of additive correction for first polarization.

The following are present only if NO_POL = 2

Title	Units	code	Description
REAL M2		*E	Real part of multiplicative factor for second polarization. (see note 1)
IMAG M2		*E	Imag. part of multiplicative factor for

		second polarization.
REAL A2	*E	Real part of additive correction for second polarization.
IMAG A2	*E	Imag. part of additive correction for second polarization.

User notes

The "code" column is element_count + basic type code. basic type codes: D=Double precision, E=single precision, A=character, I=integer, L=logical, X=bit.

1. "REAL M1", "IMAG M1", "REAL A1", "IMAG A1", "REAL M2", "IMAG M2", "REAL A2" and "IMAG A2" are arrays whose dimensions are given by the header keyword NO_IF.

2. This table will in practice be used only by standard interface routines.

Routines to access BL files

The AIPS routines BLINI and TABBL will create/read/write BL tables. Chapter 13 gives a detailed description of routines to access tables files.

C.4.3 Bandpass calibration (BP) table

Overview

This table contains information for bandpass calibration; this is frequency channel dependent calibration. The bandpass correction functions are factored into antenna based components.

Names: The file name is BPrsssvv where r is the format revision code, sss=catalog number and vv = version number.

File structure

Logical records consist of the information for a single antenna at a given time for all IF and polarizations. The file header record contains the following KEYWORDs:

Keyword	Code	Description
NO_ANT	I	The number of antennas for which there is information (actually highest antenna number).
NO_POL	I	The number of polarizations
NO_IF	I	The number of IFs.
NO_CHAN	I	The number of spectral channels present
STRT_CHN	I	First channel number present in file.

Table entries:

Title	Units	code	Description
TIME	Days	1D	Time of center of interval since 0h on reference day.
INTERVAL	Days	1E	Time interval covered.
SOURCE ID		1I	Identification number of the source used.
SUBARRAY		1I	Subarray number
ANTENNA		1I	Antenna number.
BANDWIDTH	Hz	1E	Bandwidth of individual channels

IF FREQ	Hz	*D	Reference frequency for each IF
FREQ ID		1I	Frequency group id.
REFANT 1		1I	Reference antenna
REAL 1		*E	Real part of channel gains for first polarization. (see note 1)
IMAG 1		*E	Imag. part of channel gains for first polarization.

The following are present only if NO_POL = 2

Title	Units	code	Description
REFANT 2		1I	Reference antenna
REAL 2		*E	Real part of channel gains for second polarization. (see note 1)
IMAG 2		*E	Imag. part of channel gains for second polarization.

User notes

The "code" column is element_count + basic type code. basic type codes: D=Double precision, E=single precision, A=character, I=integer, L=logical, X=bit.

1. "REAL 1", "IMAG 1", "REAL 2", and "IMAG 2" are arrays whose dimensions are given by the product of the header keywords NO_CHAN, and NO_IF. They can be considered to be arrays dimensioned (NO_CHAN,NO_IF).
2. This table will in practice be used only by standard interface routines.

Routines to access BP files

The AIPS routines BPINI and TABBP will create/read/write BP tables. Chapter 13 gives a detailed description of routines to access tables files.

C.4.4 Clean Components (CC) table

Overview

CLEAN components are stored in this table by CLEAN tasks during the deconvolution process or by modeling programs. CLEAN point components represent a set of sources which when convolved with the dirty beam and added to the residual map gives the original dirty map. The Fourier transform of these POINT components should, after CLEAN converges, reproduce the observed data. Other tasks write CC files containing Gaussian or other model fits to images or UV data.

Names: The file name is CCdsssvv, where d is the update code and sss is the sequential position of the image header in the catalog file for that disk and vv is the version number.

File Structure

Logical records consist of a single model component. If only three columns are present then the file contains only point components. CC tables headers contain no KEYWORDs.

Table entries:

Title	Units	code	Description
FLUX	Jy	1E	Flux density of component.

```
DELTAX          degrees   1E  Offset from reference pixel in X direction
                              (RA for unrotated images)
DELTAY          degrees   1E  Offset from reference pixel in Y direction
                              (Dec for unrotated images)
```

The following are optional:

```
MAJOR AX        degrees   1E  Gaussians: Major axis size (FWHM)
                              Spheres: Radius
MINOR AX        degrees   1E  Gaussian: Minor axis size.
POSANGLE        degrees   1E  Gaussian: position angle (from N thru E)
TYPE OBJ                  1E  Model type:
                                0 => point model.
                                1 => gaussian on sky.
                                2 => gaussian convolved by observation.
                                3 => uniform, optically thin sphere.
```

User notes

The "code" column is element_count + basic type code. basic type codes: D=Double precision, E=single precision, A=character, I=integer, L=logical, X=bit.

The positions given are those projected onto the same plane as the original image *i.e.* the number of pixels from the reference pixel times the pixel spacing. If accurate positions are desired the appropriate transformation must be made.

Routines for accessing CC files

AIPS routines CCINI, TABINI, TABIO allow read and write access to CC tables. The tasks APCLN, SDCLN and MX (regular cleaning), IMFIT, JMFIT and UVFIT may write in CC files. The tasks PRTCC and PRTAB will print the contents of CC tables. Chapter 13 describes the routines to access tables files.

C.4.5 Calibration (CL) table

Overview

This extension table for a uv data set contains calibration information and the total observed geometric observables. This will include both calibration information passed to AIPS from external sources and that derived from AIPS calibration software. This information will not have been applied to the data. This table will need to be updated using the contents of the Solution table (SN) when satisfactory gain solutions have been derived. The information in this table will be used to calibrate data.

The records also contain the geometric observables: total group and phase delay and their time derivatives divided into several parts. These values should be updated whenever this table is updated using a Solution table.

The total model group and phase delays are divided into three contributions: 1) the geometric delay which will include terms which can be described by a single 24 hour sinusoid, 2) a atmospheric term, and 3) a "clock" (meaning everything else) term. Interpolation of the geometric term to times other than the reference time is done using the phase of the sinusoid at the reference time (GEOPHASE) and the earth rotation rate at the reference time (GEORATE). The other terms are interpolated by using the first order time derivatives. The total model values are given by the sum of these three terms. These values are the total observed values and are not to be applied to the data; the residual values are the corrections which need to be applied to the data.

Names: The file name is CLrsssvv where r is update code, sss=catalog number and vv = version number.

File Structure

Logical records consist of the information for all IFs for a single antenna. A pair of IFs consists of the IFs which are cross correlated, for example, right and left circular polarized IFs at the same frequency. In practice a pair will consist of a single IF or a pair of orthogonally polarized IFs. All IF pairs must be identical, i.e., have the same number and types of IF. The file header record contains the following KEYWORDs:

Keyword	Code	Description
NO_ANT	I	The number of antennas for which there is information (actually highest numbered antenna)
NO_POL	I	The number of polarizations.
NO_IF	I	The number of IFs.
MGMOD	E	The mean gain modulus for the entire table.

Table entries:

Title	Units	code	Description
TIME	Days	1D	Time of center of interval since 0h on reference day.
TIME INTERVAL	Days	1E	Interval over which solution was obtained.
SOURCE ID		1I	Identification number of the source used.
ANTENNA NO.		1I	Antenna number.
SUBARRAY		1I	Subarray number
FREQ ID		1I	Frequency group identifier
I.FAR.ROT		1E	Ionospheric Faraday rotation (rad/m**2)
GEODELAY	Seconds	1D	The total geometric delay at the reference time; should include only sinusoidal terms.
GEOPHASE	Turns	1D	Fractional (0-1) phase of the reference time, used to interpolate GEODELAY.
GEORATE	Hz	1D	The time derivative of GEOPHASE = earth rotation rate at the reference time.
DOPPOFF	Hz	*E	Doppler frequency offset. (see note 1)
CLKGD 1	Seconds	*E	Clock group delay for pol. 1.
DCLKGD 1	Sec/sec	*E	Time derivative of CLKGD 1.
CLKPD 1	Seconds	*E	Clock phase delay for pol. 1.
DCLKPD 1	Sec/sec	*E	Time derivative of CLKPD 1.
ATMGD 1	Seconds	*E	Atmospheric group delay for pol. 1.
DATMGD 1	Sec/sec	*E	Time derivative of ATMGD 1.
ATMPD 1	Seconds	*E	Atmospheric phase delay for pol. 1.
DATPGD 1	Sec/sec	*E	Time derivative of ATMPD 1.
REAL 1		*E	Real part of the gain calibration factor; calibrated = raw * CALi * conj(CALj) for antennas i,j.
IMAG 1		*E	Imaginary part of the gain calibration.
DELAY 1	Seconds	*E	Residual delay of pol. 1
RATE 1	Sec/sec	*E	Residual rate of pol. 1 Note: DELAY n and RATE n are the residual values by which the data are to be corrected during calibration.
TSYS 1	Kelvins	*E	System temperature of pol. 1.

WEIGHT 1	*E	Weight of 1st polarization.
REFANT 1	*I	Reference antenna used for solution.

The following are present only if NO_POL = 2

Title	Units	code	Description
CLKGD 2	Seconds	*E	Clock group delay for pol. 2.
DCLKGD 2	Sec/sec	*E	Time derivative of CLKGD 2.
CLKPD 2	Seconds	*E	Clock phase delay for pol. 2.
DCLKPD 2	Sec/sec	*E	Time derivative of CLKPD 2.
ATMGD 2	Seconds	*E	Atmospheric group delay for pol. 2.
DATMGD 2	Sec/sec	*E	Time derivative of ATMGD 2.
ATMPD 2	Seconds	*E	Atmospheric phase delay for pol. 2.
DATPGD 2	Sec/sec	*E	Time derivative of ATMPD 2.
REAL 2		*E	Real part of the gain calibration factor for 2nd polarization.
IMAG 2		*E	Imaginary part of the gain calibration.
DELAY 2	Seconds	*E	Residual delay of pol. 2
RATE 2	Sec/sec	*E	Residual rate of pol. 2
TSYS 2	Kelvins	*E	System temperature of pol. 2.
WEIGHT 2		*E	Weight of 2nd polarization.
REFANT 2		*I	Reference antenna used for solution.

User Notes

The "code" column is element_count + basic type code. basic type codes: D=Double precision, E=single precision, A=character, I=integer, L=logical, X=bit.

1. "DOPPOFF", "REAL n", "IMAG n", "DELAY n", "RATE n", "TSYS n", "WEIGHT n" and "REFANT n" are arrays whose dimensions are given by the header keyword NO_IF.

2. The geometric observables should be updated every time the file is updated from a Solution (SN) table.

AIPS routines to access CL files

CALINI and TABCAL read and write CL tables. Also chapter 13 gives a detailed description of routines to access tables files.

C.4.6 Frequency (CH) table

Overview

Note: this table is now replaced by the FQ table and is included here for historical completeness.

This extension table for a uv data set contains relevant information about the IFs in the raw uv data file. For these purposes an IF consists of the output from a receiver or baseband converter; all measured polarizations at the same frequency are considered part of the same IF. IFs can be arbitrarily spaced in frequency. Examples are the A-C and B-D IFs of the VLA and the output of independent video converters in VLBI recorders. Individual, regularly spaced frequency or delay channels derived from the correlation of such IFs are not themselves considered IFs.

Names: The file name is CHrsssvv where r is the format revision code (A, B,...) disk number, sss=catalog number and vv = version number.

File Structure

The table entries give the frequency offset to the center of the IF and the sideband of each IF. The file header record contains no KEYWORDs.

Table entries:

Title	Units	code	Description
IF NO.		1I	The IF pixel number of the entry.
FREQUENCY OFFSET	Hz	1D	Frequency offset from ref. freq. True = Ref + Offset
SIDEBAND		1I	Sideband of each IF -1 => 0 video freq. is high freq. end 1 => 0 video freq. is low freq. end

User Notes

The "code" column is element_count + basic type code. basic type codes: D=Double precision, E=single precision, A=character, I=integer, L=logical, X=bit.

The true frequency of the observations are the signed sums of the reference frequency in the subarray antenna (AN) file, the peculiar source doppler offset from the source table (SU file) Any time dependent doppler correction from the CL table and the IF frequency offset from this file.

Routines to access CH files

AIPS utility routine CHNDAT will read of create/write these tables; CHNCOP will copy all or a subset of the table. Also see chapter 13 for a detailed description of routines to access tables files.

C.4.7 Single dish calibration (CS) table

Overview

This extension table for a single dish data set contains position, gain and baseline offset corrections to be applied to single dish data in a uv data like format.

Names: The file name is CSdsssvv where d = release code, sss=catalog number and vv = version number.

File structure

Logical records consist of the information for all IFs for a single beam at a give time. The file header record contains the following KEYWORDs:

Keyword	Code	Description
NO_BEAM	I	The number of beams for which there is information
NO_POL	I	The number of polarizations.
NO_IF	I	The number of IFs.

Table entries:

Title	Units	code	Description
TIME	Days	1E	Time since 0h on reference day.
RA	Deg	1E	apparent RA at center time
DEC	Deg	1E	apparent Dec at center time
BEAM NO.		1I	Beam number.
SUBARRAY		1I	Subarray number
FACTOR 1		*E	Gain factor calibrated = factor * (raw + offset) for 1st polarization. (see note 1)
OFFSET 1		*E	offset
RAOFF 1	Degrees	*E	RA correction (to be added)
DECOFF 1	Degrees	*E	Declination correction

The following are present only if NO_POL = 2

Title	Units	code	Description
FACTOR 2		*E	Gain factor calibrated = factor * (raw + offset) for 2nd polarization. (see note 1)
OFFSET 2		*E	offset
RAOFF 2	Degrees	*E	RA correction
DECOFF 2	Degrees	*E	Declination correction

User notes.

The "code" column is element_count + basic type code. basic type codes: D=Double precision, E=single precision, A=character, I=integer, L=logical, X=bit.

1. "FACTOR n", "OFFSET n", "RAOFF n" and "DECOFF n" are arrays whose dimensions are given by the header keyword NO_IF.

Routines to access CS files

Routine CSINI will create/initialize/open/read-keywords-from/ write-keywords-to a solution table and TABCS will do I/O. Also chapter 13 gives a detailed description of routines to access tables files.

C.4.8 Flag (FG) table

Overview

This extension table for a uv data set contains the editing information. This information may or may not have been applied to the data depending on the uv data file type. This file will contain a list of specifications of data to be flagged.

Names: The file name is FGrsssvv where r the AIPS update code, sss=catalog number and vv = version number.

File Structure

Each logical record consists of a specification of data to be flagged. These specifications are independent and may overlap. Data is to be rejected if it is specified in any flagging record that is currently selected. Any entry may be temporarily disabled by deselecting that table entry. The file header record contains no KEYWORDs.

Table entries:

Title	Units	code	Description
SOURCE		1I	Source id number of the source to be flagged. 0 => all sources
SUBARRAY		1I	Subarray number, 0 => all
ANTS		2I	1st element = number of the first antenna, 0 => all baselines to all antennas flagged. 2nd element = number of the second antenna, 0 => all baselines to ANTS(1) flagged
TIMERANG	Days	2E	Beginning and end time of flagging in the same system as the data is labeled. Both = 0 => data flagged for all times.
IFS		2I	Numbers of first and last IF group to be flagged.
CHANS		2I	First and last channel number in IF group.
PFLAGS		4X	Array of flags for the polarizations. 4 are kept even if fewer polarizations are present. Flags in same order as data, bit set => correlator value flagged.
REASON		24A	Reason code for flagging (24 char)

User Notes

The "code" column is element_count + basic type code. basic type codes: D=Double precision, E=single precision, A=character, I=integer, L=logical, X=bit.

Routines to access FG files

FLGINI and TABFLG do I/O to flag files. Chapter 13 gives a detailed description of routines to access tables files.

C.4.9 Frequency (FQ) table

Overview

This extension table for a uv data set contains relevant information about the IFs in the raw uv data file. A data file may contain data made with different sets of IF frequencies or bandwidths. Data containing more than one such frequency group (or set of bandwidths) will have a random parameter indicating an entry in the FQ table which gives the details of the frequencies and/or bandwidths.

Names: The file name is FQrsssvv where r is the format revision code (A, B,...), sss=catalog number and vv = version number.

File Structure

The table entries give the frequency offset to the center of the IF, bandwidth and the sideband of each IF. The file header record contains the following KEYWORD:

Keyword	Code	Description
NO_IF	I	The number of IFs.

Table entries:

Title	Units	code	Description
FRQSEL		1I	Frequency group identifier
IF FREQ	Hz	*D	Frequency offset from ref. freq. (see note 1)
CH WIDTH	Hz	*E	Channel bandwidth (note 1)
TOTAL BANDWIDTH	Hz	*E	Total bandwidth (note 1)
SIDEBAND		*I	Sideband of each IF (note 1) -1 => 0 freq. is high freq. end 1 => 0 freq. is low freq. end

User Notes

The "code" column is element_count + basic type code. basic type codes: D=Double precision, E=single precision, A=character, I=integer, L=logical, X=bit.

1. "IF FREQ", "CH WIDTH", "TOTAL BANDWIDTH", and "SIDEBAND" are arrays whose dimensions are given by the header keyword NO_IF.
2. The true frequency of the observations are the signed sums of the reference frequency in the subarray antenna (AN) file, the peculiar source doppler offset from the source table (SU file), any time dependent doppler correction from the CL table and the IF frequency offset from this file.

Routines to access FQ files

AIPS utility routine CHNDAT will read or create/write these tables; CHNCOP will copy all or a subset of the table. FQINI and TABFQ also provide access to FQ tables. Also see chapter 13 for a detailed description of routines to access tables files.

C.4.10 Index (NX) table

Overview

This extension table for a uv data set contains an index of the data file. There is an entry for each scan which gives the source, time range, range of visibility numbers and other necessary information. A scan is defined here to consist of a sequence of data from a given subarray which terminates when the source, observing band or observing mode changes or there is a gap in the data much longer than the integration time of the data.

Names: The file name is NXrsssvv where r is the format revision code, sss=catalog number and vv = version number.

File Structure

Logical records consist of the information for a single scan on a single subarray. The file header record contains no KEYWORDs.

Table entries:

Title	Units	code	Description
TIME	Days	1E	Time of center of interval since 0h on reference day.
TIME INTERVAL	Days	1E	Interval over which solution was obtained.

SOURCE ID	1I	Identification number of the source used.
SUBARRAY	1I	Subarray number
START VIS	1I	First visibility number (0 relative) of data in this scan.
END VIS	1I	Last visibility number (0 relative) of data in this scan.
FREQ ID	1I	Frequency group identifier.

User Notes

The "code" column is element_count + basic type code. basic type codes: D=Double precision, E=single precision, A=character, I=integer, L=logical, X=bit.

Routines to access NX files

The AIPS routines NDXINI and TABNDX will create/read/write NX tables. Chapter 13 gives a detailed description of routines to access tables files.

C.4.11 Solution (SN) table

Overview

This extension table for a uv data set contains gain solutions derived from calibration software in AIPS. This information may or may not have been applied to the data or calibration "CL" table depending on the uv data file type. For multi–source files, the SN table is to be applied to the CL table; for single-source data files, the SN table is applied directly to the data. The table keyword APPLIED indicates if the appropriate application has been made.

Names: The file name is SNrsssvv where r = format revision code, sss=catalog number and vv = version number.

File Structure

Logical records consist of the information for all IFs for a single antenna. The file header record contains the following KEYWORDs:

Keyword	Code	Description
NO_ANT	I	The number of antennas for which there is information
NO_POL	I	The number of IFs per IF pair.
NO_IF	I	The number of IF pairs.
NO_NODES	I	The number of interpolation nodes.
MGMOD	E	The mean gain modulus for the entire table.
APPLIED	L	If true then the table has already been applied.
TYPE	I	"Solution" type: 1=>"Clock", 2=>"atmosphere" This is to be used to determine which total model parameters are to be updated.
RA_OFF1	E	The right ascension offset of the first interpolation node. (degrees)
DEC_OFF1	E	The declination offset of the first
...		
RA_OFFn	E	The right ascension offset of the n th interpolation node n = NO_NODES.
DEC_OFn	E	The declination offset of the n th interpolation node n = NO_NODES.

Table entries:

Title	Units	code	Description
TIME	Days	1D	Time of center of interval since 0h on reference day.
TIME INTERVAL	Days	1E	Interval over which solution was obtained.
SOURCE ID		1I	Identification number of the source used.
ANTENNA NO.		1I	Antenna number.
FREQ ID		1I	Frequency group identifier
I.FAR.ROT		1E	Ionospheric Faraday rotation (rad/m**2)
SUBARRAY		1I	Subarray number
NODE NO.		1I	Node number
REAL 1		*E	Real part of the gain calibration factor; calibrated = raw * CALi * conjg(CALj) for antennas i,j, for 1st polarization. (see note 1)
IMAG 1		*E	Imaginary part of the gain calibration.
DELAY 1	Seconds	*E	Residual pair delay for pol. 1.
RATE 1	Sec/sec	*E	Residual phase delay rate for pol. 1.
WEIGHT 1		*E	Weight of 1st polarization
REFANT 1		*I	Reference antenna used for solution.

The following are present only if NO_POL = 2

Title	Units	code	Description
REAL 2		*E	Real part of the gain calibration factor; calibrated = raw * CALi * conjg(CALj) for antennas i,j, for 2nd polarization.
IMAG 2		*E	Imaginary part of the gain calibration.
DELAY 2	Seconds	*E	Residual pair delay for pol. 2.
RATE 2	Sec/sec	*E	Residual phase delay rate for pol. 2.
WEIGHT 2		*E	Weight of 2nd polarization.
REFANT 2		*I	Reference antenna used for solution.

User Notes

The "code" column is element_count + basic type code. basic type codes: D=Double precision, E=single precision, A=character, I=integer, L=logical, X=bit.

1. "REAL n", "IMAG n", "DELAY n", "RATE n", "WEIGHT n" and "REFANT n" are arrays whose dimensions are given by the header keyword NO_IF.

Routines to access SN files

Routine SNINI will create/initialize/open/read-keywords-from/ write-keywords-to a solution table and TABSN will do I/O. Also chapter 13 gives a detailed description of routines to access tables files.

C.4.12 Source (SU) table

Overview

This extension table for a uv data set contains relevant information about the sources in a raw uv data file. This includes positions.

Names: The file name is SUrsssvv where r is the format revision code, sss=catalog number and vv = version number.

File Structure

Each logical record consists of the position and other information about a source in the raw uv data file. Sources are distinguished in the data file by a source ID number.

For moving sources the apparent position is for 0 hours IAT on the reference day defined in the subarray AN file. The motion from this position is given by PMRA and PMDEC. The file header record contains the following KEYWORDs:

Keyword	code	Description
NO_IF	I	The number of IF pairs.
VELTYP	A	Velocity type
VELDEF	A	Velocity definition 'RADIO','OPTICAL'

Table entries:

Title	Units	code	Description
ID. NO.		1I	The source identification number.
SOURCE		16A	Name of the source to be flagged, (16 Char)
QUAL		1I	Source qualifier
CALCODE		4A	Calibrator code (4 char)
IFLUX	Jy	*E	Flux density at reference frequency (Ipol) (See note 1)
QFLUX	Jy	*E	Flux density at reference frequency (Qpol)
UFLUX	Jy	*E	Flux density at reference frequency (Upol)
VFLUX	Jy	*E	Flux density at reference frequency (Vpol)
FREQOFF	Hz	*D	Frequency offset (note 1)
BANDWIDTH	Hz	1D	Bandwidth of channel
RAEPO	degrees	1D	Right ascension at standard mean epoch
DECEPO	degrees	1D	Declination at standard mean epoch
EPOCH	years	1D	Date in years since year 0.0 of the standard epoch.
RAAPP	degrees	1D	apparent Right ascension at 0h IAT on reference day in uv file header.
DECAPP	degrees	1D	apparent declination at 0h IAT on reference day in uv file header.
LSRVEL	m/sec	*D	LSR velocity (see note 1)
RESTFREQ	Hz	*D	Line rest frequency (note 1)
PMRA	deg/day	1D	Proper motion in RA
PMDEC	deg/day	1D	Proper motion in Dec

User Notes

The "code" column is element_count + basic type code. basic type codes: D=Double precision, E=single precision, A=character, I=integer, L=logical, X=bit.

1. Entries with dimensions marked '*' are arrays whose dimension is given by the header keyword NO_IF.

2. The true frequency of the observations are the signed sums of the reference frequency in the catalog header, the peculiar source frequency offset from this table a time dependent doppler offset from the CL table and the IF frequency offset from the FQ table.

Routines to access SU files

AIPS routines SOUINI and TABSOU create/read/write SU tables. Chapter 13 gives a detailed descriptions of routines to access tables files.

C.5 Task Specific Tables

A number of tasks use temporary tables to keep track of information. These are described in the following sections.

C.5.1 SPFLG baseline (BL) table

Overview

NOTE: This temporary table is not to be confused with the standard calibration BL table. This temporary extension table for a uv data set contains baseline numbers and is used internally by SPFLG.

Names: The file name is BLrsssvv where r the AIPS format revision code, sss=catalog number and vv = version number.

File Structure

The file header record contains no KEYWORD:

Table entries:

Title	Units	code	Description
ANTENNA1		1I	First antenna number
ANTENNA2		1I	Second antenna number.
BASELINE		1I	Baseline code number.

User Notes

The "code" column is element_count + basic type code. basic type codes: D=Double precision, E=single precision, A=character, I=integer, L=logical, X=bit.

Routines to access SPFLG type BL files

uses TABINI and TABIO to read and write its BL tables.

C.5.2 SPFLG/TVFLG Temporary Flag (FC) table

Overview

This extension table for a uv data set contains a temporary copy of editing instructions. These are used internally by tasks TVFLG, and SPFLG.

Names: The file name is FCrsssvv where r the AIPS update code, sss=catalog number and vv = version number.

File Structure

The details of these files are identical to FG tables.

C.5.3 ANCAL System Temperature (TY) table

Overview

This extension table for a uv data set contains system temperatures and amplitude calibration factors and is used internally by ANCAL.

Names: The file name is TYrsssvv where r the AIPS update code, sss=catalog number and vv = version number.

File Structure

The file header record contains the following KEYWORD:

Keyword	code	Description
NO_IF	I	The number of IFs.

Table entries:

Title	Units	code	Description
TIME	Day	1R	Time in days
ANTENNA NO.		1I	Antenna number
CALFAC		*R	Amplitude calibration factor 1 / IF
TSYS	Kelvin	*R	System temperature 1/IF

User Notes

The "code" column is element_count + basic type code. basic type codes: D=Double precision, E=single precision, A=character, I=integer, L=logical, X=bit.

"CALFAC" and "TSYS" are arrays of dimension NO_IF.

Routines to access TY files

TYINI and TABTY do I/O to TY files. Chapter 13 gives a detailed description of routines to access tables files.

Index