Going AIPS:
A Programmers Guide to the NRAO Astronomical Image Processing System

W. D. Cotton and a cast of AIPS

Version 15 July 85

VOLUME 2

ABSTRACT

This is the second of a two-volume manual for persons wishing to write programs using the NRAO Astronomical Image Processing System (AIPS). This volume contains information about some of the more detailed features of the AIPS system.

CONTENTS

CHAPTER 12 USING THE ARRAY PROCESSORS

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, TV displays, array processors, and plotters which are covered elsewhere. Many of the same routines used for disk I/O are also used for I/O to other devices but their use may be modified to suit the physical properties of the particular device. 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 principle 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.

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.

9.2.1 Opening Tape Files

Tape files are opened using ZOPEN in a way similar to disk files. Details about ZOPEN and examples of its use can be found in the chapter on disk I/O. However, to tell the AIPS routines that the file is on a tape drive and to specify which tape drive, the LUN and file name are different from those used for disk files. The LUN for tape files must be 31 or 32. When constructing the name of the file using ZPFIL use 'MT' as the file type and the (one relative) tape drive number as the volumn number, the rest of the values sent to ZPHFIL are ignored by ZOPEN and are arbitrary.

9.2.2 Positioning Tapes

Once the file has been opened in AIPS the tape may be positioned, mounted or dismounted, or EOFs may be written using ZTAPE. NOTE: mounting and dismounting are generally done only by the AIPS program itself. Details of the call sequence to ZTAPE 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

The same routines to write to disk files can be used to talk to tape files although several call arguments have altered meanings for tape files.

9.2.3.1 MINI3/MDIS3 And UVINIT/UVDISK - Double buffered I/O can be done using MINI3/MDIS3 and UVINIT/UVDISK. For these pairs of routines the primary difference between their use on disk and tape is that the physical blocks on the tape are: 1) a single logical record of an image (a row, or the first dimension) if written using MINI3/MDIS3 or 2) the number of logical records (visibilities) requested in a single call (NPIO) to UVINIT. Since these routines know or care little about the internal structure of the data read or written, in practice, any format records can be processed.

9.2.3.2 ZFIO - Single buffered I/O can be done using ZFIO but the input variable used for to block number becomes the byte count for the transfer.

9.2.3.3 VBOUT - The utility routine VBOUT will collect variable length records and block them, unspanned, into IBM format physical blocks up to 4008 bytes long. The tape must be opened with ZOPEN as a non-map file. The principle use of this routine is to write VLA "EXPORT" format tapes. Details of the call sequence as well as other important useage notes are found at the end of this chapter.

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.

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 varies 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.

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, QMS Lasergraphics printer and interactive graphics terminals such as the Tektronix 4012. This section deals with the Tektronics 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 intepret 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 funotion with the converted units, such as display them.

Current AIPS use of graphics is quite primative. In the future we will probably convert to use of the GKS 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 ZOPEN. AIPS logical unit 7 is reserved for this device type, and should be used in the call to ZOPEN. When constructing the device name with ZPHFIL, a device type of 'TK' must be used. A volume number of 1 and zero values for the other auguments should be used to remain consistent with other tasks. On the VAX, each AIPS is assigned a graphics terminal on start up according to a set of logical names. Thus, ZOPEN on the VAX ignores everything in the name exoept TK.

9.3.2 Writing To The Graphics Terminal

Before writing to the graphics terminal, the programmer must set some values in common. Common INCS:CTVC.INC can be initialized by calling routine YTVCIN. Most values in this common are for the TV display, but array MAXXTK 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). In common INCS:CTKS.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 truncate vectors that run off the plot and write the buffer when it fills up.

Characters can be written to a Tectronix terminal by calling routine TKCHAR. TKCHAR allows the programmer to write characters either horizontally or vertically. TKCHAR uses the hardware character generator in the Tektronics, 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 can find one in task PRTPL that can be adapted to a graphics terminal.

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. The 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 routines ICINIT and ICWRIT can be used to set up the image catalog for the graphics terminal. See the chapter on catalogues 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*2 TK, NO, N1, ITKLUN, ITKIND, IERR, TKSIZE, TKPOS,
     *   IPOS, IDRAW, NCHAR, IHORZ, IPLANE, BUFFER(256), VOL, CNO,
     *   CATBLK(256), LINE(40)
      LOGICAL*2 T,F
      REAL*4    DEVNAM(6), BLCX, BLCY, TRCX, TRCY, CENTER, DCX, DCY
          ...
      INCLUDE 'INCS:DHDR.INC'
      INCLUDE 'INCS:DDCH.INC'
      INCLUDE 'INCS:DTVC.INC'
      INCLUDE 'INCS:DTKS.INC'
      INCLUDE 'INCS:CHDR.INC'
      INCLUDE 'INCS:CDCH.INC'
      INCLUDE 'INCS:CTVC.INC'
      INCLUDE 'INCS:CTKS.INC'

      .
      .
      .
C                                          Open the Tektronix device.
      ITKLUN = 7
      CALL ZPHFIL (TK, N1, NO, NO, DEVNAM, IERR)
      IF (IERR.NE.0) GO TO 900
      CALL ZOPEN (ITKLUN, ITKIND, N1, DEVNAM, F, T, T, IERR)
      IF (IERR.NE.0) GO TO 900
```

```
C                                       Set variables in common.
      CALL YTVCIN
      TKSIZE = 20
      TKPOS = 0
C                                       Make screen be 100 by 100
C                                       units.
      SCALEX = MAXXTK(1) / 100.0
      SCALEY = MAXXTK(2) / 100.0
      RXO = 0.0
      RYO = 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
C                                       X and Y and position vector.
      IPOS = 1
C                                       2 is the code for scale X and
C                                       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
      ENCODE (NCHAR, 1000, LINE)
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                                       Calculate image plane.  These
```

```
C                                          values are found in common
C                                          set up in CDCH.INC.
      IPLANE = NGRAY + NGRAPH + NTKDEV
      CALL ICINIT (IPLANE, BUFFER)
C                                          CATBLK, VOL and CNO were
C                                          found when map was opened.
      CATBLK(I2VOL) = VOL
      CATBLK(I2CNO) = CNO
C                                          Set plot type to MISC
      CATBLK(I2PLT) = 1
      CALL ICWRIT (IPLANE, NO, CATBLK, BUFFER, IERR)
C                                          Close graphics terminal.
      CALL ZCLOSE (ITKLUN, ITKIND, IERR)
      .
      .
      .
 1000 FORMAT ('This is a test')
```

9.4 INCLUDES

9.4.1 CTKS.INC

```
C                                                        Include CTKS
      COMMON /TKSPCL/ TKBUFF, SCALEX, SCALEY, RXO, RYO, RXL, RYL,
     *   TKPOS, TKSIZE
C                                                        End CTKS
```

9.4.2 CTVC.INC

```
C                                                         Include CTVC
      COMMON /TVCHAR/ NGRAY, NGRAPH, NIMAGE, MAXXTV, MAXINT, SCXINC,
     *   SCYINC, MXZOOM, NTVHDR, CSIZTV, GRPHIC, ALLONE, MAXXTK,
     *   CSIZTK, TYPSPL, TVALUS, TVXMOD, TVYMOD, TVDUMS, TVZOOM,
     *   TVSCRX, TVSCRY, TVLIMG, TVSPLT, TVSPLM, TVSPLC, TYPMOV,
     *   YBUFF
C                                                         End CTVC
```

9.4.3 DTKS.INC

```
C                                                        Include DTKS
      REAL*4    TKBUFF(20), SCALEX, SCALEY, RXO, RYO, RXL, RYL
      INTEGER*2 TKPOS, TKSIZE
C                                                        End DTKS
```

9.4.4 DTVC.INC

```
C                                                              Include DTVC
      INTEGER*2 NGRAY, NGRAPH, NIMAGE, MAXXTV(2), MAXINT, SCXINC,
     *    SCYINC, MXZOOM, NTVHDR, CSIZTV(2), GRPHIC, ALLONE, MAXXTK(2),
     *    CSIZTK(2), TYPSPL, TVALUS, TVXMOD, TVYMOD, TVDUMS(7),
     *    TVZOOM(3), TVSCRX(16), TVSCRY(16), TVLIMG(4), TVSPLT(2),
     *    TVSPLM, TVSPLC, TYPMOV(16), YBUFF(168)
C                                                              End DTVC
```

9.5 ROUTINES

9.5.1 ICINIT - Initializes image catalog for plane IPLANE

```
   ICINIT (IPLANE, BUFF)
Input:
  IPLANE      I*2      Image plane to initialize
Output:
  BUFF(256)   I*2      Working buffer
```

9.5.2 ICWRIT - writes image catalog block from ICTBL into image catalog.

```
   ICWRIT (IPLANE, IMAWIN, ICTBL, BUFF, IERR)
Inputs:
  IPLANE      I*2       image plane involved
  IMAWIN(4)   I*2       Corners of image on screen
  ICTBL       I*2(256)  Image catalog block
Outputs:
  BUFF        I*2(256)  working buffer
  IERR        I*2       error code: 0 => ok
                        1 => no room in catalog
                        2 => IO problems
```

9.5.3 MDIS3 - reads or writes image data to/from disks and other devices. MDIS3 and MINI3 are pseudo I*4 versions of yet to be written replacements MDISK and MINIT which will use true I*4.

```
      MDIS3 (OP, LUN, FIND, BUFF, BIND, IERR)
   Inputs:
     OP    I*4  Op code char string 'WRIT', 'READ','FINI'
     LUN   I*2  logical unit number
     FIND  I*2  Pointer to FTAB returned by ZOPEN
   Input and/or output:
     BUFF  ??   Buffer holding data, you better know  specification
   Output:
     BIND  I*2  Pointer to position in buffer of first pixel in window
                in the present line
     IERR  I*2  Error return: 0 => ok
                              1 -> file not open
                              2 => input error
                              3 => I/O error
                              4 => end of file
                              5 => beginning of medium
                              6 => end of medium
```

MDIS3 sets array index to the start of the next line wanted.
NOTE: the line sequence is set by the WIN parameter in MINI3, if the vaules of WIN(2) and Win(4) are switched then the file will be accessed backwards.
A call with OP = 'FINI' flushes the buffer when writing.
MINI3 MUST be called before MDIS3.

9.5.4 MINI3 - initializes the I/O tables for MDIS3.

```
      MINI3 (OP, LUN, IND, LX, LY, WIN, BUFF, BFSZ, BYTPIX,
     *   BLKOF, IERR)
   Inputs:
     OP   R*4  Operation code character string: 'READ','WRIT'
     LUN  I*2  logical unit number
     IND  I*2  pointer to FTAB, returned by ZOPEN when file is opened
     LX   I*2  Number of pixels per line in X-direction for whole
               plane
     LY   I*2  Number of lines in whole plane.
     WIN  I*2(4)  Xmin,Ymin,Xmax,Ymax defining desired subrectangle in
               the plane. A subimage may NOT be specified for 'WRIT'.
     BFSZ I*2  Size of total available buffer in bytes, should be even
                  Special case: BUFSZ-32767 is treated as though
                  BUFSZ-32768 to allow double buffering of 16Kbyte
                  records.
     BYTPIX I*2 Number of bytes per pixel in stored map
     BLKOF I*2(2) Pseudo I*4 block number, 1 relative, of first map
                    pixel in the desired plane. Use COMOF3 + ZMATH4
                    to set.
   Outputs:
```

```
     IERR  I*2 Error return: 0 => ok
                             1 -> file not open
                             2 => input error
                             7 => Buffer too small
                             3 => I/O error on initialize
                             4 => end of file
                             5 => beginning of medium
                             6 => end of medium
```

MINI3 sets up special section of FTAB for quick return, double buffered I/O. N.B. This routine is designed to read/write images one plane at a time. One can run the planes together iff the rows are not blocked: i.e. iff NBPS / (LX * BYTPIX) < 2.
Usage notes: For map I/O the first 16 words in each FTAB entry contain a user table to handle double buffer I/O, the rest contain system-dependent I/O tables. A "major line" is 1 row or 1 sector if more than 1 line fits in a sector. FTAB user table entries, with offsets from the FIND pointer are:

```
   FTAB + 0 => LUN using this entry
          1 => No. of major lines transfered per I/O op
          2 => No. of major times a buffer has been accessed
          3 => No. of major lines remaining on disk
          4 => Output index for first pixel in window
          5 => No. pixels to increment for next major line
          6 => Which buffer to use for I/O; -1 => single buffer
          7 => Block offset in file for next operation (lsb  I*4)
          8 => msb of pseudo I*4 block offset
          9 => Block increment in file for each operation
         10 => No. of bytes transferred
         11 => I/O op code 1=> read, 2 => write.
         12 => BYTPIX
         13 => # rows / major line (>= 1)
         14 => # times this major line has been accessed
         15 => # pixels to increment for next row (= LX)
```

9.5.5 TEKFLS - writes the output buffer TKBUFF to the TEKTRONIX 4012.

```
    TEKFLS (FIND,IERR)
Inputs:
  FIND     I*2 FTAB position assigned to TEK 4012.
Outputs:
  IERR     I*2 error flag. 0-ok,  .GT. 1-write error from ZFIO
```

9.5.6 TEKVEC - puts control characters, and X and Y coordinates into the TEKTRONIX output buffer.

```
    TEKVEC (XX, YY, IN, FIND, IERR)
Inputs:
  XX     I*2 X coordinate value.
  YY     I*2 Y coordinate value.
  IN     I*2 control value:
             1 - Scale XX and YY and precede coordinates
                 by 'write dark vector' control character
             2 - Scale XX and YY, put in buffer; will write
                 bright vector.
             3 - XX and YY are not scaled, 'write dark
                 vector' control character is put into
                 the buffer.
             4 - no scale, write bright vector.
  FIND     I*2 FTAB position of TEKTRONIX device.
Output:
  IERR     I*2 error code, 0-ok, 1-write error.
Common variables modified:
  TKBUFF
  TKPOS
  RXL, RYL
```

9.5.7 TKCHAR - writes characters to a TEKTRONIX 4012.

```
    TKCHAR (INCHAR, IANGL, DCX, DCY, TEXT, ITFIND, IERR)
Inputs:
  INCHAR   I*2 number of characters.
  IANGL    I*2 0-horizontal, other - vertical.
  DCX      R*4 X distance in characters from current position.
  DCY      R*4 Y distance in characters from current position.
  TEXT     R*4(??) packed characters.
  ITFIND   R*4 FTAB index of open TEK.
Outputs:
  IERR     I*2 error indicator. 0 - ok.
```

9.5.8 TKCLR - will clear the screen for a Tektronix 401n.

```
    TKCLR (DEVFND, IERR)
Inputs:
  DEVFND   I*2   FTAB index of an open device.
Output:
  IERR     I*2   Error code from the last I/O routine. 0-ok.
```

9.5.9 TKCURS - activates the cursor on the TEKTRONIX 4012 and waits for a response from the 4012 keyboard. After the response the cursor will disappear and TEKCUR will return the coordinate positions. The TEKTRONIX must have opened (by ZOPEN) before this routine is called.

```
    TKCURS (IFIND, IOBLK, IX, IY, IERR)
Inputs:
  IFIND   I*2 index into FTAB for open TEKTRONIX device.
  IOBLK   I*2(256) I/O block for TEKTRONIX device.
Outputs:
  IX      I*2 x cursor position.
  IY      I*2 y cursor position.
  IERR    I*2 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.

9.5.10 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*2   1 => dark vector, 2 => bright vector
  X    I*2   X coordinate value.
  Y    I*2   Y coordinate value.
  FIND I*2   FTAB position of TEKTRONIX device.
Outputs:
  IERR I*2   error code, 0-ok, 1-write error.
Common variables modified:
  TKBUFF
  TKPOS
```

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

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

WRIT arranges data in a buffer until it is full. Then as many full blocks as possible are written to the disk with the remainder left for the next disk write. For tape I/O data is always written with the block size specified to UVINIT; one I/O operation per call. For disk writes, left-over data is transferred to the beginning of

buffer 1 if that is the next buffer to be filled. Value of NIO in the call is the number of vis. rec. to be added to the buffer and may be fewer than the number specified to UVINIT. On return NIO is the maximum number which may be sent next time. On return BIND is the pointer in BUFFER to begin filling new data.

FLSH writes integral numbers of blocks and moves any data left over to the beginning of buffer half 1. One exception to this is when NIO => -NIO or 0, in which case the entire remaining data in the buffer is written. After the call BIND is the pointer in BUFFER for new data. The principal difference between FLSH and WRIT is that FLSH always forces an I/O transfer. This may cause trouble if a transfer of less than 1 block is requested. A call with a nonpositive value of NIO should be the last call and corresponds to a call to MDIS3 with opcode 'FINI'.

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

UVDISK (OP, LUN, FIND, BUFFER, NIO, BIND, IERR)

Inputs:

OP	R*4	Opcode 'READ','WRIT','FLSH' are legal
LUN	I*2	Logical unit number
FIND	I*2	FTAB pointer returned by ZOPEN
BUFFFER()	I*2	Buffer for I/O
NIO	I*2	For writes, the number of visibilites added to the buffer; not used for reads.

Output:

NIO	I*2	For reads, the number of visibilities ready in the buffer; For writes, the maximum number which can be added to the buffer. If zero for read or write then the file is completely read or written.
BIND	I*2	The pointer in the buffer to the first word of the next record for reads, or the first word of the next record to be copied into the buffer for writes.
IERR	I*2	Return error code. 0 => OK 1 => file not open in FTAB 2 => input error 3 => I/O error 4 => end of file 7 => attempt to write more vis than specified to UVINIT or will fit in buffer.

9.5.12 UVINIT - sets up bookkeeping for the UV data I/O routine UVDISK. I/O for these routines is double buffered (if possible) quick return I/O. UVDISK will run much more efficiently if on disk LREC*NPIO*BP is an integral number of blocks. Otherwise partial writes or oversize reads will have to be done. Minimum disk I/O is one block. The buffer size should include an extra NBPS bytes for each buffer for non tape read if NPIO records does not correspond to

an integral number of disk sectors (NBPS bytes). 2*NBPS extra bytes required for each buffer for write.

```
      UVINIT (OP, LUN, FIND, NVIS, VISOFF, LREC, NPIO,
     *   BUFSZ, BUFFER, BO, BP, BIND, IERR)
```

Inputs:

OP	R*4	OP code, 'READ' or 'WRIT' for desired operation.
LUN	I*2	Logical unit number of file.
FIND	I*2	FTAB pointer for file returned by ZOPEN.
NVIS	I*4	Total number of visibilities to read. NVIS+VISOFF must be no greater than the total number in the file.
VISOFF	I*4	Offset in vis. rec. of first vis. rec. from BO.
LREC	I*2	Number of values in a visibility record.
NPIO	I*2	Number of visibilities per call to UVDISK. Determines block size for tape I/O
BUFSZ	I*2	Size in bytes of the buffer. If 32767 given, 32768 is assumed.
BUFFER( )	I*2	Buffer
BO	I*4	Block offset to begin transfer from (1-relative)
BP	I*2	Bytes per value in the vis. record.

Output:

NPIO	I*2	For WRITE, the max. number of visibilities which can be accepted.
BIND	I*2	Pointer in BUFFER for WRITE operations.
IERR	I*2	Return error code: 0 => OK 1 => file not open in FTAB 2 => invalid input parameter. 3 => I/O error 4 => End of file. 7 => buffer too small

Note: VISOFF and BO are additive.

UVINIT sets and UVDISK uses values in the FTAB:

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

9.5.13 VBOUT - VBOUT writes variable blocked records of I*2 data to tape. Maximum block size on the tape is 4008 bytes. Tape must be opened (non-map) before first call. For overlaid programs COMMON /VBCOM/ should be kept in a segment which is core-resident from the first call 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 two characters per integer as local integers: ie call ZCLC8 followed by ZI16IL on such data before calling VBOUT.

```
      VBOUT (N, IDATA, LUN, NUM, IERR)

    Inputs:
       N      I*2 Number of I*2 words in array IDATA
                  If N = 0 the buffer is flushed.
       IDATA  I*2 Array containing data to be written.
       LUN    I*2 LUN of tape to be written on.
       NUM    I*2 The record number to be written, must be 1 on
                  the first and only the first record in a file.
    Output:
       IERR   I*2  Return error code 0 => OK
                                 1 => LSERCH error (tape not open)
                                 2 => ZFIO error.
```

9.5.14 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.

9.5.15 ZOPEN - opens logical files, performing full open on disk files and sets up an FTAB entry for double buffering.

```
      ZOPEN (LUN, IND, IVOL, PNAME, MAP, EXCL, WAIT, IERR)

   Inputs:
      LUN    I*2       Logical unit number.
      IVOL   I*2       Disk volume containing file, 1,2,3,...
      PNAME  R*4(6)    24-character physical file name,left justified,
                       packed, and padded with blanks.
      MAP    L*2       is this a map file ?
      EXCL   L*2       desire exclusive use?
      WAIT   L*2       I will wait?
   Output:
      IND    I*2       Index into FTAB for the file control block.
      IERR   I*2       Error return code:
                           0 = no error
                           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

9.5.16 ZPHFIL - constructs a physical file name in PNAM from ITYPE, IVOL, NSEQ, and IVER. New version designed either for public data files or user specific files. This routine contains the logical assignment list for Graphics devices. Numerical values are encoded as hexidecimal numbers.

EXAMPLE: If ITYPE='MA', IVOL=8, NSEQ=801, IVER=153, NLUSER=768 then
PNAME='DA08:MA832199;1' for public data or
PNAME='DA08:MA832199.300;1' for private data
ITYPE = 'MT' leads to special name for tapes
ITYPE = 'TK' leads to special name for TEK4012 plotter CRT
ITYPE = 'TV' leads to special name for TV device
ITYPE = 'ME' leads to special logical for POPS memory files

ZPHFIL (ITYPE, IVOL, NSEQ, IVER, PNAM, IERR)

Inputs:

ITYPE	I*2	Two characters denoting type of file. For example, 'MA' for map file.
IVOL	I*2	Number of the disk volume to be used.
NSEQ	I*2	User supplied sequence number. 000-999.
IVER	I*2	User suppplied version number. 00-255.

Outputs:

PNAM	R*4(6)	>= 24-byte field to receive the physical file name, left justified (packed) and padded with blanks.
IERR	I*2	Error return code. 0 = good return. 1 = error.

9.5.17 ZTAPE - Performs standard tape manipulating functions.

ZTAPE (OP, LUN, FIND, COUNT, IERR)

Inputs:

OP R*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. Works for VMS 3.0 & later.
'MONT' = mount tape. Works for VMS 3.0 and later.
'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*2  logical unit number
  FIND I*2  FTAB pointer.  Drive number for MOUNT/DISMOUNT.
  COUNT I*2 Number of records or file marks to skip. On MOUNT
            this value is the density.
Outputs:
  IERR  I*2  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
```

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 gray 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. BLANK) 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. Gray scale images provide a more 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 sells only a Model 75. Our initial choice undoubtedly colors 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 and 70/F. 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, and for IIS Model 75, developed by IIS.

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 n bits deep which hold the gray-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.

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 gray-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). 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, n-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. Arrays inside AIPS are likely to be dimensioned for 8-bit image planes and a 10-bit OFM. (These assumptions probably should be generalized in time.)

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.

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 integer powers of two without affect on the images stored in the image memories. The highest 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, appropriate 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 new nonstandard tasks for histogram equilization 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.

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 and CTVC.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. 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 gray-scale image plane (affects the image catalogue mostly).
MAXXTV(2)	The number of pixels in the X and Y directions.
MAXINT	The highest gray-scale intensity = 2 ** n - 1.
SCXINC	The minimum increment in scroll in the X direction.
SCYINC	The minimum increment in scroll in the Y direction.
MXZOOM	The highest power of two for zooming.
NTVHDR	The number of integer words in the TV I/O header (probably no longer used).
CSIZTV(2)	The size of characters in pixels in the X, Y directions.
GRPHIC	The bit pattern representing the set of graphics overlay memories (normally -32768).
ALLONE	The bit pattern representing all bits on (-1).
MAXXTK(2)	The number of pixels in the X, Y directions on the TEK graphics device.
CSIZTK(2)	The size of characters on the TEK graphics device 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.
TVDUMS(7)	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 gray 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: 2 * (magnification power of two) + 8 * (number frames remaining). Add 1 if this is the first plane of the movie.
YBUFF(168)	Machine dependent parameters.

There is a second TV include which controls I/O, but is little used elsewhere. It is obtained by including DTVD.INC and CTVD.INC 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. The current implementations are

```
Z...OP  :
     ZM70OP (LUN, IND, IERR)
     ZDEAOP (LUN, IND, 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.
```

Z...CL :
ZM70CL (LUN, IND, IERR),
ZDEACL (LUN, IND, IERR),
Performs a simple close (deassign) via a call to ZCLOSE and clears the FTAB entry. The DeAnza version calls ZDEAXF ('DET ',...) to perform a deallocation before calling ZCLOSE.

Z...MC :
ZM70MC (FTAB(channel)) - Vax version
Performs a "rewind" QIO operation causing the IIS to reset its I/O status.
ZM70MC - Modcomp version
Performs a "home" I/O operation causing the IIS to reset its I/O status.
ZDEAMC
Null subroutine.

Z...XF :
ZM70XF (OPER, NBYTES, HEADER, BUFFER, 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. Issues a master clear on error.
ZDEAXF (OPER, BUFFER, NBYTES, EP1, EP2, 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).

10.2.3 The Y Routines

The Y routines may be divided into three groups which we call levels 0 through 2. Level 0 routines do not perform I/O to the TV device. Instead, they prepare data to be fed to lower level Y routines and/or handle common parameters and various conversions. It has been found that this level of Y routine often needs little alteration from one model of TV to the next. Level 1 routines do call Z...XF to perform I/O to the TV device. They may be called by both Y and non-Y routines and hence must be implemented for all TV devices. Level 2 routines also perform I/O in general, but are only called by Y routines. Hence, these do not have to be implemented for all TV devices. The reader should note that the division of Y routines into these three levels is not quite so clear as the above description would indicate. For one, some level 2 routines may have to graduate to level 1 as new application code is developed. For another, some of the level 0 routines are actually TV independent as coded for the IIS Models 70 and 75. They are called Y routines simply to allow more efficient, level 0 or 1 implementations for other TV devices.

On normal AIPS systems, the Y subroutines are stored in subdirectories separated by type of TV device. On our Vax, the subdirectories are [AIPS.reldate.APL.YSUB.xxx] with logical names APLxxx: where xxx is IIS for IIS Model 70, M75 for IIS Model 75, and DEA for DeAnza and where reldate is the date of the current AIPS release. The compile procedures select the value of xxx appropriate to the local TV device and write the object code into the link editor library in the [AIPS.reldate.APL] area. This library is then used for link editing all programs and tasks. The careful reader will note that this method does not allow for more than one kind of TV device on a given host computer. To date, we have been able to get away with this deficiency. In the future, we may have to improve the AIPS start-up procedure and the task activation subroutine (ZACTV8) so that the number of the assigned TV device is used to determine from which library the executable modules are taken.

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.

10.2.3.1 Level 0

- YCHRW writes characters into an image or graphics plane. The M70 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.

- YCNECT writes a line segment in an image or graphics plane at a specified intensity. The M70 version is TV independent.

- 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. M70 and DeAnza versions are identical.

- 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 IIS Model 70/E is tricky, the Model 70/F and DeAnza are easier.

- 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.

- YLNCLR computes a piecewise linear OFM with gamma correction. Called a Y routine solely because of the use of a 10-bit OFM.

- YSLECT enables/disables gray and graphics channels setting the proper values into TVLIMG.

- YTVCIN provides initial values for the TV characteristics commons.

- YZERO clears a gray or graphics memory by the fastest possible method.

- YTVCLS close the TV device. Actually is just an interface to the appropriate Z...CL subroutine.

- YTVMC reset the TV I/O status. Actually is just an interface to the appropriate Z...MC subroutine.

- YTVOPN Open the TV device. Actually is just an interface to the appropriate Z...OP subroutine.

10.2.3.2 Level 1

- YCRCTL reads/writes the cursor/trackball control register including position, enable/disable on each axis, blink control.

- YIMGIO reads/writes a line of image data from/to a gary-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.

- YINIT initializes all subunits of the TV, clears the TV memories, resets the image catalog, and resets status parameters in common.

- YLUT reads/writes the full channel-level lookup table for one or more image channels and colors.

- YOFM reads/writes the full OFM lookup table for one or more colors.

- YSCROL writes the soroll control registers for one or more channels.

- 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.

- YZOOMC writes the zoom control registers giving magnification and zoom center.

10.2.3.3 Level 2

10.2.3.3.1 IIS Models 70 And 75

- 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.

- 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.

- 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.

- YGGRAM reads/writes the lookup table used for graphics planes.

- 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.

- YGYHDR prepares a basic I/O control header for writing/reading image data to/from the IIS.

- 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.

- YMAGIC (Model 75 only) initializes graphics, zoom, and scroll subunits (called by YINIT only).

- YMKHDR prepares a basic I/O control header for the IIS.

- YMNMAX reads the min and max output from the sum of all enabled gray-scale planes for each color.

- 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.

- 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.

- 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.

10.2.3.3.2 DeAnza

- YGGRAM reads/writes the lookup table used for the graphics planes.
- YLOWON finds lowest channel number in a channel mask.
- YMKCUR creates and loads the cursor pattern memory with a specified shape. Only the AIPS plus sign is implemented.
- YTCOMP performs logical tests on parameter values. It is used to minimize I/O to the DeAnza control registers.
- YDEA.INC Include file giving parameter definitions to specify positions in YBUFF which correspond to the various registers in a DeAnza TV device.

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 ICINIT) 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, resp.), and CURBLINK

(via YCURSE).

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 gray planes is done to zero the memory (via YZERO) and clear the image catalogue (via ICINIT).

```
C                                              Open TV device
      CALL TVOPEN (CATBLK, JERR)
      IF (JERR.EQ.0) GO TO 50
         POTERR = 101
         GO TO 980
....
 200  ICHAN = ABS(TVCHAN) + EPS
C                                              convert to channel bit mask
      CALL DECBIT (NGRAY, ICHAN, ICHAN, ITEMP)
C                                              clear movie parameters
      CALL MOVIST (ONCODE(2), ICHAN, NO, NO, NO, IERR)
      DO 210 IP = 1,NGRAY
C                                              is plane requested
         IF (IAND(ICHAN,N2**(IP-1)).EQ.0) GO TO 210

C                                              clear image catalogue
            CALL ICINIT (IP, INBUF)
C                                              clear TV memory
            CALL YZERO (IP, JERR)
            IF (JERR.NE.0) GO TO 975
 210     CONTINUE
      GO TO 900
......
C                                              normal TV close
 900  CALL TVCLOS (CATBLK, JERR)
      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 the 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 catalogue parameters in the header. It sets the window parameters (via TVWIND), selects a single gray scale memory plane (via DECBIT), and clears the movie

parameters (via MOVIST). Finally, it finishes up the image catalogue parameters, puts the header in the image catalogue, 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).

```
      INTEGER*2 NO, N1, N6, N7, N12, N6176, MA
      INCLUDE 'INCS:DHDR.INC'
      INCLUDE 'INCS:CHDR.INC'
      DATA MA /'MA'/, NO, N1, N6, N7, N12, N6176 /0,1,6,7,12,6176/
C                                          open TV
      CALL TVOPEN (INBUF, IERR)
      IF (IERR.NE.0) GO TO 980
C                                          Map open junk: TVLOD, TVROAM
      IF (BRANCH.GT.2) GO TO 20
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
                                                    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 CHCOPY (N12, N1, INNAM, N1, SNAME)
      CALL CHCOPY (N6, N1, INCLS, N1, SCLAS)
      CALL RCOPY (N7, TVBLCO, LBLC)
      CALL RCOPY (N7, TVTRCO, LTRC)
      INC(1) = TVXINC + EPS
      INC(2) = TVYINC + EPS
      IMA = MA

C                                          open map file
      CALL MAPOPN (READ, IVOL, SNAME, SCLAS, SEQNO, IMA, USID,
     *    DLUN, DIND, CNO, CATBLK, INBUF, IERR)
      POTERR = 33
      IF (IERR.GT.1) GO TO 975
C                                          CATBLK, CT4, CT8 equivalenced
C                                          Image cat: fill in some
C                                          set image scaling too
      CALL RNGSET (PXRANG, CT4(K4DMX), CT4(K4DMN), CT8(K8BSC),
     *    CT8(K8BZE), CT4(I4RAN))
```

```
          CATBLK(I2VOL) = IVOL
          CATBLK(I2CNO) = CNO
          CALL CHCOPY (N2, N1, FUNTYP, N1, CATBLK(I2TRA))
          ITVC(1) = TVCORN(1) + EPS
          ITVC(2) = TVCORN(2) + EPS
          POTERR = 49
C                                          TVLOD
C                                          load one image plane
C                                          set windows
 100  TYPE = -1
      CALL TVWIND (TYPE, INC, LBLC, LTRC, ICHAN, ITVC, IWIN, IERR)
      IF (IERR.NE.0) GO TO 970
C                                          convert channel number
 110  CALL DECBIT (NGRAY, ICHAN, ICHAN, I)
      ICHAN = I
      CALL DECBIT (NGRAY, ICHAN, ICHAN, I)
C                                          clear movie parameters
      CALL MOVIST (OFF, ICHAN, NO, NO, NO, IERR)
C                                          do the TV load, img catlg
      CALL TVLOAD (DLUN, DIND, I, INC, ITVC, IWIN, N6176, 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, and IMERASE 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 catalogue header and writes that to the catalogue via ICWRIT. 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.)

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, VM, MX, Et Al.

Iterative map analysis programs can make good use of the TV display. The user may, for example, request that the CLEAN task (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, VM, APGS, REGLR) 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 VFIX). 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:  TVPASS  I*2   code: 0 => clear screen, else don't
C                                0,1 => don't question the user about
C                                quitting
C   Output:  TVPASS  I*2   code: 32700 => user wants to quit cleaning
C-----------------------------------------------------------------------
```

```
      INTEGER*2 TVPASS, JROW(1), WIN(4), MY, FIND, BIND, IERR,
     *   ICH, CATBLK(256), S2H(256), IQ, IB, IBLANK, I
      INTEGER*4 MX, ZERO, ONE, TWO, THREE, FOUR
      INTEGER*2 IWIN(4), IY, MAXO, MINO
      INTEGER*2 NO, N1, N2, N3, N4, N5, N6, N256
      REAL*4    XN(4), XBUFF(1), REED, S4H(128), TD, RPOS(2), ON, OFF
      REAL*4    WRIT, XFLUX, PREFIX(2), TVLMAX, TVLMIN, ARG,
     *   AMIN1, AMAX1
      LOGICAL*2 MAP, EXCL, WAIT, LERR, F
      REAL*8    S8H(64)
      INCLUDE 'INCS:DCLN.INC'
      INCLUDE 'INCS:DFIL.INC'
      INCLUDE 'INCS:DTVC.INC'
      INCLUDE 'INCS:DMSG.INC'
      INCLUDE 'INCS:DHDR.INC'
      INCLUDE 'INCS:DTVD.INC'
      INCLUDE 'INCS:CMSG.INC'
      INCLUDE 'INCS:CCLN.INC'
      INCLUDE 'INCS:CFIL.INC'
      INCLUDE 'INCS:CTVC.INC'
      INCLUDE 'INCS:CHDR.INC'
      INCLUDE 'INCS:CTVD.INC'
      COMMON /MAPHDR/ CATBLK
      EQUIVALENCE (JROW(1), BUFF2(1)),    (BUFF1(1), XBUFF(1))
      EQUIVALENCE (S2H, S4H, S8H, BUFF1(513))
      DATA MAP, EXCL, WAIT / .TRUE., 2*.TRUE./
      DATA WRIT, REED, ON, OFF /'WRIT','READ','ONNN',
     *   'OFFF'/
      DATA NO, N1, N2, N3, N4, N5, N6, N256 /0,1,2,3,4,5,6,256/
      DATA F, IBLANK /.FALSE., '  '/
      DATA ZERO, ONE, TWO, THREE, FOUR /0, 1, 2, 3, 4/
C-------------------------------------------------------------------
      ICH = 1
      CALL TVOPEN (BUFF1, IERR)
      IF (IERR.EQ.0) GO TO 10
         ENCODE (80,1000,MSGTXT) IERR
         CALL MSGWRT (N6)
         GO TO 999
 10   IF (TVPASS.NE.0) GO TO 20
         CALL YZERO (ICH, IERR)
         IF (IERR.EQ.0) GO TO 15
            ENCODE (80,1010,MSGTXT) IERR
            CALL MSGWRT (N6)
            GO TO 998
 15      CALL ICINIT (ICH, XBUFF)
 20   IF (TVFMAX.GT.TVFMIN) GO TO 25
         TVFMAX = TVREMX
         TVFMIN = TVREMN
 25   IF (TVREMX.GT.TVFMAX) TVFMAX = TVREMX
      IF (TVREMN.LT.TVFMIN) TVFMIN = TVREMN
      TVLMAX = TVFMAX - TVFMIN
      IF (0.1*TVLMAX.LE.TVREMX-TVREMN) GO TO 30
         ARG = 0.1 * TVFMIN
         TVFMIN = AMIN1 (ARG, TVREMN)
```

```
         ARG = TVFMIN + 0.1 * TVLMAX
         TVFMAX = AMAX1 (ARG, TVREMX)
         TVLMAX = TVFMAX - TVFMIN
 30    XN(1) = TVFMIN
       XN(2) = TVFMAX
       XN(3) = (MAXINT - 0.02) / TVLMAX
       XN(4) = 0.51 - TVFMIN * XN(3)
       CALL QPUT (XN, ZERO, FOUR, TWO)
C                                          Write scaling factor
       XFLUX = TVLMAX
       CALL METSCA (XFLUX, PREFIX, LERR)
       TVLMIN = TVFMIN * XFLUX / TVLMAX
       TVLMAX = TVFMAX * XFLUX / TVLMAX
       ENCODE (80,1020,MSGTXT) TVLMIN, TVLMAX, PREFIX
       CALL MSGWRT (N1)
       WIN(1) = (WINM(3,1) + WINM(1,1)) / 2 - MAXXTV(1) / 2 + 1
       WIN(1) = MAXO (N1, WIN(1))
       WIN(2) = (WINM(4,1) + WINM(2,1)) / 2 - MAXXTV(2) / 2 + 1
       WIN(2) = MAXO (N1, WIN(2))
       WIN(3) = (WINM(3,1) + WINM(1,1)) / 2 + MAXXTV(1) / 2
       WIN(3) = MINO (NX, WIN(3))
       WIN(4) = (WINM(3,1) + WINM(1,1)) / 2 + MAXXTV(2) / 2
       WIN(4) = MINO (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 MINI3 (REED, LUNRES, FIND, NX, NY, WIN, XBUFF, BUFSZ1,
      *    BPRES, BORES, IERR)
       MX = WIN(3) - WIN(1) + 1
       MY =  WIN(4) - WIN(2)+ 1
C                                          loop, passing map to IIS.
       DO 100 I = 1,MY
          IY = I + IWIN(2) - 1
          CALL MDIS3 (REED, LUNRES, FIND, XBUFF, BIND, IERR)
          IF (IERR.NE.0) GO TO 110
          CALL QPUT (XBUFF(BIND), FOUR, MX, TWO)
          CALL QWD
          CALL QVCLIP (FOUR, ONE, ZERO, ONE, FOUR, ONE, MX)
          CALL QVSMSA (FOUR, ONE, TWO, THREE, FOUR, ONE, MX)
          CALL QVFIX (FOUR, ONE, FOUR, ONE, MX)
          CALL QWR
          CALL QGET (JROW, FOUR, MX, ONE)
          CALL QWD
C                                          Send row to IIS.
```

```
         CALL YIMGIO (WRIT, ICH, IWIN, IY, NO, MX, JROW, IERR)
         IF (IERR.NE.0) GO TO 110
 100     CONTINUE
 110  CALL ZCLOSE (LUNRES, FIND, IERR)
C                                          Release the AP
      CALL QRLSE
C                                          Image catalog
      CALL COPY (N256, CATBLK, S2H)
      S2H(I2VOL) = 0
      S2H(I2CNO) = 0
      CALL FILL (N5, N1, S2H(I2DEP))
      CALL COPY (N4, IWIN, S2H(I2COR))
      CALL COPY (N4, WIN, S2H(I2WIN))
      S2H(I2TRA) = IBLANK
      S8H(K8BSC) = 1.0D0
      S8H(K8BZE) = 0.0D0
      S4H(I4RAN) = TVFMIN
      S4H(I4RAN+1) = TVFMAX
      S4H(K4DMN) = TVREMN
      S4H(K4DMX) = TVREMX
      CALL ICWRIT (ICH, IWIN, S2H, XBUFF, IERR)
      IF (IERR.EQ.0) GO TO 120
         ENCODE (80,1110,MSGTXT)
         CALL MSGWRT (N6)
C                                          Ask user to quit?
 120  IF (TVPASS.LT.2) GO TO 998
         ENCODE (80,1120,MSGTXT)
         CALL MSGWRT (N1)
         ENCODE (80,1121,MSGTXT)
         CALL MSGWRT (N1)
         RPOS(1) = MAXXTV(1)/2.0
         RPOS(2) = MAXXTV(2)/2.0
         TD = 0.2
         CALL YCURSE (ON, F, F, RPOS, IQ, IB, IERR)
         IF (IERR.NE.0) GO TO 998
         DO 130 I = 1,75
            CALL ZDELAY (TD, IERR)
            CALL YCURSE (REED, 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     ENCODE (80,1135,MSGTXT)
         CALL MSGWRT (N1)
         GO TO 150
C                                          Wants to quit
 140     TVPASS = 32700
         ENCODE (80,1140,MSGTXT)
         CALL MSGWRT (N3)
C                                          Off cursor
 150     CALL YCURSE (OFF, F, F, RPOS, IQ, IB, IERR)
 998  CALL TVCLOS (BUFF1, IERR)
C
 999  RETURN
```

```
C--------------------------------------------------------------------
 1000 FORMAT ('CANT OPEN TV IER=',I6)
 1010 FORMAT ('IMCLEAR ERROR =',I6)
 1020 FORMAT ('TVDISP: DISPLAY RANGE =',2F8.3,1X,A4,A1,'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 catalogue as needed.

10.3.6 Transfer Function Modification, Zooming

Subroutine AU6A carries out the verbs OFFTRAN, TVTRAN, TVLUT, and TVMLUT which perform modifications on the black and white (or single color) LUTs of the specified gray-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 (TVSCROLL), 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 gray-scale channel is used to set the intensity and another to set the hue. This algorithm requires the TV to have both LUTs for each channel and an OFM for the sum of the enabled channels. A log function is put in the LUTs and an exponential in the OFM which carries out the required multiplication of the two signals. Subroutines HIENH and HILUT actually carry out 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 task BLANK. It is a deliberately limited interactive routine designed to provide easy to use enhancement in black and white (via IENHNS) or pseudocolor (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 TVSCROLL 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.

```
C                                                  open TV device
      CALL TVOPEN (BUFFER, IERR)
C                                                  get start time
      CALL ZTIME (ITW)
      IF (IERR.EQ.0) GO TO 10
         POTERR = 101
         GO TO 980
......
C                                                  TVSCROL
C                                                  user instructions
 500  ENCODE (80,1500,MSGTXT)
      CALL MSGWRT (N1)
      ENCODE (80,1505,MSGTXT)
      CALL MSGWRT (N1)
C                                                  find channel(s) to scroll
C                                                  scroll graphics too ?
      IC = ABS(TVCHAN) + EPS
      CALL DECBIT (NGRAY, IC, IC, J)
      IF (ABS(GRCHAN).GT.EPS) IC = IOR (IC, GRPHIC)
      IF (IC.NE.0) GO TO 505
         IC = MOD (TVLIMG(1), N2**NGRAY)
```

```
         IF (IC.NE.TVLIMG(1)) IC = IOR (IC, GRPHIC)
 505   IX = 0
       IY = 0
       RPOS(1) = MAXXTV(1)/2
       RPOS(2) = MAXXTV(2)/2
C                                              turn on cursor
       CALL YCURSE (ON, 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) - MAXXTV(1)/2
             IY = RPOS(2) - MAXXTV(2)/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 (OFF, 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 ICREAD 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.EQ.0) GO TO 50
         POTERR = 101
         GO TO 980
.......
C                                              IMPOS
C                                              read cursor to get position
 600   CALL TVWHER (IQUAD, RPOS, IBUT, JERR)
       IF (JERR.NE.0) GO TO 975
```

```
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
         IF (IAND (TVLIMG(IQUAD), N2**(IP - N1)).EQ.0) GO TO 630
C                                          get img cat block
         CALL ICREAD (IP, IX, IY, CATBLK, IERR)
C                                          loop if x,y not in image
         IF (IERR.EQ.N1) GO TO 630
         IF (IERR.EQ.0)  GO TO 650
         GO TO 975
 630     CONTINUE
C                                          x,y not in on image
      ENCODE (80,1630,MSGTXT) IX, IY
      CALL MSGWRT (N6)
      GO TO 900
C                                          image -> map positions
 650  CALL IMA2MP (RPOS, RPOS)
      ENCODE (80,1650,MSGTXT) RPOS
      CALL MSGWRT (N5)
C                                          map -> sky positions
 660  CONTINUE
         CALL MP2SKY (RPOS, SKYPOS)
C                                          3rd axis pairs w 1st or 2nd
         IF ((AXTYP.EQ.2) .OR. (AXTYP.EQ.3)) CALL AXSTRN (CTYP(1,3),
     *      SKYPOS(3), KLOCA, NCHLAB(1), SAXLAB(1,1))
C                                          Primary axes
C                                          Tell user results via MSGWRT.
         ENCODE (80,1660,MSGTXT)
         ICH = 8
         DO 665 I = 1,2
            CALL AXSTRN (CTYP(1,I), SKYPOS(I), I-N1, ILEN, RSTR)
            CALL CHPACK (ILEN, RSTR, ICH, MSGTXT)
            ICH = ICH + ILEN
            CALL CHFILL (N2, RBLANK, ICH, MSGTXT)
            ICH = ICH + 2
 665        CONTINUE
         ILEN = 81 - ICH
         CALL CHFILL (ILEN, RBLANK, ICH, MSGTXT)
         CALL MSGWRT (N5)
C                                          Secondary axes values
         IF ((NCHLAB(1).LE.0) .AND. (NCHLAB(2).LE.0)) GO TO 900
            ICH = 8
            DO 670 I = 1,2
               IF (NCHLAB(I).LE.0) GO TO 670
                  CALL CHPACK (NCHLAB(I), SAXLAB(1,I), ICH, MSGTXT)
                  ICH = ICH + NCHLAB(I)
                  CALL CHFILL (N2, RBLANK, ICH, MSGTXT)
                  ICH = ICH + 2
 670           CONTINUE
            ILEN = 81 - ICH
```

```
            CALL CHFILL (ILEN, RBLANK, ICH, MSGTXT)
            CALL MSGWRT (N5)
            GO TO 900
.....
C                                           normal TV close
 900  CALL TVCLOS (CATBLK, JERR)
      GO TO 999
```

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 catalogue 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, MINI3, and MDIS3. 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 two functions which generate and use blotches: the verb TVSTAT which returns image statistics within the blotch area and the task BLANK which blanks out all pixels outside the blotch. In both, 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). The subroutine BLTFIL does the filling of the polygons on the TV graphics screen for both TVSTAT and BLANK.

10.3.9 Roam

Roam is mode of display which requires multiple gray-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 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 verb TVMOVIE is 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 gray-scale TV memory is subdivided into a 2 x 2, 4 x 4, or 8 x 8 matrix of images of consecutive planes of the cube. During the display phase, the zoom factor is set to 2, 4, or 8, respectively, 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 gray-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 alogorithm 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 /TVCHAR/ common).

The subroutines AU6A and TVBLNK implement the verbs TVBLINK and TVMBLINK. Blinking is simply enabling one gray-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).

10.3.11 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.

- 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.)

- TVHLD loads up to 13-bit image to two TV memories and performs an interactive histogram equilization 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.

- TVHXF does an interactive histogram equilization 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 is someone wishes to do the work.

- TVSLV loads an image, prepared by tasks TVCUB and TVSLD, to the TV. The image is a 3-dimensional representation of a data cube.

- UVDIS attempts to take an FFT of an image and display the complex results on the TV as intensity and color-encoded phase.

10.4 INCLUDES

10.4.1 DTVC.INC

```
C                                                       Include DTVC
      INTEGER*2 NGRAY, NGRAPH, NIMAGE, MAXXTV(2), MAXINT, SCXINC,
     *    SCYINC, MXZOOM, NTVHDR, CSIZTV(2), GRPHIC, ALLONE, MAXXTK(2),
     *    CSIZTK(2), TYPSPL, TVALUS, TVXMOD, TVYMOD, TVDUMS(7),
     *    TVZOOM(3), TVSCRX(16), TVSCRY(16), TVLIMG(4), TVSPLT(2),
     *    TVSPLM, TVSPLC, TYPMOV(16), YBUFF(168)
C                                                       End DTVC
```

10.4.2 CTVC.INC

```
C                                                       Include CTVC
      COMMON /TVCHAR/ NGRAY, NGRAPH, NIMAGE, MAXXTV, MAXINT, SCXINC,
     *    SCYINC, MXZOOM, NTVHDR, CSIZTV, GRPHIC, ALLONE, MAXXTK,
     *    CSIZTK, TYPSPL, TVALUS, TVXMOD, TVYMOD, TVDUMS, TVZOOM,
     *    TVSCRX, TVSCRY, TVLIMG, TVSPLT, TVSPLM, TVSPLC, TYPMOV,
     *    YBUFF
C                                                       End CTVC
```

10.4.3 DTVD.INC

```
C                                                       Include DTVD
      INTEGER*2 TVLUN, TVIND, TVLUN2, TVIND2, TVBFNO
      LOGICAL*2 TVMAP
C                                                       End DTVD
```

10.4.4 CTVD.INC

```
C                                                       Include CTVD
      COMMON /TVDEV/ TVLUN, TVIND, TVLUN2, TVIND2, TVBFNO, TVMAP
C                                                       End CTVD
```

10.5 Y-ROUTINE PRECURSOR REMARKS:

10.5.1 Level 0

10.5.1.1 YCHRW - writes characters into image planes of the TV. The format is 5 by 7 with one blank all around: net 7 in X by 9 in Y 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 some TVs.

```
      YCHRW (CHAN, X, Y, COUNT, STRING, SCRTCH, IERR)

   Inputs: CHAN    I*2      channel select  (1 to NGRAY + NGRAPH)
           X       I*2      X position lower left corner first char.
           Y       I*2      Y position lower left corner first char.
           COUNT   I*2      number of characters in STRING
           STRING  R*4      character string
   Output: SCRTCH  I*2(>)   scratch buffer (dim = 14*count+8 < 1031)
           IERR    I*2      error code of Z...XF:0 - ok
                                                 2 - input error
```

10.5.1.2 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*2       start X position
            Y1      I*2       start Y position
            X2      I*2       end X position
            Y2      I*2       end Y position
            IC      I*2       Channel (1 to NGRAY+NGRAPH)
            BUFFER  I*2(512)  BUFFER(1 - 512) contains desired
                              intensity (size here for I2S)
    Output: IERR    I*2        error code : 0 => ok
```

10.5.1.3 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.

```
      YCUCOR (RPOS, QUAD, CORN, IERR)

   Inputs: RPOS    R*4(2)  X,Y screen pos before  zoom & scroll
   Output: QUAD    I*2     TV quadrant to use for scrolls
                           Out: if in=-1, no scroll, else find
                           quadrant (needs real TV pos)
           CORN    R*4(7)  Image coordinates (pixels)
           IERR    I*2     error code of Z...XF : 0 - ok
                                                  2 - input error
```

10.5.1.4 YCURSE - reads cursor positions and controls the blink and visibility of the TV cursor.

```
      YCURSE (OP, WAIT, CORR, RPOS, QUAD, EVTMOD, IERR)

   Inputs: OP      R*4     'READ' read cursor position
                           'ONNN' place cursor at RPOS & leave on
                           'OFFF' turn cursor off
                           'BLNK' reverse sense of cursor blink
           WAIT    L*2     wait for event & return RPOS & EVTMOD
                           (done on all OPs)
           CORR    L*2     T => correct RPOS for zoom & scroll
   In/Out: RPOS    R*4(2)  X,Y screen pos before  zoom & scroll
           QUAD    I*2     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: EVTMOD  I*2     event # (0 none, 1-7 low buttons,
                           8-15 the "quit" button)
           IERR    I*2     error code of Z...XF : 0 - ok
                                                  2 - input error
```

10.5.1.5 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 + red	blotch
4	insert	black	label backgrounds
5-7	add	nothing	null channels
8	insert	purple	cursor

YGRAPH (OP, CHAN, SCRTCH, IERR)

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

10.5.1.6 YLNCLR - computes a piecewise linear OFM and writes it to the TV. If NEND(NPOINT) is 256 (512) then the OFM is repeated 4 (2) times.

YLNCLR (COLOR, NPOINT, NEND, SLOPE, OFFSET, GAMMA, BUFFER, IERR)

Inputs:	COLOR	I*2	color bit mask: RGB = 421
	NPOINT	I*2	# of segments
	NEND	I*2	end points of segments
	SLOPE	R*4(NPOINT)	slopes of segments
	OFFSET	R*4(NPOINT)	offsets of segments
	GAMMA	R*4	power applied to colors (1 /gamma)
Output:	BUFFER	I*2(1024)	scratch buffer
	IERR	I*2	error code of Z...XF : 0 - ok

Form is C = (i-1)*SLOPE + OFFSET with 0 <= C <= 1.0.

10.5.1.7 YSLECT - enables and disables gray and graphics planes.

```
      YSLECT (OP, CHAN, COLOR, BUFFER, IERR)

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

10.5.1.8 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)
```

10.5.1.9 YZERO - fills an TV memory plane with zeros the fast way.
Note: this is equivalent to YINIT, but avoids linking with all the routines called by the main parts of YINIT.

```
      YZERO (CHAN, IERR)

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

10.5.1.10 YTVCLS - closes TV device associated with LUN removing any EXCLusive use state and clears up the FTAB.

```
      YTVCLS (LUN, IND, IERR)

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

10.5.1.11 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)
```

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

```
      YTVOPN (LUN, IND, IERR)

   Inputs:  LUN    I*2    Logical unit number to use
   Output:  IND    I*2    Pointer to FTAB entry for open device
            IERR   I*2    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
```

10.5.2 Level 1

.tp 19

10.5.2.1 YCRCTL - reads/writes the cursor/trackball control register of TV.

```
     YCRCTL (OP, ON, X, Y, LINKX, LINKY, RBLINK, BUTTON,
    *   VRTRTC, IERR)

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

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

10.5.2.3 YINIT - initializes the TV subunits: doing everything.

```
      YINIT (SCRTCH, IERR)

   Output: SCRTCH  I*2(1024)  scratch buffer (can be 256 for CHAN &
                              1 for ZERO & REST)
           IERR    I*2        error code of Z...XF - 0 => ok
                                                     2 => input error
```

10.5.2.4 YLUT - reads/writes full channel look up tables to TV.

```
      YLUT (OP, CHANNL, COLOR, VRTRTC, LUT, IERR)

   Inputs: OP      R*4        'READ' from TV, 'WRIT' to TV
           CHANNL  I*2        channel select bit mask
           COLOR   I*2        color select bit mask (RGB <-> 421)
           VRTRTC  L*2        T => do it only during vertical retrace
   In/Out: LUT     I*2(256)   look up table (ls 9 bits used)
   Out:    IERR    I*2        error code of Z...XF : 0 => ok
```

10.5.2.5 YOFM - reads/writes full OFM look up tables to TV.

```
      YOFM (OP, COLOR, VRTRTC, OFM, IERR)

   Inputs: OP      R*4        'READ' from TV, 'WRIT' to TV
           COLOR   I*2        color select bit mask (RGB <-> 421)
           VRTRTC  L*2        T => do it only during vertical retrace
   In/Out: OFM     I*2(1024)   look up table (ls 10 bits used)
   Out:    IERR    I*2        error code of Z...XF : 0 => ok
```

10.5.2.6 YSCROL - writes the scroll registers on the TV.

```
      YSCROL (CHANNL, SCROLX, SCROLY, VRTRTC, IERR)

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

10.5.2.7 YSPLIT - reads/writes the look up table/ split screen control registers of the TV.
Quadrants are numbered CCW from top right.

```
      YSPLIT (OP, XSPLT , YSPLT , RCHANS, GCHANS, BCHANS,
     *   VRTRTC, IERR)

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

10.5.2.8 YZOOMC - writes (ONLY!) the zoom control registers of the TV.

```
      YZOOMC (MAG, XZOOM, YZOOM, VRTRTC, IERR)

   Inputs: MAG     I*2   0-3 for magnification 1,2,4,8 times, resp.
           XZOOM   I*2   X center of expansion (1-512, 1 => LHS)
           YZOOM   I*2   Y center of expansion (1-512, 1 => bot)
   Output: IERR    I*2   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)

10.5.3.1 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, SHIFT, CARYIN, CARRY, EQUAL, IERR)
```

Inputs:	OP	R*4	'READ' from TV or 'WRIT' to TV
In/Out:	ARMODE	L*2	T => arithmetic mode F => logic mode
	BFUNC	I*2	function number (1-16) in blotch
	NFUNC	I*2	function number (1-16) outside blotch
	CONSTS	I*2(8)	constant array (may select as ALU output)
	OUTSEL	I*2(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*2	T => extend sign of OFM on input to ALU
	ESHIFT	L*2	T => extend sign of ALU output if SHIFT
	SHIFT	L*2	T => right shift ALU output
	CARYIN	L*2	T => add one to arithmetic results
Output:	CARRY	L*2	T => carry condition occurred in frame
	EQUAL	L*2	T => equal condition occurred in frame
	IERR	I*2	error code of Z...XF : 0 - ok
			2 - input error

10.5.3.2 YFDBCK - sends a feedback command to the TV.

```
      YFDBCK (COLOR, CHANNL, BITPL, PIXOFF, BYPIFM, EXTERN,
     *    ZERO, ACCUM, ADDWRT, IERR)
```

Inputs:	COLOR	I*2	bit map of color to be fedback (RGB = 4,2,1)
	CHANNL	I*2	bit map of channels to receive feedback
	BITPL	I*2	bit map of bit planes to receive feedback
	PIXOFF	I*2	offset fedback image to left by 0 - 1 pixels
NOTE:	I2S literature claims only 1 bit here not the three that their software (NOT this routine) uses.		
	BYPIFM	L*2	F => image goes thru IFM lookup before store
	EXTERN	L*2	T => image from external input (iedigitizer)
	ZERO	L*2	T => feed back all zeros
	ACCUM	L*2	T => use 16-bit accumulator mode then CHANNL must give even-odd pair lsbyte goes to even (lower) # channel
	ADDWRT	L*2	T => additive write F => replace old data
Outputs:	IERR	I*2	error code of Z...XF: 0 -> ok
			2 -> input error

10.5.3.3 YGYHDR - builds an TV header to write image data. The actual I/O must be done by calls to Z...XF.

```
      YGYHDR (OP, NPIXEL, XINIT, YINIT, IANGLE, CHANNL,
     *    PLANES, PACKED, BYPIFM, BYTE, ADDWRT, ACCUM, VRTRTC, HEADER,
     *    IERR)
```

Inputs:	OP	R*4	'READ' from TV or 'WRIT' to TV
	NPIXEL	I*2	number of pixel values to I/O
	XINIT	I*2	first pixel X coordinate (1-512, 1 -› LHS)
	YINIT	I*2	first pixel Y coordinate (1-512, 1 -› bot)
	IANGLE	I*2	(0 =› data I/O horizontal to right, 1 =› up, 2 =› to left, 3 =› down)
	CHANNL	I*2	channel select bit mask
	PLANES	I*2	bit plane select bit mask
	PACKED	L*2	T =› 2 values/word, F =› 1 value/word
	BYPIFM	L*2	F =› IFM lookup applied to data (write)
	BYTE	L*2	T =› 8 values/byte (needs XINIT = 8*n+1)
	ADDWRT	L*2	T =› OR data with present memory contents
	ACCUM	L*2	T =› use 16-bit accumulator mode
	VRTRTC	L*2	T =› do it only during vertical retrace
Output:	HEADER	I*2(8)	header to be sent to TV
	IERR	I*2	error code of 0 =› ok 2 =› input error

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

10.5.3.5 YRHIST - reads the histogram of the output of a selected OFM of the TV.
**** Warning: the results are 18-bit integers stored in a standard AIPS pseudo I*4 order (ls 16 bits in first word).

```
      YRHIST (MODE, COLOR, INITI, NINT, HISTOG, IERR)

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

10.5.4 Selected Applications Subroutines

10.5.4.1 TVOPEN - opens the TV, passing pointers through common /TVDEV/.

```
      TVOPEN (BUF, IERR)

   OUTPUTS: BUF   I*2(256)   Scratch buffer
            IERR  I*2        Error return from ZOPEN
                             = 10 TV unavailable to this version
```

10.5.4.2 TVCLOS - closes the TV device and the TV status disk file, updating the information on the disk.

```
      TVCLOS (BUF, IERR)

   Outputs: BUF   I*2(256)   Scratch buffer
            IERR  I*2        Error code : 0 => ok
                             else as returned by ZFIO
```

10.5.4.3 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.

```
      TVFIND (MAXPL, TYPE, IPL, UNIQUE, CATBLK, SCRTCH,
     *   IERR)
```

Inputs:	MAXPL	I*2	Highest plane number allowed (i.e. do graphics count?)
	TYPE	I*2	2-char image type to restrict search
Output:	IPL	I*2	Plane number found
	UNIQUE	L*2	T => only one image visible now (all types)
	CATBLK	I*2(256)	Image catalog block found
	SCRTCH	I*2(256)	Scratch buffer
	IERR	I*2	Error code: 0 => ok 1 => no image 2 => I/O error in image catalog 3 => TV error

10.5.4.4 TVWIND - sets windows for normal and split screen TV loads.

```
      TVWIND (TYPE, PXINC, BLC, TRC, ICHAN, ITVC, IWIN,
     *   IERR)
```

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

10.5.4.5 TVLOAD - loads 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,
     *   IERR)

   Inputs:
      LUN       I*2     Logical unit # of map file
      IND       I*2     FTAB pointer for map file
      IPL       I*2     Channel to load
      PXINC     I*2(2)  Increment in x,y between included pixels
      IMAWIN    I*2(4)  TV corners: BLC x,y TRC x,y
      WIN       I*2(4)  Map window: ""
      BUFSZ     I*2     Buffer size in bytes
   Outputs
      IERR      I*2     Error code: 0 => ok
                                    1 => input errors
                                    2 => MINI3 errors
                                    3 => MDIS3 errors
   Commons: /MAPHDR/ CATBLK  image header
            /IMBUF / BUFF    work space for I/O
```

10.5.4.6 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 (ICHAN, NLEVS, IERR)

   Inputs: ICHAN     I*2     Selected gray-scale channel
           NLEVS     I*2     Number of gray levels (usually MAXINT+1)
   OUTPUT: IERR      I*2     Error code: 0 -> ok
                             else set by Z...XF
```

10.5.4.7 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.

```
      IMANOT (OP, X, Y, IANGL, CENTER, COUNT, STRING,
     *   SCRTCH, IERR)
```

Inputs:	OP	R*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*2	X position of string
	Y	I*2	Y position of string
	IANGL	I*2	0 - horizontal, 3 - vertical (DOWN)
	CENTER	I*2	0 - XY are lower left first character 1 - XY are center of string 2 - XY are top right of last character
	COUNT	I*2	number of characters in STRING
	STRING	R*4()	character string
Output:	SCRTCH	I*2(>)	scratch buffer (>256, 14*count)
	IERR	I*2	error code of Z...XF : 0 - ok 2 - input error

10.5.4.8 IMCHAR - causes characters to appear on the TV by calling IMCHRW.

```
      IMCHAR (CHAN, X, Y, IANGL, CENTER, COUNT, STRING,
     *   SCRTCH, IERR)
```

Inputs:	CHAN	I*2	channel number (1 - NGRAY+NGRAPH)
	X	I*2	X position of string
	Y	I*2	Y position of string
	IANGL	I*2	0 - horizontal (to right), 3 - vertical (down) ONLY ones supported.
	CENTER	I*2	0 - XY are lower left of first character 1 - XY are center of string 2 - XY are upper right of last character
	COUNT	I*2	number of characters in STRING
	STRING	R*4()	character string to go to TV
Output:	SCRTCH	I*2(>)	scratch buffer (14*count)
	IERR	I*2	error code of Z...XF: 0 - ok 2 - input error

10.5.4.9 IMVECT - writes a connected sequence of line segments on a TV channel calling YCNECT

```
IMVECT (OP, CHAN, COUNT, XDATA, YDATA, SCRTCH, IERR)

Inputs: OP      R*4          'ONNN' line of ones (max intensity)
                             'OFFF' line of zeros (min intensity)
        CHAN    I*2          channel number (1 to NGRAY+NGRAPH)
        COUNT   I*2          number of X,Y pairs ( > 1)
        XDATA   I*2(COUNT)   X coordinates X1,X2,...
        YDATA   I*2(COUNT)   Y coordinates Y1,Y2,...
Output: SCRTCH  I*2(512)     scratch buffer
        IERR    I*2          error code of Z...XF - 0 => ok
                                                    2 => input error
```

10.5.4.10 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*2          channel select bit mask
        ICOLOR  I*2          color select bit mask
In/Out: ITYPE   I*2          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*4(2)       Cursor position: initial -> final
Output: BUFFER  I*2(>768)    scratch buffer
        IERR    I*2          error code of Z...XF: 0 => ok
```

10.5.4.11 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*2       not = 0  => event has occurred
        DOCOR  L*2       Scroll correction parameter for YCURSE
        QUAD   I*2       quadrant parameter for YCURSE
In/out: RP     R*4(2)    cursor position read (fixed on wraps)
        PP     R*4(2)    previous cursor position
        IT     I*2(3)    time of last action
Output: DOIT   L*2       T => something has happened.
```

10.5.4.12 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, BSC, BZE, RANG)

   INPUTS:
      IR(2)    R*4   Range values specified by user
      MMAX     R*4   Map maximum value from header
      MMIN     R*4   Map minimum value
      BSC      R*8   Map scaling factor from header
      BZE      R*8   Map zero offset from header
   OUTPUTS:
      RANG(2)  R*4   Output range values calculated using
                        defaults and map scaling
```

10.5.4.13 DECBIT - translates a decimal based channel number into a binary channel no. e.g. 1453 => 2**0 + 2**3 + 2**4 + 2**2 A maximum of nine channels are addressable (6 at a time)

```
      DECBIT (NMAX, ICHAN, IPL, LOW)

   INPUTS:
      NMAX        I*2     Maximum allowed channel number
      ICHAN       I*2     Input channel decimal number
   OUTPUTS:
      IPL         I*2     Binary channel # pattern
      LOW         I*2     Lowest of specified channels
```

10.5.4.14 MOVIST - sets and resets the movie status parameters in the TV common.

```
      MOVIST (OP, ICHAN, NFR, NFRPCH, MAG, IERR)

   Inputs: OP       R*4   'ONNN' when turning on a movie
                          'OFFF' when clearing channel(s)
           ICHAN    I*2   Bit pattern of channels involved (OFFF)
                          Actual first channel number (1-NGRAY, ONNN)
           NFR      I*2   Number of frames in movie total (ONNN)
           NFRPCH   I*2   Number of frames per TV channel (ONNN)
           MAG      I*2   Magnification number (0 - 3, ONNN)
   Output: IERR     I*2   Error = 2 => bad input, else ok
```

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 independant "commands" that tell a device how to draw the plot. As of the time this chapter was written, this file is always an extension file associated with a cataloged file. However, the plot file could itself be 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 Tektronics 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 an variety of coordinate systems in use in astronomy. The complexity of these routines is commensuate to their power. Fortunately, a set of plot program templates exist to provide a starting point. These routines are described in a latter 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.	I*2(3)	Task name (2 chars / word)
2.	I*2(6)	Date/time of file creation YYYY,MM,DD,HH,MM,SS
3.	I*2	Number of words of task parameter data
4.	R*4(*)	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*2 variables or ASCII 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. This causes 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 disk volume number, sss is the Catalog slot number of the associated map, and vv is the version number.

11.2.3 Types Of Plot File Logical Records

11.2.3.1 Initialize Plot Record. - The first logical record in a plot file must be of this type.

FIELD	TYPE	DESCRIPTION
1.	I*2	Opcode (equal to 1 for this record type).
2.	I*2	User number.
3.	I*2(3)	Date: yyyy, mm, dd
4.	I*2	Type of plot: 1 = miscellaneous 2 = contour 3 = grey scale

4 = 3D profile
5 = slice
6 = contour plus polarization lines
7 = histogram

11.2.3.2 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*2	Opcode (equal to 2 for this record type).
2.	I*2	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*2	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*2(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*2(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*2(4)	10 * the number of character positions outside the plot window on the left, bottom, right, and top respectively
7.	I*2(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.

11.2.3.3 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*2	Opcode (equals 3 for this record type).
2.	I*2	Lowest allowed pixel intensity.
3.	I*2	Highest allowed pixel intensity.
4.	I*2	Number of pixels on the X axis.
5.	I*2	Number of pixels on the Y axis.

11.2.3.4 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*2	Opcode (equals 4 for this record type).
2.	I*2	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*2	Scaled Y position.

11.2.3.5 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*2	Opcode (equals 5 for this record type).
2.	I*2	Scaled final X position.
3.	I*2	Scaled final Y position.

11.2.3.6 Write Character String Record. - This record tells a device to write a character string starting at the current position.

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

11.2.3.7 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*2	Opcode (equals 7 for this record type).
2.	I*2	Number of pixel values.
3.	I*2	Angle code: 0 = write pixels horizontally. 1 = write pixels vertically (up).
4.	I*2	X offset in characters * 100.
5.	I*2	Y offset in characters * 100.
6.	I*2(n)	n (equal to field 2) pixel values.

11.2.3.8 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 interpreted by routines such as AU9A (TKSKYPOS, TKMAPPOS, ect.). Routines that write to non-interactive graphics devices (PRTPL) ignore this record.

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

11.2.3.9 End Of Plot Record. - This record marks the end of a plot file.

FIELD	TYPE	DESCRIPTION
1.	I*2	Opcode (equals 32767 for this record type).

11.3 PLOT PARAFORM TASKS

11.3.1 Introduction

Three paraform tasks (PFPL1, PFPL2 and PFPL3) 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 IMNAME, BLC, TRC, XYRATIO, PIXRANGE, etc., and work for both integer and floating point maps. 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 axis of the plot are the same as the X and Y axis 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 axis 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 axis of the plot has no real relation to the map axis. An example of this type of plot is the histogram produced by task IMEAN.

The structure of all three paraform tasks are very similar. The major differences are in subroutine PLINIT (the subroutine that initializes the commons used in labeling the plot), PLLABL (this routine does the actual labeling), and in the example plots in subroutine PLTTOR. 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 a map file corresponding to the users inputs from AIPS.

2. Create an extension file of type PL (plot) to be associated with the map file. The header of the map 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 PLTTOR. The programmer will have to modify the code in PLTTOR 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. On the Vax, one should copy files NOTPGM:PFPLn.FOR, and HLPFIL:PFPLn.HLP ("n" stands for 1, 2 or 3) to a user directory and work with the program there. Useful information on running a task from a user's directory, and on compiling and linking tasks and on modifying skeleton tasks can be found in other chapters of this manual.

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

```
PROGRAM PFPLn
```

and the data statement

```
DATA PRGNAM /'PF','PL','n '/
```

should be changed to use the new name. The name in the HELP file should also be changed.

Next, the programmer should compile and link the task in his directory and try running it from AIPS by using adverb VERSION. This will assure the programmer that the task does work, and also demonstrate the current output of the task.

11.3.3 Labeling The Plot

The labeling of the plot takes place in two subroutines called by subroutine PLTTOR. PLINIT 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 PLLABL uses the information provided by PLINIT 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 PLINIT or PLLABL 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 NTEXT (number of lines of text), and TEXT (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 NTEXT to zero to suppress all of the lines. If the programmer wishes to use more than two lines, then the second dimension of array TEXT must be changed in all the routines in which TEXT is 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 PLTTOR. 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 does not use the Easy I/O (WaWa) package, but instead 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 standard 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 with scaling 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, and handling both floating and integer maps in the upper level program.

There are four I/O routines in this program, MAKNAM (fills in a real 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, and converts the values to floating point numbers, if necessary). 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 PLTTOR) to read data from the map, one row at a time.

The I/O routines in this program use a common named MAPHDR. This name was chosen to interface with several of the plotting routines which expect this common to have the map header as the first 256 words. 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. Integer (1) or real (2) flag.

4. Blanked value for integers 0=no blanking.

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 common /INPARM/, which can be found in the main program and in several of the subroutines. The variable NPARMS is initialized in an assignment statement in the main program. This must be changed to correspond to the new number of floating point numbers received from AIPS.

11.3.7 The Three Paraform Plot Tasks

11.3.7.1 PFPL1 - This task should be used when developing a plotting task in which the X and Y axis of the plot are the same as the X and Y axis 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.

If PCODES(1) equals

1 then the plot axis consists 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 XUNIT 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 YUNIT 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 TITLE 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 PLTTOR 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.

11.3.7.2 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 XUNIT 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 YUNIT 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 TITLE 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.

11.3.7.3 PFPL3 - This task should be used when developing a plotting task in which the X and Y axis have no relation to the map X and Y axis. The plot could be of a function, a histogram of some values, or a table.

The only PCODES value 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 TITLE. If PCODES(5) is not zero then this signals the existence of grey scale pixels. The program automatically uses whatever strings are in variables XUNIT and YUNIT 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 PLTTOR reads the map to determine the histogram values and the range of the Y axis (number of pixels). Then the standard initializing routine (PLINIT) and labeling routine (PLLABL) 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.

11.3.8 Routines

11.3.8.1 PLEND - Do some plotting cleanup functions. Write "end of plot" record, close plot file, check for vectors that were off the plot.

```
      PLEND (IOBLK, ISTAT)
In/Out:
  IOBLK  I*2(256)  Work I/O buffer
  ISTAT  I*2       0-successful completion, other-dies unnaturally.
```

11.3.8.2 PLPOS - This routine will put a 'position vector' command in an AIPS plot file.

```
      PLPOS (X, Y, IERR)
Inputs:
  X      R*4  X value.
  Y      R*4  Y value.
  COMMON /PLTCOM/
Output:
  IERR   I*2  Error code.  0 means OK.
```

11.3.8.3 PLVEC - This routine will put a 'draw vector' command in an AIPS plot file.

```
      PLVEC (X, Y, IERR)
Inputs:
  X       R*4  X value.
  Y       R*4  Y value.
  COMMON /PLTCOM/
Output:
  IERR    I*2  Error code.  0 means OK.
```

11.3.8.4 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 (NP, RPARM, IERR)
Inputs:
  NP      I*2      Number of floating point words in parameter list
                   received from AIPS.
  RPARM   R*4(NP)  AIPS parameters.
Output:
  IERR    I*2      Error code. two digit, first digit indi-
                   cates subroutine: 1: MAPOPN, 2: MADDEX,
                   3: ZPHFIL, 4: GINIT, second digit indi-
                   cates error code of that subroutine.
```

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

```
      PLGRY (IANGLE, NVAL, VALUES, IERR)
Inputs:
  IANGLE  I*2      Angle code. 0 = horizontal, 1 = vertical.
  NVAL    I*2      The number of grey scale pixel values.
  VALUES  R*4(?)   Grey scale values.
Output:
  IERR    I*2      Error code. 0=ok.
```

11.3.8.6 MAKNAM - This routine will construct a WaWa I/O name string, given the values that make up the thing.

```
      MAKNAM (INAME, INCLAS, SEQ, VOL, TYPE, USER, NAMSTR)
Inputs:
  INAME   R*4(3)  file name
  INCLAS  R*4(2)  file class
  SEQ     R*4     file sequence number.
  VOL     R*4     file disk volume.
  TYPE    R*4     file type.
  USER    R*4     file user number.
```

Output:

NAMSTR	R*4(9)	"Name string" in the tradition of WaWa I/O. NAME(1:3) name, NAME(4:5) class, NAME(6) seq, NAME(7) volume, NAME(8) type, NAME(9) user.

11.3.8.7 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, IHD, IMSTUF, DSCAL, IERR

Inputs:

ILUN	I*2	Logical unit number to use for the map file.
ACCESS	R*4	'READ' or 'WRITE' status to mark catalog.
NAME	R*4(9)	"Name string" in the tradition of WaWa I/O. NAME(1:3) name, NAME(4:5) class, NAME(6) seq, NAME(7) volume, NAME(8) type, NAME(9) user.
BLC	R*4(7)	Bottom left corner of map.
TRC	R*4(7)	Top right corner of map
IBSIZE	I*2	Size of I/O buffer in INTEGER*2 values.

Outputs:

COMMON	/CFILES/	Values updated so that subroutine DIE will close this file.
IHD	I*2(256)	Map header.
IMSTUF	I*2(37)	I/O pointers and stuff that are needed by other I/O routines compatible with this one. They are: 1. LUN 2. FTAB index 3. integer (1) or real (2) flag. 4. Blanked value for integers 0=no blanking. 5. Catalog slot of image. 6. Size of I/O 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 backwrd read on 2nd axis.
DSCAL	R*8(2)	Scale factors to use with this image.
IERR	I*2	Error code. 0=ok.

11.3.8.8 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, IHD, IMSTUF, DSCAL, IERR)

Inputs:

```
  BLC         R*4(7)  Bottom left corner of map.
  TRC         R*4(7)  Top right corner of map
  IBSIZE      I*2     Size of I/O buffer in INTEGER*2 values.
  IHD         I*2(256)Map header.
  IMSTUF(1)   I*2     LUN
  IMSTUF(2)   I*2     FTAB index
  IMSTUF(7)   I*2     Volume number of image.
  IMSTUF(5)   I*2     Catalog slot of image.
  IMSTUF(6)   I*2     Size of I/O buffer in bytes of all things.
Outputs:
  IMSTUF(3)   I*2     Integer (1) or real (2) flag.
  IMSTUF(4)   I*2     Blanked value for integers 0=no blanking.
  IMSTUF(8)   I*2     Number of dimensions in image.
  IMSTUF(9)   I*2     Number of values read per row of image.
  IMSTUF(10-16)       Number of values along all 7 axis
  IMSTUF(17-30)       Window in BLC TRC pairs along all 7 axis.
  IMSTUF(31-36)       Current position on last six axis.
  IMSTUF(37)  I*2     1 if read fwd -1 is bckwrd read on 2nd axis.
  DSCAL       R*8(2)  Scale factors to use with this image.
  IERR        I*2     Error code. 0=ok.
```

11.3.8.9 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, converting integer values to floating point, if necessary.

```
      GETROW (IMSTUF, DSCAL, IOBLK, ROW, EOF, IERR)
Inputs:
  IMSTUF   I*2(37) I/O pointers, LUNs, counters and such. They are
                   set in INTMIO.
  DSCAL    R*8(2)  Actual value = DSCAL(1) * disk value  +  DSCAL(2)
In/Out:
  IOBLK    I*2(?)  I/O buffer.
Outputs:
  ROW      R*4(?)  Scaled output row of image.
  EOF      L*2     TRUE means last row specified in INTMIO by the
                   BLC, TRC arguments has been read.
  IERR     I*2     Error code, 0=ok, others from MDISK.
```

CHAPTER 12

USING THE ARRAY PROCESSORS

12.1 OVERVIEW

Many of the more important of the AIPS tasks do a great deal of computation while the cpu of the host computer of most AIPS systems is rather slow. 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 sequence.

12.1.1 Why Use The Array Processor?

The principle reason for using an array processor is speed. The design of most array processors optimizes its performance for repetitive arithmetic operations making it mush 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 transfered to and the results transfered 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 FFTs) operations which are highly reptitious 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 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.

- AIPS currently uses APs essentially as vector arithmetic units. That is, data is sent into the AP, some (usually vector) operation is done, and the reults is returned to the host CPU. The principle 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 manufactures. Many of the more complex of the AIPS operations are better described as pipelined scalar operations. In the AIPS useage, most high level control and use of disk storage is done in the host CPU and only arithmetic operations are done in the AP.

- 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 dissappear from the task's assigned AP data memory between these calls. This concept has little meaning for virtual AP except that the data memory is cleared after a QINIT call.

- 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 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.

- 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 and the new, cheap APs.

- 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.

- 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 CDCH.INC) as (1024KAPWRD-1). Word addressing only is used.

- 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 algrothyms (especially gridding and the in-core CLEAN) in terms of vector operations. This reformulation will likely require vector logical operations, Gather, Scatter, Merge and Compress operations.

- 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 most also be ways to synchronize the operations. Then 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.

12.3.1.1 Q Routine Arguments - The call arguments to the Q routines (AP-routines) are local long integers (Integer4). 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.

12.3.1.2 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 includes DDCH.INC and CDCH.INC. The size of the AP is in the I*2 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.

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 then 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/ (includes DBPR.INC and CBPR.INC) which control the AP roller subroutine QROLL. The text of these includes is shown at the end of this chapter and the use of the values are described in the detailed description of QROLL given at the end of this chapter.

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, will create a scratch file (using the /CFILES/ 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 catalogued 'MA' type files or scratch files using the /CFILES/ common system. APIO can handle arbitrary row lengths. This is done by breaking up the logical records if they are larger than 16384 bytes or the buffer size.

NOTE: it is important that data read with APIO either have a logical record length of 16384 bytes or less or have been written by APIO with the same buffer size; this may be a problem for catalogued files if the row length is greater that 4096 for real format data or 8192 for scaled integers. The problem is that APIO will break up logical records if they are longer than 16384 bytes or the buffer size and MDISK may leave blank space on the disk if the shorter logical record does not fill a disk sector. For this reason it is good to use a buffer size of 16384 bytes or greater when reading or writing catalogued files with APIO. It is IMPORTANT to always use the same size buffer when accessing a given file.

Useage notes for APIO:

1. Opening the file.

   If APIO determines that the file is not open it will do so. The file can be either a catalogued file or a scratch file using the /CFILES/ common system. If the catalogue 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 catalogued), 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 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 very wrong results which go away when apparently unnecessary timing calls are added; such as calls to QWR between calls to computation routines.

We use three timing calls:

- QWD suspends the calling program until data transfers to or from the AP are complete.

- QWR suspends the calling program until the AP completes all computations.

- 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.

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. APLINK creates a Fortran or host assembly language routine with the executable module in a data statment.

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.

12.3.6.1 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. The source code for NRAO written microcode is kept in the file FPSSUB:WDC.AP.

12.3.6.2 Vector Function Chainer. - The principle 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 imaginary part of the last complex value into the real part of the first value in the array. This is allowed because the imaginary part of the first value and the real part of the last value are always zero. This convention allows the use of the same AP memory or disk space for the input and output arrays from a real-to-complex FFT.

We also adopt the convention for FFTs that the second half of a one dimensional array come 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 aperature plane (reverse transform).

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

The FFT routines require REAL format data without blanking in an array which is a power of two on a side. In addition, the center of an image in a catalogued 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 catalogued file, float scaled integer input, 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

Since not all systems have array processors and many AIPS systems are running on VAXes which have very large address spaces and virtual memory, there is a set of Fortran and assembly language routines which emulate the functions of an array processor, ie. the "pseudo-array processor". The pseudo-AP consists of a Common, obtained by the INCLUDEs DAPC.INC, CAPC.INC, and EAPC.INC, which serves as the AP data memory and a set of routines which operate on data in this common. There are pseudo-AP routines duplicating all of the functions of the true array processor so that a task is simple linked with the appropriate library to use either a true or the pseudo-AP. Listings of the pseudo-AP includes appear at the end of this chapter. Since Fortran requires one relative indexing whereas the AP addressing is zero relative, pseudo AP routines must add 1 to addresses.

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 /CFILES/ 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 REAL N x M arrays in the /CFILES/ scratch files
C   ISCRA and ISCRB and writes the result in scratch file ISCRC.
```

```
C    Inputs:
C     ISCRA      I*2  /CFILES/ scratch file number of first input file.
C     ISCRB      I*2  /CFILES/ scratch file number of second input file.
C     ISCRC      I*2  /CFILES/ scratch file number of output file.
C     N          I*2  Length of a row in the array
C     M          I*2  Number of rows in the array.
C    Output:
C     IRET       I*2  Return error code 0=>OK, otherwise APIO error
C                     code.
C-----------------------------------------------------------------------
      INTEGER*2 N, M, INCR, FLIST(22,3), LOOP, IRET,
     *   ISCRA, ISCRB, ISCRC,
     *   NO, N22
      INTEGER*4 APLOCA, APLOCB, APLOCC, LEN, KAP, BO4,
     *   ZERO, ONE, TWO
      REAL*4 BUFF1(4096), BUFF2(4096), BUFF3(4096), READ, WRITE, CLOSE
      INCLUDE 'INCS:DDCH.INC'
      INCLUDE 'INCS:CDCH.INC'
      DATA READ, WRITE, CLOSE /'READ','WRIT','CLOS'/
      DATA NO, N22 /0,22/,  ZERO, ONE, TWO /0,1,2/
C-----------------------------------------------------------------------
C                                          Setup for APIO
      CALL FILL (N22, NO, FLIST)
C                                          Pixel type = floating
      FLIST(4,1) = 0
C                                          Size of array
      FLIST(5,1) = N
      FLIST(6,1) = M
C                                          Buffer size (4096 reals)
      FLIST(13,1) = 4096 * 2 * NWDPFP
C                                          Copy for other files
      CALL COPY (N22, FLIST(1,1), FLIST(1,2))
      CALL COPY (N22, FLIST(1,1), FLIST(1,3))
C                                          Set LUNs
      FLIST(1,1) = 16
      FLIST(1,2) = 17
      FLIST(1,3) = 18
C                                          Set /CFILES/ 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 (ZERO, ZERO, 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, ONE, APLOCB, ONE, APLOCC, ONE, LEN)
C                                          Wait for opertaion to finish
      CALL QWR
C                                          Write result to disk.
      CALL APIO (WRITE, 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 (CLOSE, FLIST(1,1), APLOCA, BUFF1, IRET)
C                                          Check for error
      IF (IRET.NE.0) GO TO 999
      CALL APIO (CLOSE, FLIST(1,2), APLOCB, BUFF2, IRET)
C                                          Check for error
      IF (IRET.NE.0) GO TO 999
      CALL APIO (CLOSE, FLIST(1,3), APLOCC, BUFF3, IRET)
 999  RETURN
C-------------------------------------------------------------------------
      END
```

12.6 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 statments as described in the following list.

- Dxxx.INC. These INCLUDE files contain Fortran type (with dimension) declarations.
- Cxxx.INC. These files contain Fortran COMMON statments.
- Exxx.INC. These contain Fortran EQUIVALENCE statments.
- Vxxx.INC. These contain Fortran DATA statments.
- Ixxx.INC. Similar to Dxxx.INC files in that they contain type declarations but the declaration of some varaible is omitted. This type of include is used in the main program to reserve space for the omitted variable in the appropriate common. The omitted variable must be declared and dimensioned separately.
- Zxxx.INC. These INCLUDE files contain declarations which may change from one computer or installation to another.

12.6.1 CAPC.INC

```
C                                                          Include CAPC
      COMMON /APFAKE/ RWORK, APCORE
      COMMON /SPF/ SPAD
C                                                          End CAPC
```

12.6.2 CBPR.INC

```
C                                                          Include CBPR
      COMMON /BPROLC/ XTLAST, DELTIM, DELAY, TRUEAP
C                                                          End CBPR
```

12.6.3 CDCD.INC

```
C                                                              Include CDCH
      COMMON /DCHCOM/ NVOL, NBPS, NSPG, NBTB1, NTAB1, NBTB2, NTAB2,
     *    NBTB3, NTAB3, NTAPED, CRTMAX, PRTMAX, NBATQS, MAXXPR,
     *    CSIZPR, NINTRN, KAPWRD, NCHPFP, NWDPFP, NWDPDP, NBITWD,
     *    NWDLIN, NCHLIN, NTVDEV, NTKDEV, BLANKV, XPRDMM, XTKDMM,
     *    NTVACC, NTKACC, UCTSIZ, BYTFLP, SYSNAM, VERNAM, USELIM,
     *    IFILIT, RLSNAM
      COMMON /FTABCM/ DEVTAB, FTAB
C                                                              End CDCH.
```

12.6.4 DAPC.INC

```
C                                                              Include DAPC
      REAL*4    APCORE(1), RWORK(4096)
      INTEGER*4 APCORI(1), IWORK(4069), SPAD(16)
      COMPLEX   CWORK(2048)
C                                                              End DAPC
```

12.6.5 DBPR.INC

```
C                                                              Include DBPR
      REAL*4    DELAY
      REAL*8    XTLAST, DELTIM
      LOGICAL*2 TRUEAP
C                                                              End DBPR
```

12.6.6 DDCH.INC

```
C                                                              Include DDCH
      REAL*4 XPRDMM, XTKDMM, SYSNAM(5), VERNAM, RLSNAM(2)
      INTEGER*2 NVOL, NBPS, NSPG, NBTB1, NTAB1, NBTB2, NTAB2,
     *    NBTB3, NTAB3, NTAPED, CRTMAX, PRTMAX, NBATQS, MAXXPR(2),
     *    CSIZPR(2), NINTRN, KAPWRD, NCHPFP, NWDPFP, NWDPDP,
     *    NBITWD, NWDLIN, NCHLIN, NTVDEV, NTKDEV, BLANKV, NTVACC,
     *    NTKACC, UCTSIZ, BYTFLP, USELIM, IFILIT,
     *    DEVTAB(50), FTAB(1)
C                                                              End DDCH.
```

12.6.7 EAPC.INC

```
C                                                                       Include EAPC
      EQUIVALENCE (APCORE, APCORI), (RWORK, IWORK, CWORK)
C                                                                       End EAPC
```

12.6.8 IDCH.INC

```
C                                                                       Include IDCH
      REAL*4 XPRDMM, XTKDMM, SYSNAM(5), VERNAM, RLSNAM(2)
      INTEGER*2 NVOL, NBPS, NSPG, NBTB1, NTAB1, NBTB2, NTAB2,
     *    NBTB3, NTAB3, NTAPED, CRTMAX, PRTMAX, NBATQS, MAXXPR(2),
     *    CSIZPR(2), NINTRN, KAPWRD, NCHPFP, NWDPFP, NWDPDP,
     *    NBITWD, NWDLIN, NCHLIN, NTVDEV, NTKDEV, BLANKV, NTVACC,
     *    NTKACC, UCTSIZ, BYTFLP, IFILIT,
     *    USELIM, DEVTAB(50)
C                                                                       End IDCH.
```

12.7 ROUTINES

12.7.1 Utility Routines

12.7.1.1 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 catalogued 'MA' type files or scratch files using the /CFILES/ common system.

```
      APIO (OPCODE, FLIST, APLOC, BUFFER, IRET)

    Inputs:
       OPCODE    R*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*2 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 catalogues files or
                           /CFILES/ number for scratch files.
                       3 = catalogue slot number for catalogued files,
                           .LE. 0 indicates that the file is a scratch
                           file.
                       4 = pixel type. 0=>floating, 1=scaled integer.
                       5 = Length of a logical record (row) in pixels.
                       6 = Number of rows in a plane.
                       7,8 = P I*4 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 16384
                           bytes to subimage rows. Reversing the
                           order of FLIST(10) and FLIST(12) will
                           cause the rows to be accessed in the
                           reverse order.
                       13 = Buffer size in bytes. 32767 => 32768.
           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
                       17-18 = actual length of logical row as I*4
                       19-22 = Spare.
       APLOC       I*4 Base address in AP for data.
```

```
      BUFFER(*)  R*4 Working buffer.
   Output:
      IRET       I*2 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
                                  MINIT error codes + 20, or
                                  ZOPEN error codes + 30.
```

12.7.1.2 QROLL - checks if it is time to roll the AP as determined by values bet by QINIT, copies the first NWORDs of AP main data memory to a scratch file, gives up the AP, does a task delay for DELAY, goes back into the AP queue and loads the scratch file back into the AP. If NWORD .le. 0 then the AP is not rolled but the AP is given up and the task goes back into the AP queue.
NOTE: APROLL is called by QROLL and uses common /CFILES/ for the scratch file. A scratch file of "type" 'AR' created by APROLL and then destroyed by QROLL after use.
NOTE: LUN 8 is used for I/O and a AIPS "map" I/O slot is opened if the AP memory is actually rolled.
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*4  Number of words of AP memory to save.
                     If .le. 0 the contents of the AP memory are not
                     saved.
    BUFFER(*)   R*4  Work buffer.
    BUFSZ       I*2  Size of BUFFER in bytes.
   Inputs from COMMON /BPROLC/  (set by QINIT)
    TRUEAP      L*2  True if a real AP (to be rolled)
    XTLAST      R*8  Real time AP assigned (min).
    DELTIM      R*8  Time interval between rolls (min).
    DELAY       R*4  Time to delay task (seconds).
   Outputs:
    IRET        I*2  Return error code, 0=>OK
                                        2 => couldn't reload AP.
```

12.7.1.3 DSKFFT - 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.

```
DSKFFT (NR, NC, IDIR, HERM, LI, LW, LO,
*   JBUFSZ, BUFF1, BUFF2, SMAX, SMIN, IERR)
```

Inputs:

NR	I*2	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.
N	I*2	The number of columns in input array (# rows in output).
IDIR	I*2	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*2	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*2	File number in /CFILES/ of input.
LW	I*2	File number in /CFILES/ of work file (may equal LI).
LO	I*2	File number in /CFILES/ of output.
JBUFSZ	I*2	Size of BUFF1, BUFF2 in bytes. Should be large at least 4096 R*4 words.

Output:

BUFF1	R*4	Working buffer
BUFF2	R*4	Working buffer
SMAX	R*4	For HERM=.TRUE. the maximum value in the output file.
SMIN	R*4	For HERM=.TRUE. the minimum value in the output file.
IERR	I*2	Return error code, 0=›OK, otherwise error.

NOTE: Uses AIPS LUNs 23, 24, 25.

12.7.1.4 PEAKFN - searches a region around the center of an image to locate the pixel location of the maximum. Will handle data cubes and either integer or floating images.

```
PEAKFN (LUN, VOL, CNO, IDEPTH, CATBLK, IBUFF,
*   BUFFER, JBUFSZ, PEAKX, PEAKY, IRET)
```

Inputs:

LUN	I*2	Logical unit number to use.

VOL	I*2	Disk on which image resides.
CNO	I*2	Catalog slot number of image.
IDEPTH(5)	I*2	Depth in image of desired plane.
CATBLK(256)	I*2	Catalog header block for image.
IBUFF(*)	I*2	Integer work buffer.
BUFFER(*)	R*4	Real work buffer should be physically the same as IBUFF.
JBUFSZ	I*2	Size of the IBUFF/BUFFER in bytes
Output:		
PEAKX	R*4	X coordinate of peak pixel location.
PEAKY	R*4	Y coordinate of peak pixel location.
IRET	I*2	Return code, 0=› OK, otherwise error.

12.7.1.5 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. Output file is REAL*4 but the input may be either INTEGER*2 of REAL*4. 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 are given, XOFF and YOFF will still be used to end-around rotate the region inside the window.

```
      PLNGET (IDISK, ICNO, CORN, JWIN, XOFF, YOFF,
     *   NOSCR, NX, NY, BUFF1, IBUFF1, BUFF2, BUFSZ1, BUFSZ2,
     *   LUN1, LUN2, IRET)
```

Inputs:		
IDISK	I*2	Input image disk number.
ICNO	I*2	Input image catalogue slot number.
CORN(7)	I*2	BLC in input image (1 & 2 ignored)
JWIN(4)	I*2	Window in plane.
XOFF	I*2	offset in cells in first dimension of the center from MX/2+1 (MX 1st dim. of input win.)
YOFF	I*2	offset in cells in second dimension of the center from MY/2+1 (MY 2nd dim. of input win.)
NOSCR	I*2	Scratch file number in common /CFILES/ for output.
NX, NY	I*2	Dimensions of output file.
BUFF1(*)	R*4	Work buffer
IBUFF1(*)	I*2	Work buffer (should be the same as BUFF1)
BUFF2(*)	R*4	Work buffer.
BUFSZ1	I*2	Size in bytes of BUFF1/IBUFF1
BUFSZ2	I*2	Size in bytes of BUFF2
LUN1, LUN2	I*2	Log. unit numbers to use.
Output:		
IRET	I*2	Return error code, 0 -› OK, 1 - couldn't copy input CATBLK 2 - wrong number of bits/pixel in input map.

3 = input map has inhibit bits.
4 = couldn't open output map file.
5 = couldn't init input map.
6 = couldn't init output map.
7 = read error input map.
8 = write error output map.
9 = error computing block offset
10 = output file too small.

Useage notes:
CATBLK in COMMON /MAPHDR/ is set to the input file CATBLK.

12.7.2 Array Prooessor Routines

The names and funotions 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.

- QGET (HOST, AP, N, TYPE) Transfer data from AP to host
- QGSP (I, NREG) Reads the value of an SPAD register (FPS and pseudo)
- QPUT (HOST, AP, N, TYPE) Transfer data from host to AP.
- QRFT (UDATA, UFT, UPHO, NFT, NDATA) Computes real, inverse Fourier transform from arbitrarily spaced data.
- QWAIT (no arguments) Suspends host until all transfers and computations are complete.
- QWD (no arguments) Suspends host until all transfers of data are complete.
- QWR (no arguments) Suspends host until all computations are complete.
- QBOXSU (A, I, NB, C, J, N) Does a boxcar sum on a vector.
- QINIT (I1, I2, I3) Assigns and initializes AP.
- QRLSE (no arguments) Releases the AP
- QCFFT (C, N, F) Complex FFT.
- QCRVMU (A, I, B, J, C, K, N) Complex - real vector multiply.

- QCSQTR (CORNER, SIZE, ROW) In-place transpose of square complex matrix.

- QCVCMU (A, I, B, C, J, N) Multiplies a complex scalar times the complex conjugate of a complex vector producing a real vector.

- QCVCON (A, I, C, K, N) Take complex conjugate of complex vector.

- QCVEXP (A, I, C, K, N) Complex vector exponentiation.

- QCVJAD (A, I, B, J, C, K, N) Adds a complex vector to the complex conjugate of another complex vector.

- QCVMAG (A, I, C, K, N) Complex vector magnitude squared.

- QCVMMA (A, I, C, N) Finds the maximum square modulus of a complex vector.

- QCVMOV (A, I, C, K, N) Copy one complex vector to another.

- QCVMUL (A, I, B, J, C, K, N, F) Multiply two complex vectors.

- QCVSDI (A, I, B, C, J, N) Divide a weighted complex vector by a complex scalar, weight is multiplied by the amplitude of the scalar.

- QCVSMS (A, I, B, C, J, D, K, N ,FLAG) Subtract a real vecrot tiems a complex scalar from a complex vector.

- QDIRAD (A, IA, B, N) Complex directed add.

- QHIST (A, I, C, N, NB, AMAX, AMIN) Compute histogram of a vector.

- QLVGT (A, I, B, J, C, K, N) Logical vector greater than.

- QMAXMI (A, I, MAX, MIN, N) Find maximum and minimum values in a vector.

- QMAXV (A, I, C, N) Find maximum in an array.

- QMINV (A, I, C, N) Find minimum in an array.

- QMTRAN (A, I, C, K, MC, NC) Matrix transpose.

- QPHSRO (A, I, B, J, PHASO, DELPHS, N) Imposes a phase gradient on a complex vector.

- QPOLAR (A, I, C, K, N) Rectangular to polar conversion.

- QRECT (A, I, C, K, N) Polar to rectangular conversion.
- QRFFT (C, N, F) Real to complex or vice versa fast Fourier transform.
- QSVE (A, I, C, N) Sum of vector elements.
- QSVESQ (A, I, C ,N) Sum of the square of the elements of a vector.
- QVABS (A, I, C, K, N) Vector absolute value.
- QVADD (A, I, B, J, C, K, N) Vector add.
- QVCLIP (A, I, B, C, D, L, N) Vector clip.
- QVCLR (C, K, N) Vector clear.
- QVCOS (A, I, C, K, N) Vector cosine.
- QVDIV (A, I, B, J, C, K, N) Vector division.
- QVEXP (A, I, C, K, N) Vector exponentiation.
- QVFILL (A, C, K, N) Vector fill.
- QVFIX (A, I, C, K, N) Vector real to integer.
- QVFLT (A, I, C, K, N) Vector integer to real.
- QVIDIV (A, I, D1, D2, B, J, N) Divide a vector by the product of two scalar integers.
- QVLN (A, I, C, K, N) Vector natural logarithm.
- QVMA (A, I, B, J, C, K, D, L, N) Vector multiply and add.
- QVMOV (A, I, C, K, N) Copy one vector to another.
- QVMUL (A, I, B, J, C, K, N) Vector multiply.
- QVNEG (A, I, C, K, N) Take negative of a vector.
- QVRVRS (C, K, N) Reverse a vector.
- QVSADD (A, I, B, C, K, N) Vector scalar add.
- QVSIN (A, I, C, K, N) Vector sine.
- QVSMA (A, I, B, C, K, D, L, N) Vector scalar multiply and add.

- QVSMAFX (A, I, B, C, D, L, N) Vector scalar multiply, add and fix.
- QVSMSA (A, I, B, C, D, L, N) Vector scalar multiply, scalar add.
- QVSMUL (A, I, B, C, K, N) Vector scalar multiply.
- QVSQ (A, I, C, K, N) Vector square.
- QVSQRT (A, I, C, K, N) Vector square root.
- QVSUB (A, I, B, J, C, K, N) Subtract two vectors.
- QVSWAP (A, I, C, K, N) Swap two vectors.
- QVTRAN (M, N, IAD, LV) Transpose 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 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 I*4 unless otherwise marked.

12.7.3.1 QGET - Transfer data from AP memory to host core.

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

Inputs:
   AP      I*4     Target area in AP; 0-relative, increment=1
   N       I*4     Number of elements
   TYPE    I*4     Data type:
                    0   data is I*4 in host
                    1   data is I*2 in host
                    2   data is R*4 in host
 Output:
   HOST(*) R*4/I*2 Data array in "host"
```

12.7.3.2 QGSP - Read contents of SPAD register. FPS and Pseudo AP only.

```
      QGSP (I, NREG)

    Inputs:
      NREG  I*4  SPAD register number desired
    Outputs:
      I     I*4  Contents of the SPAD register.
```

12.7.3.3 QPUT - Transfer data from host memory to AP memory.

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

   Inputs:
      AP       I*4      Target area in AP; 0 - relative, increment=1.
      N        I*4      Number of elements
      TYPE     I*4      Data type:
                        0   data is I*4 in host
                        1   data is I*2 in host
                        2   data is R*4 in host
      HOST(*) R*4/I*2 Data array in "host"
```

12.7.3.4 QRFT - Computes a real, inverse fourier transform from arbitarily 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.
```

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

```
QWAIT
```

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

```
QWD
```

12.7.3.7 QWR - Suspend host task until all AP computations are complete.

```
QWR
```

12.7.3.8 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

12.7.3.9 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*2	Dummy
I2	I*2	Dummy

Outputs:

I3	I*2	AP number (Neg. to indicate virtual AP, ie. not to be rolled.

12.7.3.10 QRLSE - Releases the AP, lowers task priority for AIPS batch tasks using a true AP.

QRLSE

12.7.3.11 QCFFT - Do an in-place complex fast Fourier tasnsform.

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

12.7.3.12 QCRVMU - Multiply the elements of a complex vector by the elements of a real vector.

$$C(mK)+iC(mK+1) = (A(mI)*B(mJ)) + i(A(mI+1)*B(mJ)) \quad m=0 \text{ to } N-1$$

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

Inputs:

A	Source complex vector base address.
I	Increment of A
B	Source real vector base address
J	Increment of B
C	Destination vector base address
K	Increment of C
N	Element count

12.7.3.13 QCSQTR - Do an inplace 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.

12.7.3.14 QCVCMU - Multiply a scalar complex value times the complex conjugate of a vector producing a real vector.

C(K) = REAL(B)*A(K)+IMAG(B)*A(K+1) K = 1,N

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

Inputs:

A	Source complex vector base address.
I	Increment of A
B	Address of scalar (real part)
C	Destination real vector base address.
J	Increment of C
N	Element count (reals)

12.7.3.15 QCVCON - Take complex conjugate of a vector.

C(k) = Re(A(k)) - i * Im(A(k)
for k = 0,N-1

QCVCON (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

12.7.3.16 QCVEXP - Exponentiate a complex vector.

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

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
N	Element count

12.7.3.17 QCVJAD - Add the elements of one complex vector to the complex conjugate of the elements of another complex vector.

$$C(k) = Re(A(k))+Re(B(k))+i\ (Im(A(k))-Im(B(k))) \quad \text{for } k = 0,N-1$$

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

Inputs:

A	Source vector base address.
I	Increment of A
B	Source vector base address (conjugate)
J	Increment of B
C	Destination vector base address
K	Increment of C
N	Element count

12.7.3.18 QCVMAG - Square the magnitude of the elements of a complex vector.

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

QCVMAG (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

12.7.3.19 QCVMMA - Find the maximum of the square modulus of a complex vector.

QCVMMA (A, I, C, N)

Inputs:

A	Source vector base address
I	Increment of A
C	Destination vector. 0 = MAX(A ** 2) (real) 1 = location of max (integer)
N	Element count

Also:

SPAD(15) = index of max.

12.7.3.20 QCVMOV - Copy one complex vector to another.

```
QCVMOV (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

12.7.3.21 QCVMUL - Multiply the elements of two complex vectors.

```
(C(mK)+iC(mK+1)) = (B(mJ)+iB(mJ+1)*(A(mI)+iA(mI+1)) if F=1
(C(mK)+iC(mK+1)) = (B(mJ)+iB(mJ+1)*(A(mI)-iA(mI+1)) if F=-1

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

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
F	Conjugate flag, 1 => normal complex multiply -1 => multiply with conj of A

12.7.3.22 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, N-1

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

Inputs:

A	Source vector base address.
I	Increment of A

B	Source scalar address.
C	Destination vector base address
J	Increment of C
N	Element count

12.7.3.23 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, N-1

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

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

Inputs:

A	Source complex vector base address.
I	Increment of A
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
N	Element count
FLAG	Flag, if < 0 multiply complex scalar by i

12.7.3.24 QDIRAD - Do a complex directed add.

```
B(A(IA*J)) = B(A(IA*J))+A(IA*J+1)  for J = 0,N-1
B(A(IA*J)+1) = B(A(IA*J)+1)+A(IA*J+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
B	Destination vector base address
N	Element count

12.7.3.25 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.

12.7.3.26 QLVGT - Logical vector greater than.

C(mK) = 1.0 if A(mI)›B(mJ)
C(mK) = 0.0 if A(mI)=‹B(mJ) for m = 0,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

12.7.3.27 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.

12.7.3.28 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

12.7.3.29 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

12.7.3.30 QMTRAN - Transpose a matrix.

$$C((p+qMC)K) = A((q+pNC)I)$$
$$p = 0 \text{ to } MC-1$$
$$q = 0 \text{ to } NC-1$$

```
      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

12.7.3.31 QPHSRO - Add a phase gradient to a complex array.

B(j) = A(j)*EXP(-i*(PHASO+j*DELPHS)) for j = 0,N-1

```
QPHSRO (A, I, B, J, PHASO, DELPHS, N)
```

Inputs:

A	Source vector base address.
I	Increment of A
B	Destination base address.
J	Increment of B
PHASO	Address of complex unit vector with phase PHASO
DELPHS	Address of complex unit vector with phase DELPHS
N	Element count

12.7.3.32 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
C	Destination vector base address
K	C address increment
N	Element count

12.7.3.33 QRECT - Polar to rectangular vector conversion.

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

12.7.3.34 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.

12.7.3.35 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

12.7.3.36 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

12.7.3.37 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

12.7.3.38 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

12.7.3.39 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

12.7.3.40 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

12.7.3.41 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

12.7.3.42 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

12.7.3.43 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

12.7.3.44 QVFILL - Fill a vector with a constant.

C(mK) = A for m = 0 , 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

12.7.3.45 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

12.7.3.46 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

12.7.3.47 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.

12.7.3.48 QVLN - Take the natural logrithm 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

12.7.3.49 QVMA - Multiply two vectors and adds a third.

D(mL) = (A(mI) * B(mJ)) + C(mK) for m = 0, 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

12.7.3.50 QVMOV - Copy the elements of one vector to another.

C(mK) = A(mI) for m = 0, 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

12.7.3.51 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

12.7.3.52 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

12.7.3.53 QVRVRS - Reverse the elements in a vector.

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

QVRVRS (C, K, N)

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

12.7.3.54 QVSADD - Add a scalar to the elements of a vector.

C(mK) = B + A(mI) for m = 0, 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

12.7.3.55 QVSIN - Take the sine of the elements of a vector.

C(mK) = SIN (A(mI)) for m = 0,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

12.7.3.56 QVSMA - Multiply the elements of a vector by a scalar and adds to the elements of another vector.

D(mL) = (A(mI) * B) + C(mK) for m = 0, 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

12.7.3.57 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,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

12.7.3.58 QVSMSA - Multiply the elements of a vector by a scalar and add a second scalar.

D(mL) = (A(mI)*B)+C for m=0,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

12.7.3.59 QVSMUL - Multiply the elements of a vector by a scalar.

C(mK) = A(mI) * B for m = 0, 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

12.7.3.60 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

12.7.3.61 QVSQRT - Take the square root of the elements of a vector.

C(mK) = SQRT (A(mI)) for m = 0,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

12.7.3.62 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

12.7.3.63 QVSWAP - Swap the elements of a vector.

A(mI) = C(mK) and C(mK) = A(mI) for m = 0, 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

12.7.3.64 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 extention files and to describe the format design for these files. The next section describe 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. 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 look up 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 of AIPS tables. Hopefully there will be may 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.

- TABCOP copies the entire contents of one or more tables of a given type.

- TABKEY reads or writes keyword value pairs to a table header.

- 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/initilize 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 initilization 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. These routines are briefly described in the following list; details of the call sequences are given at the end of this chapter.

- CCINI creates/initilizes CC (CLEAN component or gaussian model files).

- CHNDAT reads/writes/creates the contents of CH (IF descriptor) tables.

- FLGINI and TABFLG access FM (Flag) tables.

- GAINI and TABGA access GA (Gain) tables.

- NDXINI and TABNDX access NX (Index) tables.

- 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 refered to as "Row data". Associated with each column is label information; this included a label and data type for each column and a format to use if the file needs to be converted into a character file. 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 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 (512 bytes) 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	R8	1	-
single precision floating	R4	2	-
character (4 / floating)	R4	3	+ 10 * 1
long integer	I4	4	-
logical	L2	5	-
integer	I2	6	-
bit (NBITWD / integer)	I2	7	+ 10 * 1
select flag	I2	9	-

Declarations:

```
INTEGER*2  I2(*)
INTEGER*4  I4(*)
LOGICAL*2  L2(*)
REAL*4     R4(*)
REAL*8     R8(*)
EQUIVALENCE (I2, I4, L2, R4, R8)
```

The ordering is chosen to allow some machines to preprocess the LOGICAL*2 statement into a LOGICAL*4 if needed. More esoteric preprocessing may be required on less standard machines.

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 info / 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 (6 R*4s, 4 chars/R*4) in physical column order
     m+1 - i    : Units (2 R*4s, 4 chars/R*4) 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 / (256 / (6 * NWDPFP))
       i = m + 1 +  NCOL / (256 / (2 * NWDPFP))
       k = i + 1 +  NKEY / (256 / (4 * NWDPFP))
       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 -  2   (I*4) Number 512-byte records now in file
  3 -  4   (I*4) Max number rows allowed in current file
  5 -  6   (I*4) Number rows (logical records) now in file
  7              Number of bytes/value (2 for TA files)
  8              # values/logical (# I*2s/row incl. select for TA)
  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         Physical file name (set on each TABINI call)
 29 - 31         Creation task name (2 chars / integer)
 32              Disk number
 33 - 38         Last write access date: ZDATE(33), ZTIME(36)
 39 - 41         Last write access task name (2 chars / integer)
 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 - 60         Reserved
***********
 61              Number of selection strings now in file
 62              Next available R*4 address for a selection string
 63              First R*4 address of selection string 1
 64              First R*4 address of selection string 2
 65              First R*4 address of selection string 3
 66              First R*4 address of selection string 4
 67              First R*4 address of selection string 5
 68              First R*4 address of selection string 6
 69              First R*4 address of selection string 7
 70              First R*4 address of selection string 8
********** for TABIO / TABINI use only **********
 71              IOP : 1 => read, 2 => writ
 72              Number I*2 words per logical record (incl. select)
 73 - 74  (I*4) Current table row physical record in BUFFER
 75 - 76  (I*4) Current table row logical record in BUFFER
 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         Table title (4 chars / real)
129 -256         lookup table as COLPTR(logical column) = phys column
```

13.4.4 Keyword/value Records

The keyword/value pairs are stored in 4 single precision floating locations, 256 / (4 * NWDPFP) per physical record. The keyword is an 8-character string stored as 4 characters per real. It is left justified and the first character must imply the data type used for the value. The value is stored left justified in the 3rd and 4th reals using as many integer words as needed (see table below).

The first character of the keyword must specify the type of the binary value as:

D	double precision floating point
F	single precision floating point
C	8-character string in 4 chars / real
J	long integer
L	logical
I	integer

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(1+NWDPFP)	2nd 4 chars of keyword
RECORD(1+2*NWDPFP)	value

where the value occupies the following number of integer words

type D	NWDPDP
F	NWDPFP
C	2 * NWDPFP
J	NWDPLI
L	NWDPLO
I	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 TABP, is used by the non-I/O portions of the table package. TABP(1,1) - TABP(128,1) contains the subscript of the appropriate array for the logical columns. TABP(1,2) - TABP(128,2) contains the data type for each logical column. The programmer must fill in TABP(1,2) - TABP(NCOL,2) before calling TABINI when TABINI is to create the table extension file. TABINI will return a complete set of TABP under all circumstances.

13.4.6 Fundamental Table Access Subroutines

There are a set of basic table handeling routines which apply to all tables files. The following list gives a short description; the details of the call sequences and useage are found at the end of this chapter.

- TABINI creates/opens/catalogues an AIPS table.
- TABIO does I/O to a tables file. Row data, keyword/value pairs and control information are passed through this subroutine.
- GETCOL returns the value and value type at a specified row and column from an open table.
- FNDCOL locates the logical column number for a column with a specified label.

13.5 ROUTINES

Following are the descriptions of the call sequences and useage notes for the routines discussed in this chapter.

13.5.1 CCINI

creates and/or opens for writing (and reading) a specified CC (components table) file.

```
      SUBROUTINE CCINI (LUN, NCOL, VOL, CNO, VER, CATBLK, BUF, IERR)
```

Inputs:	LUN	I*2	Logical unit number to use
	VOL	I*2	Disk number
	CNO	I*2	Catalog number
In/out:	NCOL	I*2	Number of columns: 3 or 7 are allowed.
	VER	I*2	Input: desired version number 0 -› new Output: that used
	CATBLK	I*2(256)	File catalog header block
Output:	BUF	I*2(768)	First 512 words required for later calls to TABIO
	IERR	I*2	Error codes from TABINI or TABIO

13.5.2 CHNDAT

creates and fills or reads CH (IF descriptor) extension tables.

```
      CHNDAT (OPCODE, BUFFER, DISK, CNO, VER, CATBLK, LUN,
     *   NIF, FOFF, ISBAND, IERR)
```

Inputs:		
OPCODE	R*4	Operation code: 'WRIT' = create/init for write or read 'READ' = open for read only
BUFFER(512)	I*2	I/O buffer and related storage, also defines file if open.
DISK	I*2	Disk to use.
CNO	I*2	Catalogue slot number
VER	I*2	CH file version
CATBLK(256)	I*2	Catalogue header block.
LUN	I*2	Logical unit number to use
Input/Output:		
NIF	I*2	Number of IFs.
FOFF(*)	R*8	Frequency offset in Hz from ref. freq. True = reference + offset.
ISBAND(*)	I*2	Sideband of each IF. -1 =› 0 video freq. is high freq. end 1 =› 0 video freq. is low freq. end
Output:		

IERR	I*2	Return error code, 0=›OK, else TABINI or TABIO error.

13.5.3 FLGINI

creates and initilizes FLAG (FM) extension tables.

```
FLGINI (OPCODE, BUFFER, DISK, CNO, VER, CATBLK, LUN,
*   IFMRNO, FMKOLS, IERR)
```

Inputs:

OPCODE	R*4	Operation code: 'WRIT' = create/init for write or read 'READ' = open for read only
BUFFER(512)	I*2	I/O buffer and related storage, also defines file if open.
DISK	I*2	Disk to use.
CNO	I*2	Catalogue slot number
VER	I*2	FM file version
CATBLK(256)	I*2	Catalogue header block.
LUN	I*2	Logical unit number to use

Output:

IFMRNO	I*4	Next scan number, start of the file if READ, the last+1 if WRITE
FMKOLS(12)	I*2	The column pointer array in order, ID. NO., SUBARRAY, ANT1, ANT2, BTIME, ETIME, BIF, EIF, BCHAN, ECHAL, PFLAGS, REASON
IERR	I*2	Return error code, 0=›OK, else TABINI or TABIO error.

13.5.4 FNDCOL

locates the logical column number(s) which are titled with specified strings.

```
FNDCOL (NKEY, KEYS, LKEY, LORDER, BUFFER, KOLS, IERR)
```

Inputs:	NKEY	I*2	Number columns to be found
	KEYS	R*4(LKEY,N)	Column titles to locate (4 chars/real)
	LKEY	I*2	Number R*4 words to check in each of KEYS (legal values 1 through 6)
	LORDER	L*2	T => logical order desired, else phys.
In/out:	BUFFER	I*2(›512)	TABINI/TABIO buffer/ header/ work area
Output:	KOLS	I*2(NKEY)	Logical column numbers: 0 => none, -1 => more than one (!)
	IERR	I*2	Error code: 0 => ok, 1 - 10 from ZFIO ›10 = 10 + # of failed columns

13.5.5 GAINI

creates and initilizes gain (GA) extension tables.

```
      GAINI (OPCODE, BUFFER, DISK, CNO, VER, CATBLK, LUN,
     *   IGARNO, GAKOLS, NUMANT, NUMPOL, NUMIF, NUMNOD, INLEVL, GMMOD,
     *   RANOD, DECNOD, IERR)
```

Inputs:

OPCODE	R*4	Operation code: 'WRIT' = create/init for write or read 'READ' = open for read only
BUFFER(512)	I*2	I/O buffer and related storage, also defines file if open.
DISK	I*2	Disk to use.
CNO	I*2	Catalogue slot number
VER	I*2	GA file version
CATBLK(256)	I*2	Catalogue header block.
LUN	I*2	Logical unit number to use

Input/output

NUMANT	I*2	Number of antennas
NUMPOL	I*2	Number of IFs per group
NUMIF	I*2	Number of IF groups
NUMNOD	I*2	Number of interpolation nodes. Will handle up to 25 interpolation nodes.
INLEVL	I*2	Number of gain levels, 0=Abs., higher=›diff. calibration also included.
GMMOD	R*4	Mean gain modulus
RANOD(*)	R*4	RA offset of interpolation nodes (deg.)
DECNOD(*)	R*4	Dec. offset of interpolation nodes (deg.)

Output:

IGARNO	I*4	Next scan number, start of the file if READ, the last+1 if WRITE
GAKOLS(32)	I*2	The column pointer array in order, TIME, TIME INT., SOURCE ID., ANTENNA NO., SUBARRAY, REF. ANT., CALIBRATION LEVEL, IF NUMBER, REAL1, IMAG1, IONIPH1, TSYS1, DELAY1, RATE1, TGRDEL1, TPHDEL1, TDGRDEL1, TDPHDEL1, WEIGHT1, SNR1, Following used if 2 polarizations per IF REAL2, IMAG2, IONIPH2, TSYS2, DELAY2, RATE2, TGRDEL2, TPHDEL2, TDGRDEL2, TDPHDEL2, WEIGHT2, SNR2,
IERR	I*2	Return error code, 0=›OK, else TABINI or TABIO error.

13.5.6 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*4	Table row number: n.b. I*4
	ICOL	I*2	Table column number
	DATP	I*2(256)	Pointer array returned by TABINI
In/out:	BUFFER	I*2(*)	Control area set up by TABINI, used in TABIO
Output:	RTYPE	I*2	Type of column: 1 -> R*8, 2 -> R*4, 4 -> I*4, 5 -> L*?, 6 -> I*2 3+10*L -> character length L unpacked 7+10*L -> bit array length L packed
	RESULT	???	Value of column: use R*8, R*4, I*4, I*2 equivalenced arrays
	SCRTCH	I*2(*)	Scratch large enough to hold a row
	IERR	I*2	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

13.5.7 INDXIN

initilizes index (NX) file, finds first scan selected. If there is no index file the first and last records are set to the first and last records of the data file.

```
      INDXIN (IERR)
```

Inputs from common /SELCAL/		
NSOUWD	I*2	Number of sources included or excluded; if 0 all sources are included.
DOSWNT	L*2	If .TRUE. then sources in SOUWAN are included If .FALSE. then excluded.
SOUWAN(30)	I*2	The source numbers of sources included or excluded.
TIMRNG(8)	R*4	Start day, hour, min, sec, end day, hour, min,sec. 0's => all
Output:		
IERR	I*2	Return code, 0=>OK, otherwise INDEX file exists but cannot be read.
Output to common /SELCAL/:		
INXRNO	I*4	Current INDEX file record number. If .LT. 0 then there is no index file.
FSTVIS	I*4	First visibility number of current scan.
LSTVIS	I*4	Last visibility number of current scan.

CURSOU	I*2	Current source number.
NXKOLS(6)	I*2	Pointer array for index records. In order: TIME, TIME INT, SOURCE I, SUBARRAY, START VIS, ENDVIS.

13.5.8 SOUINI

creates and initilizes source (SU) extension tables.

```
      SOUINI (OPCODE, BUFFER, DISK, CNO, VER, CATBLK, LUN,
     *   ISURNO, SUKOLS, IERR)
```

Inputs:

OPCODE	R*4	Operation code: 'WRIT' = create/init for write or read 'READ' = open for read only
BUFFER(512)	I*2	I/O buffer and related storage, also defines file if open.
DISK	I*2	Disk to use.
CNO	I*2	Catalogue slot number
VER	I*2	SU file version
CATBLK(256)	I*2	Catalogue header block.
LUN	I*2	Logical unit number to use

Output:

ISURNO	I*4	Next scan number, start of the file if READ, the last+1 if WRITE
SUKOLS(15)	I*2	The column pointer array in order, ID. NO., SOURCE, QUAL, CALCODE, IFLUX, QFLUX, UFLUX, VFLUX, FREQOFF, BANDWIDTH, RAEPO, DECEPO, EPOCH, RAAPP, DECAPP
IERR	I*2	Return error code, 0=›OK, else TABINI or TABIO error.

13.5.9 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	I*2	Extension table type (e.g. 'CC','AN')
INVER	I*2	Version number to copy, 0 => copy all.
OUTVER	I*2	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*2    LUN for old file
 LUNNEW      I*2    LUN for new file
 VOLOLD      I*2    Disk number for old file.
 VOLNEW      I*2    Disk number for new file.
 CNOOLD      I*2    Catalog slot number for old file
 CNONEW      I*2    Catalog slot number for new file
In/out:
 CATNEW(256)I*2     Catalog header for new file.
Output:
 BUFF1(256)  I*2    Work buffer
 BUFF2(256)  I*2    Work buffer - will have CATBLK of old file
 IRET        I*2    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
```

13.5.10 TABGA

does I/O to GAIN extention tables. Usually used after setup by GAINI.

```
      TABGA (OPCODE, BUFFER, IGARNO, GAKOLS, NUMPOL,
     *    TIME, TIMEI, SOURID, ANTNO, SUBA, REFAN, LEVNO, IFNO,
     *    CREAL, CIMAG, IONIPH, TSYS, DELAY, RATE, TGRDEL, TPHDEL,
     *    TDGDEL, TDPDEL, WEIGHT, SNR, IERR)

    Inputs:
     OPCODE         R*4 Operation code:
                        'READ' = read entry from table.
                        'WRIT' = write entry in table.
                        'CLOS' = close file, flush on write
     BUFFER(512)    I*2 I/O buffer and related storage, also defines file
                        if open. Should have been returned by TABINI or
                        GAINI.
     IGARNO         I*4 Next scan number to read or write.
     GAKOLS(32)     I*2 The column pointer array in order, TIME,
                        TIME INT., SOURCE ID., ANTENNA NO., SUBARRAY,
                        REF. ANT., CALIBRATION LEVEL,
                        IF NUMBER, REAL1, IMAG1, IONIPH1, TSYS1, DELAY1,
                        RATE1, TGRDEL1, TPHDEL1, TDGRDEL1, TDPHDEL1,
                        WEIGHT1, SNR1,
                        Following used if 2 polarizations per IF
                        REAL2, IMAG2, IONIPH2, TSYS2, DELAY2,
                        RATE2, TGRDEL2, TPHDEL2, TDGRDEL2, TDPHDEL2,
                        WEIGHT2, SNR2,
     NUMPOL         I*2 Number of polarizations per IF.
    Input/output: (written to or read from GAIN file)
     TIME           R*4 Center time of GAIN record (Days)
```

TIMEI	R*4	Time interval covered by record (days)
SOURID	I*2	Source ID as defined in the SOURCE table.
ANTNO	I*2	Antenna number.
SUBA	I*2	Subarray number.
IFNO	I*2	If pixel number.
CREAL(2)	R*4	Real part of the complex gain, one for each poln.
CIMAG(2)	R*4	Imag part of the complex gain, one for each poln.
IONIPH(2)	R*4	Ionispheric phase correction, one for each poln.
TSYS(2)	R*4	System temperature (K), one for each poln.
DELAY(2)	R*4	Residual group delay (sec), one for each poln.
RATE(2)	R*4	Residual fringe rate (Hz), one for each poln.
TGRDEL(2)	R*4	Total group delay (sec), one for each poln.
TPHDEL(2)	R*4	Total phase delay (sec), one for each poln.
TDGDEL(2)	R*4	Total time derivative of group delay (sec/sec), one for each poln.
TDPDEL(2)	R*4	Total time derivative of phase delay (sec/sec), one for each poln.
TRATE(2)	R*4	Total fringe rate (Hz), one for each poln.
WEIGHT(2)	R*4	Weight of solution, one for each poln.
SNR(2)	R*4	Signal to noise ratio from fit.
Output:		
IGARNO	I*4	Next GAIN number.
IERR	I*2	Error code, 0=›OK else TABIO error. Note: -1=› read but record deselected.

13.5.11 TABINI

creates/opens a table extension file. If a file is created, it is catalogued 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	R*4	'READ' only, 'WRIT' read or write
PTYPE	I*2	File physical type: 2 characters
VOL	I*2	Disk number
CNO	I*2	Primary file catalog number
VER	I*2	Version number: ‹= 0 highest on READ highest+1 on WRIT (i.e. create one) output: version number used
CATBLK	I*2(256)	Primary file catalog header record
LUN	I*2	Logical unit number to use
NKEY	I*2	Maximum number of keyword/value pairs input: used on create, checked on WRIT (‹= recorded); output: actual
NREC	I*2	Number rows for create/extend input: used on WRIT only.
NCOL	I*2	Number of logical columns (not incl select) input: used on create, checked on WRIT (0 => any); output: actual

DATP	I*2(128,2)	DATPTR, DATYPE: DATYPE input on create, output actual for both
NBUF	I*2	Number I*2 words in BUFFER
BUFFER	I*2(*)	I/O buffer (* >= 512 as needed)
IERR	I*2	Error codes: 0 => OK, -1 => OK, new file created, 1 => bad input, 2 => cannot find/open, 3 => I/O error 4 => create error

13.5.12 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. 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	R*4	'READ', 'CLOS', 'WRIT' write with row selected 'FLAG' write with row de-selected
IRCODE	I*2	Type of record: 0 => table row 1 => DATPTR/DATYPE record 2 => data selection string 3 => titles 4 => units 5 => keyword/value pairs
IRNO	I*4 (!)	Logical record number: IRCODE = 0 => row number IRCODE = 1 => ignored IRCODE = 2 => string number IRCODE = 3 => column number IRCODE = 4 => column number IRCODE = 5 => keyword number
Input/Output:		
RECORD	I*2(*)	Appropriate data (input or output): IRCODE = 0 => row IRCODE = 1 => DATP IRCODE = 2 => select string IRCODE = 3 => column title IRCODE = 4 => column units IRCODE = 5 => keyword/value
BUFFER	I*2(>=768)	I/O control,scratch buffer (in/out)
Output:		
IERR	I*2	Error code: 0 => ok -1 => row read, but it is flagged 1 file not open, 2 input error 3 I/O error 4 logical EOF

5 error in file expansion

13.5.13 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,
     *   IERR)
```

Inputs:		
OPCODE	R*4	Operation desired, 'READ', 'WRIT'
KEYWRD(2,*)	R*4	Keywords to read/write, 4 char. per word.
NUMKEY	I*2	Number of keywords to read/write.
BUFFER(*)	I*2	Buffer being use for table I/O.
Output/Inputs:		
LOCS(NUMKEY)	I*2	The word offset of first short 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*2	The array of keyword values; due to word alignment problems on some machines values longer than a short integer should be copied, eg. if the 5th keyword (XXX) is a R*4: IPOINT = LOCS(5) CALL COPY (NWDPFP, VALUES(IPOINT), XXX) Output on READ, input on WRIT
Output:		
IERR	I*2	Return code, 0=›OK, 1-10 =›TABIO error 19 =› unrecognized data type. 20 =› bad OPCODE 20+n =› n keywords not found on READ.

13.5.14 TABFLG

does I/O to FLAG (FM) extention tables. Usually used after setup by FLGINI.

```
      TABFLG (OPCODE, BUFFER, IFMRNO, FMKOLS, SOURID, SUBA,
     *   ANT1, ANT2, BTIME, ETIME, BIF, EIF, BCHAN, ECHAN, PFLAGS,
     *   REASON, IERR)
```

Inputs:

OPCODE R*4 Operation code:

```
                    'READ' = read entry from table.
                    'WRIT' = write entry in table (must have been
                    opened with 'WRIT'.
                    'CLOS' = close file, flush on write
 BUFFER(512)  I*2 I/O buffer and related storage, also defines file
                    if open. Should have been returned by FLGINI or
                    TABINI.
 IFMRNO       I*4 Next FLAG entry number to read or write.
 FMKOLS(12)   I*2 The column pointer array in order, ID. NO.,
                    SUBARRAY, ANT1, ANT2, BTIME, ETIME, BIF, EIF,
                    BCHAN, ECHAN, PFLAGS, REASON
Input/output: (written to or read from FLAG file)
 SOURID       I*2 Source ID as defined in the SOURCE table.
 SUBA         I*2 Subarray number.
 ANT1         I*2 First antenna number, 0=>all
 ANT2         I*2 Second antenna number, 0=>all
 ETIME        R*4 Start time of data to be flagged (Days)
 BTIME        R*4 End time of data to be flagged (Days)
 BIF          I*2 First IF number to flag. 0=>all
 EIF          I*2 Last IF number to flag. 0=>all higher than BIF
 BCHAN        I*2 First channel number to flag. 0=>all
 ECHAN        I*2 Last channel number to flag. 0=>all higher.
 PFLAGS(4)    L*2 Polarization flags, same order as in data.
                    .TRUE. => polarization flagged.
 REASON(6)    R*4 Reason for flagging, 24 char. at 4 char/R*4.
Output:
 IFMRNO       I*4 Next scan number.
 IERR         I*2 Error code, 0=>OK else TABIO error.
                    Note: -1=> read but record deselected.
```

13.5.15 TABNDX

does I/O to INDEX extention tables. Usually used after setup by NDXINI.

```
      TABNDX (OPCODE, BUFFER, INXRNO, NXKOLS, TIME, DTIME,
     *   IDSOUR, SUBARR, VSTART, VEND, IERR)

    Inputs:
     OPCODE       R*4 Operation code:
                    'READ' = read entry from table.
                    'WRIT' = write entry in table.
                    'CLOS' = close file, flush on write
     BUFFER(512)  I*2 I/O buffer and related storage, also defines file
                    if open. Should have been returned by NDXINI or
                    TABINI.
     INXRNO       I*4 Next scan number to read or write.
     NXKOLS(6)    I*2 The column pointer array in order, TIME,
                    TIME INTERVAL, SOURCE ID, SUBARRAY, START VIS,
                    END VIS.
```

```
Input/output: (written to or read from INDEX file)
 TIME          R*4 Start time of the scan (Days)
 DTIME         R*4 Duration of scan (Days)
 IDSOUR        I*2 Source ID as defined in then SOURCE table.
 SUBARR        I*2 Subarray number.
 VSTART        I*4 First visibility number in file.
 VEND          I*4 Last visibility number in file.
Output:
 INXRNO        I*4 Next scan number.
 IERR          I*2 Error code, 0=>OK else TABIO error.
                   Note: -1=> read but record deselected.
```

13.5.16 TABSOU

does I/O to source (SU) extention tables. Usually used after setup by SOUINI.

```
 TABSOU (OPCODE, BUFFER, ISURNO, SUKOLS, IDSOU, SOUNAM,
*   QUAL, CALCOD, FLUX, FREQO, BANDW, RAEPO, DECEPO, EPOCH,
*   RAAPP, DECAPP, IERR)

Inputs:
 OPCODE        R*4 Operation code:
                   'READ' = read entry from table.
                   'WRIT' = write entry in table.
                   'CLOS' = close file, flush on write
 BUFFER(512)   I*2 I/O buffer and related storage, also defines file
                   if open. Should have been returned by SOUINI or
                   TABINI.
 ISURNO        I*4 Next scan number to read or write.
 SUKOLS(15)    I*2 The column pointer array in order, ID. NO.,
                   SOURCE, QUAL, CALCODE, IFLUX, QFLUX, UFLUX,
                   VFLUX, FREQOFF, BANDWIDTH, RAEPO, DECEPO, EPOCH,
                   RAAPP, DECAPP
Input/output: (written to or read from INDEX file)
 IDSOUR        I*2 Source ID as defined in then SOURCE table.
 SOUNAM(4)     R*4 Source name (AIPS Packed string)
 QUAL          I*2 Source qualifier.
 CALCOD        R*4 Calibrator code 4 char.
 FLUX(4)       R*4 Total flux density I, Q, U, V pol, (Jy)
 FREQO         R*8 Frequency offset (Hz)
 BANDW         R*8 Bandwidth (Hz)
 RAEPO         R*8 Right ascension at mean EPOCH (degrees)
 DECEPO        R*8 Declination at mean EPOCH (degrees)
 EPOCH         R*8 Mean Epoch for position in yr. since year 0.0
 RAAPP         R*8 Apparent Right ascension (degrees)
 DECAPP        R*8 Apparent Declination(degrees)
Output:
 ISURNO        I*4 Next scan number.
 IERR          I*2 Error code, 0=>OK else TABIO error.
                   Note: -1=> read but record deselected.
```

13.5.17 TABSRT

sorts an AIPS table extention 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 (Slowest 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.

```
TABSRT (DISK, CNO, TYPE, INVER, OUTVER, KEY, FKEY,
*   BUFFER, BUFSZ, TABUFF, NBUF, CATBLK, IERR)
```

Inputs:

DISK	I*2	Disk number of the file.
CNO	I*2	Catalogue slot number.
TYPE	I*2	Two character type code (e.g. 'CC')
INVER	I*2	Input version number
OUTVER	I*2	Output version number
KEY(2,2)	I*2	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. Column no. is the physical no.
FKEY(2,2)	R*4	Key coefficients, 0=>1, see above.
BUFSZ	I*2	Size of BUFFER in bytes.
NBUF	I*2	Size of TABUFF in (I*2) words.
CATBLK(256)	I*2	Catalogue header record.

Output:

TABUFF(*)	I*2	Buffer large enough to handle I/O to table.
BUFFER(*)	R*4	I/O work buffer
IERR	I*2	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. Includes DFIL.INC and CFIL.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 initilized.

IF a disk based sort is required, then a 4-way merge sort will be used; the FTAB declaration in the main program and the call to ZDCHIN should be large enough to handle 8 map-like files and 2 Non-map files at the same time. (Additional files may be required if they are left open by the calling program).

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.

CHAPTER 14

FITS TAPES

14.1 OVERVIEW

The principle route for getting data and images into and out of AIPS is by FITS (Flexible Image Transport System) format tape 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 advantage that it is a self-defining format and the actual bit pattern on the tape is independent of the machine on which the tape was written. The purpose of this chapter is to describe the general features of FITS and the details of the AIPS implementation of FITS. This chapter is not intended to be a rigorous description of the FITS standards.

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 WGLAS.

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 catalogue 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 auxilary 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 PHILOSPHY

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 standarized 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 dimension and 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 be fairly straight forward on any computer used for astronomical image processing. Simple also implies that the structure of the file 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 obselete 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 multidimensional 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 tapes are 2880 8-bit bytes long with one record per tape block. (Larger blocking factors are being considered but have not yet been implemented.) 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 by skipping header and trailer records.

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 accomodate more primitive systems, a fixed format is mandatory for the required keywords and suggested for the optional keywords. This fixed format is as follows:

- Keyword name beginning in column 1.
- "=" in column 9
- T or F (logical true or false) in column 30.
- Real part (integer or floating) right justified, ending in column 30.
- Imaginary part (integer or floating) right justified, ending in column 50.
- 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.

14.3.2.1 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.

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.

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.

- BSCALE (floating) is the scale factor used to convert tape pixel values to true values (true = [tape BSCALE] + BZERO).

- BZERO (floating) is the offset applied to true pixel values (see BSCALE).

- BUNIT (character) gives the brightness units.

- BLANK (integer) is the tape pixel value assigned to undefined pixels.

- OBJECT (character) is the image name.

- DATE (character) is the date the file was written ('dd/mm/yy')

- DATE-OBS (character) is the date of data acquition ('dd/mm/yy').

- ORIGIN (character)is the tape writing institution.
- INSTRUME (character) is the data acquisition instrument.
- TELESCOP (character) is the data acquisition telescope.
- OBSERVER (character) is the observer name / identification.
- blank in col 1-8 (none) means columns 9 - 80 are a comment.
- COMMENT (none) means columns 9 - 80 are a comment.
- HISTORY (none) means columns 9 - 80 are a comment.
- CRVALn (floating) is the value of physical coordinate on axis n at the reference pixel.
- 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.
- CDELTn (floating) is the increment in physical coordinate along axis n as FORTRAN counter increases by 1.
- CTYPEn (character) is the type of physical coordinate on axis n.
- CROTAn (floating) is the rotation angle of actual axis n from stated coordinate type.
- DATAMAX (floating) is the maximum data value in file (after scaling).
- DATAMIN (floating) is the minimum data value in file.
- EPOCH (floating) is the epoch of coordinate system (years).

Of these keywords, all are well defined except the rotation; see the chapter on the catalogue 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.

14.3.2.2 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.

14.3.2.3 AIPS Nonstandard Image File Keywords - There are a number of keywords used by AIPS which are not standard.

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

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

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

- OPHRAE11 (floating) an obscure number related to the Right ascension of the center on an image made on the VLA pipeline PDP11.
- OPHDCE11(floating) an obscure number related to the declination of the center on an image made on the VLA pipeline PDP11.
- 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.

14.3.2.4 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 catalogue 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 catalogue header and AIPS memo number 27. 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 catalogue header uses the same convention, the same problems occur internally to AIPS. See the chapter on the AIPS catalogue header for a brief discussion of the conventions used in AIPS. The same conventions are used when reading and writing FITS tapes.

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

```
000000000111111111122222222223333333333444444444455555555556666666666
123456789012345678901234567890123456789012345678901234567890123456789
SIMPLE  =                    T /
BITPIX  =                   16 /
NAXIS   =                    4 /
NAXIS1  =                 2048 /
NAXIS2  =                 1024 /
NAXIS3  =                    1 /
NAXIS4  =                    1 /
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
```

14.3.2.6 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 small integer array.

4. ZI32IL extracts 32-bit twos complement integers from a buffer and puts them in a local array of pseudo I*4 integers.

5. ZI8L8 converts 8-bit unsigned binary numbers to "bytes" (half of a local small integer).

6. ZILI16 converts a buffer of local small integers to a buffer of standard 16-bit, twos complement integers. ZR8P4 converts between pseudo I*4 and double precision (R*8).

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 preceeded 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 structure of a group is shown in the following.

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

where r1 ... rk are random parameters 1 thru k
p11 ... pmn are the pixel value 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.

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):

- 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.
- PSCALn (floating) gives the scale factor for random parameter n. True_value = tape_value * PSCALn + PZEROn
- 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'.

- SORT ORDER (character) the order of the groups.
- 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).

14.4.2.1 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 weight. A non positive weight indicates that the the visibility has been flagged. The scaling desired for the weight may be different 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 is in this format. The weights and flags are passed as random parameters. More on this later in the broadcast.

14.4.2.2 Antennas And Subarrays - If data from different arrays (or different VLA configurations) are combined, the physical identity of a given antenna 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.

14.4.2.3 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). See the uv data section of the disk I/O chapter for more details of the AIPS internal uv data coordinate systems.

14.4.2.4 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 slowest. The two keys are coded as characters given by the following table:

B => baseline number
T => time order
U => u spatial freq. coordinate
V => v spatial freq. coordinate
W => w spatial freq. coordinate
R => baseline length.
P => baseline position angle.
X => descending ABS(u)
Y => descending ABS(v)
Z => ascending ABS(u)
M => ascending ABS(v)
* => not sorted

14.4.3 Typical VLA Record Structure

The following is a uv FITS header for continuum VLA data which demonstrates the use of multiple precision random parameters. Most of the HISTORY records are removed from this example. The header indicates that the data in this example is followed by two antenna files in the old AIPS tables format.

```
000000000111111111122222222223333333333444444444455555555556666666666
123456789012345678901234567890123456789012345678901234567890123456789
SIMPLE  =                    T /
BITPIX  =                   16 /
NAXIS   =                    6 /
NAXIS1  =                    0 /NO STANDARD IMAGE JUST GROUP
NAXIS2  =                    3 /
NAXIS3  =                    4 /
NAXIS4  =                    1 /
NAXIS5  =                    1 /
NAXIS6  =                    1 /
OBJECT  = '0923+350'           / SOURCE NAME
TELESCOP= '        '           /
INSTRUME= '        '           /
OBSERVER= 'COTT    '           /
DATE-OBS= '30/04/82'           /OBSERVATION START DATE DD/MM/YY
DATE-MAP= '11/10/83'           /DATE OF LAST PROCESSING DD/MM/YY
BSCALE  =    3.30987595420E-06 /REAL = TAPE * BSCALE + BZERO
BZERO   =    0.00000000000E+00 /
BUNIT   = 'JY      '           /UNITS OF FLUX
EPOCH   =      1.950000000E+03 /EPOCH OF RA DEC
OBSRA   =    1.40795415491E+02 /ANTENNA POINTING RA
OBSDEC  =    3.50133331865E+01 /ANTENNA POINTING DEC
TABLES  =                    2 /THIS IS THE ANTENNA FILE
CTYPE2  = 'COMPLEX '           /
CRVAL2  =    1.00000000000E+00 /
CDELT2  =      1.000000000E+00 /
CRPIX2  =      1.000000000E+00 /
CROTA2  =      0.000000000E+00 /
CTYPE3  = 'STOKES  '           /
CRVAL3  =   -1.00000000000E+00 / STOKES AS RR; LL, RL, LR
CDELT3  =     -1.000000000E+00 /
CRPIX3  =      1.000000000E+00 /
CROTA3  =      0.000000000E+00 /
CTYPE4  = 'FREQ    '           /
CRVAL4  =    4.88510000000E+09 /
CDELT4  =      5.000000000E+07 /
CRPIX4  =      1.000000000E+00 /
CROTA4  =      0.000000000E+00 /
CTYPE5  = 'RA      '           /
CRVAL5  =    1.40795415491E+02 /
CDELT5  =      0.000000000E+00 /
CRPIX5  =      1.000000000E+00 /
CROTA5  =      0.000000000E+00 /
CTYPE6  = 'DEC     '           /
CRVAL6  =    3.50133331865E+01 /
CDELT6  =      0.000000000E+00 /
CRPIX6  =      1.000000000E+00 /
CROTA6  =      0.000000000E+00 /
GROUPS  =                    T /
GCOUNT  =                21389. /
PCOUNT  =                    7 /
PTYPE1  = 'UU-L    '           /
```

```
PSCAL1  =    2.56659543954E-09 /
PZERO1  =    0.00000000000E+00 /
PTYPE2  = 'VV-L    '           /
PSCAL2  =    3.46332811989E-09 /
PZERO2  =    0.00000000000E+00 /
PTYPE3  = 'WW-L    '           /
PSCAL3  =    2.33722136998E-09 /
PZERO3  =    0.00000000000E+00 /
PTYPE4  = 'BASELINE'           /
PSCAL4  =    1.00000000000E+00 /
PZERO4  =    0.00000000000E+00 /
PTYPE5  = 'BASELINE'           /
PSCAL5  =    1.00000000000E-02 /
PZERO5  =    0.00000000000E+00 /
PTYPE5  = 'DATE    '           /
PSCAL5  =    2.50000000000E-01 /
PZERO5  =    2.44508950000E+06 /
PTYPE7  = 'DATE    '           /
PSCAL7  =    1.52587890600E-05 /
PZERO7  =    0.00000000000E+00 /
         / WHERE BASELINE = 256*ANT1 + ANT2 + (ARRAY#-1)/100
HISTORY UVLOD RELEASE='15NOV83 ' /CREATED AT 11-OCT-1983 13:34:50
HISTORY UVLOD OUTNAME='0923+350  ' OUTCLASS='UVDATA'
HISTORY UVLOD OUTSEQ=    1   OUTDISK=  3
        ...
ORIGIN  = 'AIPSNRAO node CVAX   15NOV83'  /
DATE    = '11/10/83'                      / TAPE WRITTEN ON DD/MM/YY
HISTORY AIPS   IMNAME='0923+350   ' IMCLASS='XYAC  ' IMSEQ=   1   /
HISTORY AIPS   USERNO=   413           /
HISTORY AIPS   SORT ORDER = 'XY'
              / WHERE X MEANS DESC ABS(U)
              / WHERE Y MEANS DESC ABS(V)
HISTORY AIPS WTSCAL = 2.76756756757E+01 / CMPLX WTS=WTSCAL*(TAPE*BSCALE+BZE
END
```

14.5 EXTENSION FILES

There is frequently auxilary 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. 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 tape file which contains information of relevence 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 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:

- 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 last 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.

- 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 hierarchical structure giving its relation to other files (e.g. "map1.cleancomp")

- EXTVER (integer) is a version number which can be used with EXTNAME to identify a file.

- EXTLEVEL (integer) specifies the level of the extension file in a hierarchical structure. The default value for EXTLEVEL should be 1.

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 exercized 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), 6) the units (optional).

14.5.2.1 Tables Header Record - The keywords for tables extension file headers are given in the following:

- XTENSION (character) is required to be the first keyword and has a value 'TABLE ' for table extension files.
- BITPIX (integer) is a required keyword which must have a value of 8 indicating printable ASCII characters.
- NAXIS (integer) is a required keyword which must have a value of 2 for tables extension files.
- NAXIS1 (integer) is a required keyword which given the number of characters in a table entry.
- NAXIS2 (integer) is a required keyword which gives the number of entries in the table. A value of 1 is allowed.
- PCOUNT (integer) is a required keyword which must have the value of 0 for tables extension files.

- GCOUNT (integer) is a required keyword which must have the value of 1 for tables extension files.

- TFIELDS (integer) is a required keyword which must follow the GCOUNT keyword. TFIELDS gives the number of fields in each table entry.

- EXTNAME (character) is the name of the table.

- EXTVER (integer) is the version number of the table.

- EXTLEVEL (integer) is the hierarchical level number of the table, 1 is recommended. (optional)

- TBCOLnnn (integer) the pixel number of the first character in the nnn th field .

- TFORMnnn (character) the Fortran format of field nnn (I,A,E,D)

- TTYPEnnnn (character) the label for field nnn. (optional, order arbitrary)

- TUNITnnn (character) the physical units of field nnn. (optional, order arbitrary)

- 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)

- TZEROnnn (floating) the offset for field nnn. (See TSCALnnn.) Default value is 0.0 (optional, order arbitrary)

- 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)

- AUTHOR (character) the name of the author or creator of the table. (optional, order arbitrary)

- REFERENC (character) the reference for the table. (optional, order arbitrary)

- 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.

The FITS tables agreement does not include a number of data types used with AIPS tables: Short integer, logical and bit string. The following conventions are used to indicate these data types.

1. Extention name. The table name is given by the table header keyword 'EXTNAME'. AIPS will invoke the following conventions if the EXTNAME keyword is of the form 'AIPS xx' where xx is a two character table type. Likewise, tables files written by AIPS will use the same convention.

2. Integer length An integer will be considered to be short (e.g. 16 bits) if the width of the field given by the TFORMn keyword is 6 or less and long if the width is 7 or more.

3. Logical Logical variables will be written as a character string of length 1 containing either the value 'T' or 'F' indicating true or false. The units of this field will be 'LOGICAL'.

4. Bit strings Bit strings will be encoded as hexidecimal numbers in a character string. The first set of four bits will be encoded into the first character with the first bit being the most signifigant in determining the hexidecimal representation. The following bits come in the same order. The units of this field will be 'HEXnnnn' where nnnn gives the number of bits in the string.

Arbitrary Table keywords do not have a defined data type in FITS. FITS written by AIPS tapes will use the AIPS tables convention for keywords not defined by the FITS agreement and will indicate the data type by the first letter of the keyword name:

First letter	data type
F	Single precision floating
D	Double precision floating
C	Character string (8 char)
I	Short integer
J	Long integer
L	Logical

14.5.2.2 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.

14.5.2.3 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
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456
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 Older AIPS Tables

Prior to the (presumed) establishment of the standard tables extension files, AIPS had it 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.

14.5.3.1 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.

14.5.3.2 Data Records - The data records consist of floating point 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.

14.5.3.3 CC Files - The details of the AIPS old 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
```

14.5.3.4 AN Files - The details of the AIPS old 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 catalogue header. Each of the sets consists of a declaration include (Dnnn.inc), an EQUIVALENCE include (Ennn.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:

- AWORD (R*4) - this array contains the recognized keywords, two R*4 words per keyword with four characters per R*4 word. This array can be sent to GETCRD as the list of keywords.

- NCT (I*2) - this gives the number of required keyword names in CWORD which is equivalences at the beginning of AWORD.

- NKT (I*2) - this given the number of optional keywords names in KWORD which is equivalenced into AWORD after CWORD.

- ATYPE (I*2) - this array gives the data types corresponding to keywords in AWORD. 1=›logical variable, 2=›numerical value, and 3=›string.

- APOINT (I*2) - this array contains pointers in the common in the includes DHDR.INC and CHDR.INC to the AIPS catalogue header in the form 1000nbytes + 100offset + position of pointer in common. Here nbytes given the number of bytes used in the AIPS catalogue header 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
      INTEGER*2 ATYPE(150), APOINT(150), CTYPE(11), KTYPE(139),
     *   CPOINT(11), POINT(139), NKT, NCT
      REAL*4    AWORD(2,150), CWORD(2,11), KWORD(2,139), K1(2,73),
     *   K2(2,66)
C                                                               End DFUV
```

14.6.2 DFIT.INC

```
C                                                               Include DFUV
      INTEGER*2 ATYPE(82), APOINT(82), CTYPE(10), KTYPE(72), CPOINT(10),
     *   POINT(72), NKT, NCT
      REAL*4    AWORD(2,82), CWORD(2,10), KWORD(2,72)
C                                                               End DFUV
```

14.6.3 EFUV.INC

```
C                                                               Include EFUV
      EQUIVALENCE (AWORD(1,1), CWORD(1,1)),  (AWORD(1,85), K2(1,1)),
     *   (AWORD(1,12), KWORD(1,1), K1(1,1))
      EQUIVALENCE (APOINT(1), CPOINT(1)),    (APOINT(12), POINT(1))
      EQUIVALENCE (ATYPE(1), CTYPE(1)),      (ATYPE(12), KTYPE(1))
C                                                               End EFUV
```

14.6.4 EFIT.INC

```
C                                                               Include EFUV
      EQUIVALENCE (AWORD(1,1), CWORD(1,1)), (AWORD(1,11), KWORD(1,1))
      EQUIVALENCE (APOINT(1), CPOINT(1)),    (APOINT(11), POINT(1))
      EQUIVALENCE (ATYPE(1), CTYPE(1)),      (ATYPE(11), KTYPE(1))
C                                                               End EFUV
```

14.6.5 VFUV.INC

```
C                                                               Include VFUV
      DATA NCT/11/,     NKT/139/
      DATA CWORD/'SIMP','LE  ','BITP','IX  ','NAXI','S   ','NAXI',
     *  'S1  ','NAXI','S2  ','NAXI','S3  ','NAXI','S4  ','NAXI','S5  ',
     *         'NAXI','S6  ','NAXI','S7  ','NAXI','S8  '/
      DATA K1 /'OBJE','CT  ','TELE','SCOP','INST','RUME','OBSE','RVER',
     *         'DATE','-OBS','DATE','-MAP','BSCA','LE  ','BZER','O   ',
     *         'BUNI','T   ','CTYP','E1  ','CTYP','E2  ','CTYP','E3  ',
     *         'CTYP','E4  ','CTYP','E5  ','CTYP','E6  ','CTYP','E7  ',
     *         'CTYP','E8  ','CRVA','L1  ','CRVA','L2  ','CRVA','L3  ',
     *         'CRVA','L4  ','CRVA','L5  ','CRVA','L6  ','CRVA','L7  ',
     *         'CRVA','L8  ','CDEL','T1  ','CDEL','T2  ','CDEL','T3  ',
     *         'CDEL','T4  ','CDEL','T5  ','CDEL','T6  ','CDEL','T7  ',
     *         'CDEL','T8  ','CRPI','X1  ','CRPI','X2  ','CRPI','X3  ',
```

```
     *          'CRPI','X4  ','CRPI','X5  ','CRPI','X6  ','CRPI','X7  ',
     *          'CRPI','X8  ','CROT','A1  ','CROT','A2  ','CROT','A3  ',
     *          'CROT','A4  ','CROT','A5  ','CROT','A6  ','CROT','A7  ',
     *          'CROT','A8  ','EPOC','H   ','DATA','MAX ','DATA','MIN ',
     *          'BLAN','K   ','INHI','BIT ','IMNA','ME  ','IMCL','ASS ',
     *          'IMSE','Q   ','USER','NO  ','PROD','UCT ','NITE','R   ',
     *          'BMAJ','    ','BMIN','    ','BPA ','    ','VELR','EF  ',
     *          'ALTR','VAL ','ALTR','PIX ','OBSR','A   ','OBSD','EC  ',
     *          'REST','FREQ','XSHI','FT  ','YSHI','FT  ','DATE','    ',
     *          'ORIG','IN  '/
      DATA K2 /'GROU','PS  ','GCOU','NT  ','PCOU','NT  ','PTYP','E1  ',
     *          'PTYP','E2  ','PTYP','E3  ','PTYP','E4  ','PTYP','E5  ',
     *          'PTYP','E6  ','PTYP','E7  ','PTYP','E8  ','PTYP','E9  ',
     *          'PTYP','E10 ','PTYP','E11 ','PTYP','E12 ','PTYP','E13 ',
     *          'PTYP','E14 ','PTYP','E15 ','PTYP','E16 ','PTYP','E17 ',
     *          'PTYP','E18 ','PTYP','E19 ','PTYP','E20 ','PSCA','L1  ',
     *          'PSCA','L2  ','PSCA','L3  ','PSCA','L4  ','PSCA','L5  ',
     *          'PSCA','L6  ','PSCA','L7  ','PSCA','L8  ','PSCA','L9  ',
     *          'PSCA','L10 ','PSCA','L11 ','PSCA','L12 ','PSCA','L13 ',
     *          'PSCA','L14 ','PSCA','L15 ','PSCA','L16 ','PSCA','L17 ',
     *          'PSCA','L18 ','PSCA','L19 ','PSCA','L20 ','PZER','O1  ',
     *          'PZER','O2  ','PZER','O3  ','PZER','O4  ','PZER','O5  ',
     *          'PZER','O6  ','PZER','O7  ','PZER','O8  ','PZER','O9  ',
     *          'PZER','O10 ','PZER','O11 ','PZER','O12 ','PZER','O13 ',
     *          'PZER','O14 ','PZER','O15 ','PZER','O16 ','PZER','O17 ',
     *          'PZER','O18 ','PZER','O19 ','PZER','O20 ','TABL','ES  ',
     *          'SORT','ORDR','WTSC','AL  '/
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, 2,2,3,3, 2,2,2,2,
     *            2,2,2,2, 2,2,2,2, 2,2,2,3, 3,1,2,2,
     *            20*3, 20*2, 20*2, 2,3,2/
C                                          1000*nbytes + 100*offset +
C                                          position of pointer in common
      DATA CPOINT /   0, 2043, 2041, 2042, 2142, 2242, 2342, 2442,
     *             2542, 2642, 2742/
      DATA POINT / 8001, 8002, 8003, 8004, 8005, 8006, 8029, 8030,
     *             8007, 8009, 8109, 8209, 8309, 8409, 8509, 8609,
     *             8709, 8031, 8131, 8231, 8331, 8431, 8531, 8631,
     *             8731, 4010, 4110, 4210, 4310, 4410, 4510, 4610,
     *             4710, 4011, 4111, 4211, 4311, 4411, 4511, 4611,
     *             4711, 4012, 4112, 4212, 4312, 4412, 4512, 4612,
     *             4712, 4013, 4014, 4015, 4016, 2044,12017, 6218,
     *             2045, 2046, 2048, 2047, 4020, 4021, 4022, 2049,
     *             8035, 4023, 8032, 8033, 8034, 4024, 4025,    0,
     *                0, 1001, 2039, 2040,
     *            20*8008, 20*4004, 20*4004, 4004, 2048, 4004/
C                                                          End VFUV.
```

14.6.6 VFIT.INC

```
C                                                                Include VFIT
      DATA NCT/10/,     NKT/72/
      DATA CWORD/'SIMP','LE  ','BITP','IX  ','NAXI','S   ','NAXI',
     *  'S1  ','NAXI','S2  ','NAXI','S3  ','NAXI','S4  ','NAXI','S5  ',
     *         'NAXI','S6  ','NAXI','S7  '/
      DATA KWORD /'OBJE','CT  ','TELE','SCOP','INST','RUME','OBSE',
     *  'RVER','DATE','-OBS','DATE','-MAP','BSCA','LE  ','BZER','0   ',
     *         'BUNI','T   ','CTYP','E1  ','CTYP','E2  ','CTYP','E3  ',
     *         'CTYP','E4  ','CTYP','E5  ','CTYP','E6  ','CTYP','E7  ',
     *         'CRVA','L1  ','CRVA','L2  ','CRVA','L3  ','CRVA','L4  ',
     *         'CRVA','L5  ','CRVA','L6  ','CRVA','L7  ','CDEL','T1  ',
     *         'CDEL','T2  ','CDEL','T3  ','CDEL','T4  ','CDEL','T5  ',
     *         'CDEL','T6  ','CDEL','T7  ','CRPI','X1  ','CRPI','X2  ',
     *         'CRPI','X3  ','CRPI','X4  ','CRPI','X5  ','CRPI','X6  ',
     *         'CRPI','X7  ','CROT','A1  ','CROT','A2  ','CROT','A3  ',
     *         'CROT','A4  ','CROT','A5  ','CROT','A6  ','CROT','A7  ',
     *         'EPOC','H   ','DATA','MAX ','DATA','MIN ','BLAN','K   ',
     *         'INHI','BIT ','IMNA','ME  ','IMCL','ASS ','IMSE','Q   ',
     *         'USER','NO  ','PROD','UCT ','NITE','R   ','BMAJ','    ',
     *         'BMIN','    ','BPA ','    ','VELR','EF  ','ALTR','VAL ',
     *         'ALTR','PIX ','OBSR','A   ','OBSD','EC  ','REST','FREQ',
     *         'XSHI','FT  ','YSHI','FT  ','DATE','    ','ORIG','IN  ',
     *         'TABL','ES  ','OPHR','AE11','OPHD','CE11','MAPN','AM11'/
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, 2,3,3,2, 2,2,2,2, 2,2,2,2,
     *            2,2,2,2, 2,2,3,3, 2,2,2,3/
C                                      1000*nbytes + 100*offset +
C                                      position of pointer in common
      DATA CPOINT /   0, 2043, 2041, 2042, 2142, 2242, 2342, 2442,
     *             2542, 2642/
      DATA POINT / 8001, 8002, 8003, 8004, 8005, 8006, 8029, 8030,
     *             8007, 8009, 8109, 8209, 8309, 8409, 8509, 8609,
     *             8031, 8131, 8231, 8331, 8431, 8531, 8631, 4010,
     *             4110, 4210, 4310, 4410, 4510, 4610, 4011, 4111,
     *             4211, 4311, 4411, 4511, 4611, 4012, 4112, 4212,
     *             4312, 4412, 4512, 4612, 4013, 4014, 4015, 4016,
     *             2044,12017, 6218, 2045, 2046, 2048, 2047, 4020,
     *             4021, 4022, 2049, 8035, 4023, 8032, 8033, 8034,
     *             4024, 4025,    0,    0, 4001, 4101, 4201,12017/
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.

- FPARSE parses a FITS header card, unpacking the card image, interpreting it and putting the data value into the correct location in the AIPS catalogue header. This routine is for standard FITS headers but with the substitution of the INCLUDEs DFIT.INC, EFIT.INC and VFIT.INC for DFUV.INC, EFUV.INC and VFUV.INC the routine will work for FITS image tapes written on the VLA pipeline.

- GETCRD unpacks a given card image from a header block of FITS data and looks for keywords in a supplied table.

- GETSYM finds the next symbol in an unpacked buffer. A symbol is defined to begin with a letter and have up to 8 alpha-numeric characters.

- GETLOG obtains the value of a logical variable from an unpacked buffer.

- GETNUM converts an ASCII numeric field into a REAL*8 value.

- GETSTR obtains a character string from an unpacked buffer.

Following are the details of the call sequence and function of the AIPS FITS parsing utility routines.

14.7.1 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. Works for standard uv or image FITS headers.

```
      FPARSE (ICARD, FITBLK, PSCAL, POFF, PTYPES, TABLES,
     *   END, IERR)
```

Inputs:			
	ICARD	I*2	The card number (1-36) in block to interpret.
	FITBLK	I*2(1440)	A block of FITS header data.
Outputs:			
	PSCAL	R*8(20)	Random parameter scalings
	POFF	R*8(20)	Random parameter offsets
	PTYPES	R*4(20)	Random parameter types (packed chars every other one)
	TABLES	I*2	# Tables extension
	END	L*2	True if end card found, else false.
	IERR	I*2	error code 0=ok. 1=error.
COMMON /MAPHDR/			

14.7.2 GETCRD - (get card) will unpack 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.

```
      GETCRD (ICARD, NOSYM, STRSYM, SYMTAB, FITBLK, NPNT,
     *   KL, SYMBOL, TABNO, ISHIST, END, IERR)
```

Inputs:			
	ICARD	I*2	the card image (1-36) in FITS data block.
	NOSYM	I*2	the number of entries in key word table.
	STRSYM	I*2	Start search with symbol # STRSYM
	SYMTAB	I*2(2,NOSYM)	unpacked keywords, two per I*2.
	FITBLK	I*2(1440)	the block of FITS header cards.
In/Out:			
	NPNT	I*2	The position to start scan in array KL. Returns the last position scanned plus one.
	KL	I*2(80)	input the unpacked card image if NPNT > 1, else returns the unpacked card image.
Outputs:			
	SYMBOL	I*2(2)	the unpacked symbol found on the card.
	TABNO	I*2	SYMBOL matches SYMTAB(1&2,TABNO).
	ISHIST	L*2	True if history card else false.
	END	L*2	True if end card found, else false.
	IERR	I*2	0=match found, 1=no match on otherwise valid keyword, 2=card ends or other trouble

14.7.3 GETLOG - obtains the value of a logical variable from loose buffer.

```
      GETLOG (KB, LIMIT, KBP, IL)
```

Inputs:			
	KB(80)	I*2	Loose buffer of card image
	LIMIT	I*2	Number of words in loose buffer
	KBP	I*2	Pointer position at start
Outputs:			
	KBP	I*2	Pointer position of next field
	IL	I*2	Value of logical field
			0--> .false.
			1--> .true.
			2--> invalid

14.7.4 GETNUM - converts ASCII numeric field into REAL*8 number.

```
GETNUM (KB, KBPLIM, KBP, X)

Inputs:  KB    I*2( )  loose character buffer
         KBPLIM I*2    # characters in buffer
         KBP   I*2     start of numeric field
Outputs: KBP   I*2     start of next field (incl blanks)
         X     R*8     numerical value
```

14.7.5 GETSTR - obtains a hollerith value from a loose buffer.

```
GETSTR (KB, KBPLIM, NMAX, KBP, ISTR, NCHAR)

Inputs:  KB     I*2(80)   loose buffer
         KBPLIM I*2       size of loose buffer
         NMAX   I*2       max string length in characters
         KBP    I*2       start position in KB
Outputs: KBP    I*2       start position in KB next field
         ISTR   R*4(*)    packed string, blank filled
         NCHAR  I*2       # characters (0 => no string found)
```

14.7.6 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 tape and for interpreting the HI files.

```
GETSYM (LBUFF, NPNT, SYM, IERR)

Inputs:
   LBUFF(80)   I*2   Loose packed card image
   NPNT        I*2   Pointer to first character
Output:
   NPNT        I*2   Pointer value after getting symbol
   SYM(2)      R*4   Symbol, padded with blanks
   IERR        I*2   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
```

14.8 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, Grosbol, Tritton, Greisen and Wells 1984, draft reproduced in the IAU Comission 9 Astronomical Image Processing Circular.

CHAPTER 15

THE Z ROUTINES

15.1 OVERVIEW

The AIPS system has a number of types of routines the details of which depend on the hardware and/or operating system upon which the system is running. These types of routines are denoted by the first letter of the name. The types of routines which may vary from system to system are: 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.

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 applications code runs. Careful study of an existing implementation of AIPS is recommended before attempting a new installation.

NOTE: The routines described in this chapter are intended to constitute a complete and necessary set to implement ALL AIPS applications code exclusive of the image display (TV) related functions. Any (non-TV) "Z" routines not described in this chapter are lower level routines and 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. Data Manipulation Routines - These routines convert data formats from external (tape) integers and characters to local and vice versa, and move bits and bytes.

2. Disk I/O and File Manipulation - These routines create, destroy, expand, contract, open, close, read, and write disk files.

3. System Functions - These routines do various system functions such as starting and stoping processes, inquiring what processes are running, and inquiring how much space is available on a given disk drive.

4. 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.

5. Directory and Text File Routines - These routines read the directories for, and contents of, text files.

6. 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 catagories.

7. Television I/O routines. These routines are discussed in the chapter on televisions and are not discussed further here.

A detailed description of the call sequences to each of these 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 includes IDCH.INC, DDCH.INC and CDCH.INC. The text of these include files 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 standalone 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, nonwait-mode I/O and 3) map I/O which can be double buffered, 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 routine 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 each application program and ZDCHIN is not told what the actual dimension is; this may lead to problems if FTAB is dimensioned too small in a given applications task.

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 short integer.

15.1.3 Disk Files

The AIPS system uses binary files for data and text files for source code and control 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 for text files).

15.1.3.1 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 catalogues are supported. Applications software will handle up to 24 characters in a name.

An example from a VAX system with private catalogues is "DAOn:ttdcccvv.uuu" ; where n is the one relative disk drive number, DAOn: is a logical variable which is assigned to a directory, tt is a two character file type (eg. 'MA'), d is the one relative disk drive number(hex), ccc is the catalogue slot number(hex), vv is the version (hex) (01 for "MA" and "UV" files), and uuu is the users number in hexidecimal notation.

"Map" type files are files on which it should be possible to double buffer. It should be possible to contract "map" files but it is not necessary to expand "map" files so these files may be forced to be contigious on the disk. Contigious files are more efficient but they cause problems for users with large files. These files should be capable of random access with I/O beginning on a disk sector boundary.

"Non-map" files should be expandable and contractable. These files should be capable of random access with I/O beginning on a disk sector boundary.

15.1.3.2 Text Files - Text files are used primarily for storing source code and control information such as the RUN and HELP/INPUT files. Currently text files may be read but not written using AIPS routines. The source code routines are accessed primarily by AIPS managment routines such as the AIPS manual printing program.

Different types of text files are kept in different areas which have directories. 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 area are described in the following:

- HE - These are the HELP files which specify which AIPS adverbs are to be sent to tasks and contain the primary user documentation.

- IN - Same as HE. This is a relic of older versions in which the HELP files and INPUTS files were distinct.

- 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.

- DC - These are the programmer documentation files, primarily sections of the AIPS manual.

- SO - These are the "standard" source code routines. These routines are those which should conform to all AIPS coding standards.

- SR - This area contains source code which is used only by the AIPS program and standalone utility programs but not AIPS tasks.

- SI - This area contains the include files.

- SN - This area contains source code which has not been determined to meet all AIPS coding standards. Code in this area may give problems in a new installation.

- SF - This area contains the source code for the true array processor routines.

- SP - This area contains the source code for the pseudo array processor routines.

- SL - This area contains miscellaneous files.

15.2 DATA MANIPULATION ROUTINES

The internal form in which characters 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) formats to the internal formats. The format of data on FITS tape files is discussed in another chapter.

The following list gives the names and uses of the upper level data manipulation "Z" routines; in practical installations more Z routines will be required. Details of the call sequences are given later in this chapter.

- ZBYTFL - Flip the order of bytes if necessary on local machine.
- ZCLC8 - converts local characters to standard ASCII.
- ZC8CL - converts standard ASCII to the local characters.
- ZMCACL - converts Modcomp compressed ASCII to local ASCII.
- ZDM2DL converts Modcomp double precision real numbers to local.
- ZGETCH - extracts a single character from a R4 word.
- ZGTBIT - extract bits from a word.
- ZGTBYT - extract byte from a word.
- ZI16IL - converts standard 16 bit integers to the local short integer.
- ZI32IL - converts standard 32 bit integers to a pair of local short integers.
- ZI8L8 - converts 8 - bit unsigned binary numbers to bytes.
- ZILI16 - converts local short integers to external format 16 bit integers.
- ZPTBIT - sets bits in a word.
- ZPTBYT - sets a byte in a word.
- ZPUTCH - inserts a character into a string.
- ZP4I4 - converts pseudo I*4 to true I*4.
- ZRDMF - converts data packed in DEC-Magtape format to pairs of 16 bit integers.
- ZRM2RL - Converts Modcomp single precesion floating numbers to local.
- ZR8P4 - converts between pseudo I*4 and REAL*8.

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.

A short description of the 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 disk files or devices.
- ZCMPRS - contracts disk files.
- ZCREAT - creates disk files.
- ZDESTR - destroys disk files
- ZEXIST - determines if a given file exists.
- ZEXPND - expands "non-map" files.
- ZFIO - does "non-map" (single buffer) I/O to disk files and devices.
- ZMIO - does "map" (double buffer, wait mode) I/O to disk files and devices.
- ZMSGCL - closes the message file.
- ZMSGDK - writes to the message file.
- ZMSGOP - opens the message file.
- ZOPEN - opens disk files and devices.
- ZPHFIL - constructs physical file names.
- ZRENAM - changes the physical name of a file.
- ZWAIT - suspends the calling task until an I/O operation initiated by ZMIO is complete.

15.4 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 the 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 these routines; complete descriptions of the call sequence can be found at the end of this chapter. In any implementation there will be lower level Z routines called by these routines.

- ZACTV8 - activates a specified task.
- ZCPU - returns the amount of CPU time used by the current process.
- ZDATE - returns the current calender date.
- ZDELAY - delays the calling task for a specified period.
- ZFREE - determines the amount of disk space available on each of the disks.
- ZGNAME - returns the actual task/process name.

- ZMYVER - determines the default version of AIPS (NEW, OLD, TST).

- ZPRIO - raises or lowers a task's priority.

- ZPRPAS - prompted read for password.

- ZTACTQ - checks if a given process is active.

- ZTIME - returns the current time.

- ZSTAIP - restores the process to its normal state on the completion of an interactive AIPS process.

- ZTKILL - kills (aborts) a specified task.

- ZTQSPY - writes a list of the current AIPS processes running to the user monitor terminal and the message file.

- ZWHOMI - returns the name of the executing task.

15.5 DEVICE (NON-DISK) I/O ROUTINES

Many 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 sequence can be found at the end of this chapter.

- ZDOPRT - plots a bit map onto the plotter.

- ZENDPG - does a page eject on the line printer.

- ZQMSIO - Opens and/or writes to a QMS Lasergraphix

- ZTAPE - positions a tape and writes file marks.

- ZTKBUF - formats the output buffer for the graphics output device.

- ZTKCLS - closes a TK device.

- ZTKOPN - opens a TK device.

- ZTTYIO - reads and writes to the terminal.

- ZPRMPT - does a prompted read from the terminal.

15.6 DIRECTORY AND TEXT FILE ROUTINES

Text files are used for source code and control information and have been discussed previously in this chapter. Currently text files may be read but not written from AIPS routines.

The following list briefly describes the function of the special routines for text files; detailed descriptions of the call sequences are found at the end of this chapter.

- ZTOPEN - opens a text file.

- ZTREAD - reads a text file.

- ZTCLOS - closes a text file.

- ZTXMAT - searches a directory for files whose names begin with a given character string.

- ZGTDIR - returns the directory for a text file area.

15.7 MISCELLANEOUS

Several Z routines don't naturally fit in any of the above categories. The following list gives a brief description of each; details of the call sequence and function are given at the end of this chapter.

- ZDCHIN - initializes the Device Characteristics Common.

- ZMATH4 - does pesudo I*4 arithmetic.

- ZTFILL - initializes the FTAB array in the Device Characteristics Common.

- ZKDUMP - dumps an array to the user message monitor and message file in a number of different formats.

15.8 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 statments as described in the following list.

- Dxxx.INC. These INCLUDE files contain Fortran type (with dimension) declarations.

- Cxxx.INC. These files contain Fortran COMMON statments.

- Exxx.INC. These contain Fortran EQUIVALENCE statments.

- Vxxx.INC. These contain Fortran DATA statments.

- Ixxx.INC. Similar to Dxxx.INC files in that they contain type declarations but the declaration of some varaible is omitted. This type of include is used in the main program to reserve space for the omitted variable in the appropriate common. The omitted variable must be declared and dimensioned separately.

- Zxxx.INC. These INCLUDE files contain declarations which may change from one computer or installation to another.

15.8.1 CDCH.INC

```
C                                                        Include CDCH
      COMMON /DCHCOM/ XPRDMM, XTKDMM, SYSNAM, VERNAM, RLSNAM, TIMEDA,
     *    TIMESG, TIMEMS, TIMESC, TIMECA, TIMEBA, TIMEAP, RFILIT,
     *    NVOL, NBPS, NSPG, NBTB1, NTAB1, NBTB2, NTAB2, NBTB3, NTAB3,
     *    NTAPED, CRTMAX, PRTMAX, NBATQS, MAXXPR, CSIZPR, NINTRN,
     *    KAPWRD, NCHPFP, NWDPFP, NWDPDP, NWDPLI, NWDPLO, NBITWD,
     *    NWDLIN, NCHLIN, NTVDEV, NTKDEV, BLANKV, NTVACC, NTKACC,
     *    UCTSIZ, BYTFLP, USELIM, NBITCH
      COMMON /FTABCM/ DEVTAB, FTAB
C                                                        End CDCH.
```

15.8.2 CMSG.INC

```
C                                                               Include CMSG
      COMMON /MSGCOM/ MSGCNT, TSKNAM, NPOPS, NLUSER, MSGTXT,
     *    NACOUN, MSGSUP, MSGREC, MSGKIL
C                                                               End CMSG.
```

15.8.3 DDCH.INC

```
C                                                               Include DDCH
      REAL*4 XPRDMM, XTKDMM, SYSNAM(5), VERNAM, RLSNAM(2), TIMEDA(15),
     *    TIMESG, TIMEMS, TIMESC, TIMECA, TIMEBA(4), TIMEAP(3),
     *    RFILIT(14)
      INTEGER*2 NVOL, NBPS, NSPG, NBTB1, NTAB1, NBTB2, NTAB2,
     *    NBTB3, NTAB3, NTAPED, CRTMAX, PRTMAX, NBATQS, MAXXPR(2),
     *    CSIZPR(2), NINTRN, KAPWRD, NCHPFP, NWDPFP, NWDPDP, NWDPLI,
     *    NWDPLO, NBITWD, NWDLIN, NCHLIN, NTVDEV, NTKDEV, BLANKV,
     *    NTVACC, NTKACC, UCTSIZ, BYTFLP, USELIM, NBITCH,
     *    DEVTAB(50), FTAB(1)
C                                                               End DDCH.
```

15.8.4 DMSG.INC

```
C                                                               Include DMSG
      INTEGER*2 MSGCNT, TSKNAM(3), NPOPS, NLUSER, MSGSUP, MSGREC,
     *    MSGKIL
      INTEGER*4 NACOUN
      REAL*4    MSGTXT(20)
C                                                               End DMSG.
```

15.8.5 IDCH.INC

```
C                                                               Include IDCH
      REAL*4 XPRDMM, XTKDMM, SYSNAM(5), VERNAM, RLSNAM(2), TIMEDA(15),
     *    TIMESG, TIMEMS, TIMESC, TIMECA, TIMEBA(4), TIMEAP(3),
     *    RFILIT(14)
      INTEGER*2 NVOL, NBPS, NSPG, NBTB1, NTAB1, NBTB2, NTAB2,
     *    NBTB3, NTAB3, NTAPED, CRTMAX, PRTMAX, NBATQS, MAXXPR(2),
     *    CSIZPR(2), NINTRN, KAPWRD, NCHPFP, NWDPFP, NWDPDP, NWDPLI,
     *    NWDPLO, NBITWD, NWDLIN, NCHLIN, NTVDEV, NTKDEV, BLANKV,
     *    NTVACC, NTKACC, UCTSIZ, BYTFLP, USELIM, NBITCH,
```

```
      *   DEVTAB(50)
C                                                        End IDCH.
```

15.9 ROUTINES

15.9.1 Data Manipulation

15.9.1.1 ZBYTFL - interchange the low order and high order bytes for all words in the input buffer and put the results in an output buffer. The input buffer is unchanged except in the special case where the input buffer has the same starting address as the output buffer.

```
      ZBYTFL(NWORDS, INBUF, OUTBUF)
```

Inputs:	NWORDS	I*2	The length of the input buffer in I*2 words
	INBUF	I*2()	The input buffer.
Output:	OUTBUF	I*2()	The output buffer containing byte swapped words.

15.9.1.2 ZCLC8 - converts local characters in a buffer to standard 8-bit ASCII in another buffer - which may be the same buffer.

```
      ZCLC8 (NCHAR, INB, NP, OUTB)
```

Inputs:	NCHAR	I*2	Number of characters
	INB	R*4(*)	Input buffer in local chars: start at 1
	NP	I*2	Start index in output buffer in units of 8-bit chars, 1-relative
Output:	OUTB	R*4(*)	Output buffer

15.9.1.3 ZC8CL - extracts 8-bit ASCII standard characters from a buffer and stores them in the local character form. Must work even when INARR and OUTARR start at the same address.

```
      ZC8CL (NCHAR,NP,INARR,OUTARR)
```

Inputs:	NCHAR	I*2	Number of characters to extract
	NP	I*2	Start position in input buffer in units of 8-bit characters
	INARR	R*4(*)	Input buffer
Output:	OUTARR	R*4(*)	Output buffer

15.9.1.4 ZMCACL - converts Modcomp compressed ASCII characters to blank filled 80 byte records in local character form. Two characters per short integer are assumed. A blank record will be placed in the output between files. This routine is only used by task FILLR which reads VLA Modcomp/archive format data tapes.
NOTE: this routine will not work inplace.

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. (Note may be bytes 1-2)
4-	Compressed ASCII characters, a NUL (Hex 00) terminates. A negative value (most signifigant bit on) indicates a repeatition 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 (twos complement -1) indicated two 'C's.

NOTE: logical records may span physical records.
NOTE: padding to 80 byte records is already done on tape.

ZMCACL (NBYTES, INBUF, OUTBUF, LASTCH)

Inputs:

NBYTES	I*2	The length of the input buffer in bytes.
INBUF(*)	I*2	Input buffer of MODCOMP compressed ASCII 2 bytes per word.
LASTCH	I*4	Address (in RECORD) of previous last character written. Should be zero for first call.

Outputs:

NBYTES	I*2	The number of bytes in the output buffer
OUTBUF(*)	I*2	Output buffer, pack character string. Each MODCOMP logical record is converted to 80 bytes with blank filling. Each record begins at the first byte of every 40 th local short integer. Due to the expansion of the data the size of the output buffer is not strictly predictible.
LASTCH	I*4	Position in RECORD of last character written. Will contain the address - DON'T touch!

15.9.1.5 ZDM2DL - converts Modcomp R*6 or R*8 floating point data into local double precision floating point. Expects, after word flip, 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 compliment of the whole word. Before calling ZDM2DL the data should have the bytes flipped (ZI16IL) which will leave the values split between four short integers. Should work inplace. This routine is only used by task FILLR which reads VLA Modcomp/archive format data tapes.

ZDM2DL (NWORDS, INBUF, OUTBUF)

Inputs:
 NWORDS I*2 The length of the input buffer in words
 INBUF(*) R*8 The input array in MODCOMP R*6 or R*8
  If R*6 the low order two bytes should be zeroed.
Outputs:
 OUTBUF(*) R*8 The output array in local REAL*8

15.9.1.6 ZGETCH - extracts a single character from a real variable and places it in the least significant bits of an otherwise zero integer. This routine allows an NCHAR of 1 to NCHPFP.

ZGETCH (CHAR, WORD, NCHAR)

Inputs:	WORD	R*4	Word to be extracted from (packed string).
	NCHAR	I*2	Char number (1 - NCHPFP) where char n is n'th character printed under format An
Output:	CHAR	I*2	Char is LS bits, 0 in rest.

15.9.1.7 ZGTBIT - gets the lowest order NBITS of WORD and returns them in BITS with the lsb in BITS(1)...sign bit in BITS(16).

ZGTBIT (NBITS, WORD, BITS)

Inputs:	NBITS	I*2	Number of bits to copy
	WORD	I*2	Input word from which bits are extracted.
Output:	BITS(*)	I*2	Resulting "Bit" array (values 0 or 1)

15.9.1.8 ZGTBYT - extracts a byte (half a short integer) from IWORD and returns the byte in the low order byte of DATA with zero in the upper byte. NBYTE = 1 corresponds to the left byte, and NBYTE = 2 corresponds to the right byte as printed using FORTRAN A2 format.

ZGTBYT (DATA, IWORD, NBYTE)

Inputs:
 IWORD I*2 The word containing the input character.
 NBYTE I*2 1 for the "left" byte of IWORD and 2 for the "right" byte.
Outputs:
 DATA I*2 The desired character in the low order byte, padded with zeros in the upper byte.

15.9.1.9 ZI16IL - extracts 16-bit, 2's complement integers from a buffer and puts them into the local small integer form. Must work even when INB and OUTB have the same address.

```
      ZI16IL (NVAL, NP, INB, OUTB)

   Inputs: NVAL   I*2      # values to extract
           NP     I*2      start position in input counting from 1
                           in units of 16-bit integers
           INB    I*2(*)   Input buffer
   Output: OUTB   I*2(NVAL)  Output buffer
```

15.9.1.10 ZI32IL - extracts 32-bit, 2's complement integers from a buffer and puts them into the local small integer form. Must work even when INB and OUTB have the same address. The IBM order must apply to the output: i.e. the most significant part of the 32-bit integer must be at a lower index in OUTB than the least significant part. They will be picked up into standard pseudo I*4 via IP(2) = OUTB(i), IP(1) = OUTB(i+1).

```
      ZI32IL (NVAL, NP, INB, OUTB)

   Inputs: NVAL   I*2      # values to extract
           NP     I*2      start position in input counting from 1
                           in units of 32-bit integers
           INB    I*2(*)   Input buffer
   Output: OUTB   I*2(2*NVAL)  Output buffer
```

15.9.1.11 ZI8L8 - converts 8-bit unsigned binary numbers to "bytes" (one-half of a local small integer). Must work when input and output buffers are the same.

```
      ZI8L8 (NVAL, NP, INB, OUTB)

   Inputs: NVAL  I*2       # values
           NP    I*2       First value to get from INB counting from 1
                           in units of 8-bit numbers
           INB   I*2(*)    Input buffer
   Output: OUTB  I*2(NVAL/2)   Output buffer
```

15.9.1.12 ZILI16 - converts a buffer of local small integers to a buffer of standard 16-bit, 2's complement integers.

```
      ZILI16 (NINT, INB, NP, OUTB)

   Inputs: NINT   I*2     Number of integers
           INB    I*2(*)  Input buffer: start at index 1
           NP     I*2     start point in output buffer 1-relative in
                          units of standard 16-bit integers
   Outputs:OUTB   I*2(*)  Out buffer
```

15.9.1.13 ZP4I4 - Converts Pseudo I*4 integer to true I*4.

```
      ZP4I4 (P4, I4)

    Input:
      P4  I*2(2)   pseudo I*4 value
    Output:
      I4  I*4      I*4 value
```

15.9.1.14 ZPTBIT - builds WORD from NBITS bit values in the array BITS, where BITS(1) supplies the lsb, BITS(2) the next higher bit. Unspecified bits are zero filled.

```
      ZPTBIT (NBITS, WORD, BITS)

   Inputs:   NBITS    I*2  Number of bits.
             BITS(*)  I*2  "Bit" array.
   Output:   WORD     I*2  Word into which bits are placed.
```

15.9.1.15 ZPTBYT - puts the low order byte of DATA in either the "left" or the "right" byte of IWORD. The convention is that NBYTE = 1 corresponds to the left byte, and NBYTE = 2 corresponds to the right byte as printed using FORTRAN A2 format.

```
      ZPTBYT (DATA, IWORD, NBYTE)

   Inputs:
     DATA    I*2  The word containing the input character.
     NBYTE   I*2  1 for the "left" byte of IWORD and 2 for the "right"
                   byte.
   Outputs:
     IWORD   I*2  The word to receive the byte.
```

15.9.1.16 ZPUTCH - inserts the appropriate number of bits of CHAR (one character worth, taken from the least significant bits) into the specified character position of WORD.

ZPUTCH (CHAR, WORD, NCHAR)

Inputs:	CHAR	I*2	character in lsb's.
	NCHAR	I*2	character position in which to insert: char n is the n'th character printed by An.
In/out:	WORD	R*4	'word' to have character inserted. (packed string)

15.9.1.17 ZRDMF - converts data packed in DEC-Magtape format (DMF) to pairs of 16 bit integers, 1 per local short integer. 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

The Rn refer 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. For 16 bit output bits R2-R17 are returned for the word count for all other data bits F0-F15 and R0-R15 are returned. Input data is assumed packed into 2 1/2 short integers and output data will be returned in a pair of local short integers per DEC-10 word. This routine is only used by task FILLR which reads VLA Modcomp/archive format data tapes.

ZRDMF (NWORDS, INBUF, OUTBUF, FLAG)

Inputs:		
NWORDS	I*2	The length of the input buffer in DEC-10 words
INBUF(*)	I*2	Input buffer of DMF format data.
FLAG	I*2	If .gt. 0 then the first word word is the beginning of a tape block.
Outputs:		
OUTBUF(*)	I*2	Output buffer, two local short integers per input DEC-10 word.

15.9.1.18 ZRM2RL - converts Modcomp single precision floating point data into local single precision floating point. Expects, after word flip, 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 compliment of the whole word. Before calling ZRM2RL the data should have the bytes flipped (ZI16IL) which will leave the values split between two short integers. Should work inplace. This routine is only used by task FILLR which reads VLA Modcomp/archive format data tapes.

```
ZRM2RL (NWORDS, INBUF, OUTBUF)

Inputs:
 NWORDS    I*2 The length of the input buffer in words
 INBUF(*)  R*4 The input array in MODCOMP R*4
Outputs:
 OUTBUF(*) R*4 The output array in local REAL*4
```

15.9.1.19 ZR8P4 - converts between pseudo I*4 and R*8.

```
ZR8P4 (OP, INTG, DX)

Inputs: OP    R*4      '4TO8' Pseudo I*4 to R*8
                       '8TO4' R*8 to pseudo I*4
                       '4IB8' IBM i*4 to R*8
                       '8IB4' R*8 to IBM I*4
In/out: INTG  I*2(2)   the I*4
        DX    R*8      the R*8
```

Pseudo I*4 has the form of two short integers with the least significant half at the lower I*2 index.
IBM I*4 has the form of a 2's complement, 32-bit integer with the most significant 16 bits in the I*2 word of lower index and the least significant 16 bits in the I*2 word of higher index.

15.9.2 Disk I/O

15.9.2.1 ZCMPRS - releases unused disk space from a non-map file. Will also allow "map" files. File must be open. "Byte" defined as 1/2 of a small integer.

```
ZCMPRS (IVOL, PNAME, LUN, LSIZE, SCRTCH, IERR)

Inputs:  IVOL   I*2      volume number
         PNAME  R*4(6)   physical file name
         LUN    I*2      logical unit number under which file is
                         open.
In/Out:  LSIZE  I*4      (In) desired final size in bytes.
                         (out) actual final size in bytes.
```

```
Outputs: SCRTCH I*2(256) Scratch buffer
         IERR   I*2      error code: 0 => ok
                                     1 => input data error
                                     2 => compress error FMGR
```

15.9.2.2 ZCREAT - creates a disk file.

```
ZCREAT (IVOL, PNAME, ISIZE, MAP, ASIZE, SCRTCH, IERR)

Inputs:
   IVOL    I*2       Disk drive unit number(1-8).
   PNAME   R*4(6)    Physical file name given by ZPHFIL.(ASCII)
                     left justified, padded with blanks.
   ISIZE   I*4       Requested size of the file in bytes. Will be
                     rounded to next higher granual.
   MAP     L*2       True if map file.
outputs:
   ASIZE   I*4       Actual number of bytes in the new file.
                     (byte = half of a short integer)
   SCRTCH  I*2(256)  Scratch buffer.
   IERR    I*2       Error return code. The values mean:
                     0 - success.
                     1 - file already exists.
                     2 - volume is not available.
                     3 - space is not available.
                     4 - Other.
```

15.9.2.3 ZDESTR - Destroys the file associated with PNAME. The file must already be closed.

```
ZDESTR (IVOL, PNAME, IERR)

Input:
  IVOL  I*2      Volume number of disk.
  PNAME R*4(6)   Physical file name.
Output:
  IERR  I*2      Completion code. 0=good.
                                  1=file not found
                                  2=failed
```

15.9.2.4 ZEXIST - determines if a file exists. If so, the size of the file is returned.

```
      ZEXIST (IVOL, PHNAME, ISIZE, SCRTCH, IERR)

  Inputs:
    IVOL    I*2      The disk volume to seach.
                     This information is found in PHNAME.
    PHNAME R*4(6)    File name.
  Outputs:
    ISIZE   I*4      Size of the file in 512 byte blocks.
    SCRTCH I*2(256)  Scratch buffer.
    IERR    I*2      Error code 0 = file exists, 1-file not found,
                     2 = other.
```

15.9.2.5 ZEXPND - increases the size of a non-map file.

```
      ZEXPND (LUN, IVOL, PHNAME, NREC, IERR)

  Inputs: LUN     I*2       LUN of file (already open)
          IVOL    I*2       disk volume number of file
          PHNAME R*4(6)     physical file name of file
  In/Out: NREC    I*2       # 256-integer records requested/received
  Output: IERR    I*2       error code 0 => ok
                                       1 => input error
                                       2 => FMGR error
```

15.9.2.6 ZFIO - reads or writes one logical record between core and device LUN. For disk devices, the record length is always 512 bytes (a byte being defined as half of a short integer). NREC gives the random access record number (in units of 512 bytes). For non-disk devices, NREC contains the number of bytes. Used for non-map files.

Note: there is a temporary version named ZFI3 differing from ZFIO only in that NREC is an I*2 value. This version will disappear once all application code has been converted.

```
      ZFIO (OPER, LUN, FIND, NREC, BUF, IERR)

  Inputs:
     OPER    R*4  Operation = 'READ' or 'WRIT'
     LUN     I*2  logical unit number
     FIND    I*2  pointer to file area in FTAB
     NREC    I*4  record number in file: starts with 1 (DISKS);
                  number of bytes (Sequential DEVICES)
  In/Out:
     BUF     I*2(256) array to hold record
  Output:
```

```
      IERR   I*2   error code: 0 -> ok
                               1 -> file not open
                               2 -> input error
                               3 -> I/O error
                               4 -> end of file
                               5 -> begin of medium
                               6 -> end of medium
```

15.9.2.7 ZMIO - a low level random access, large record, double buffered device I/O routine. Used for "map" files.

Note: there is a temporary version named ZMI3 differing from ZMIO only in that BLKNO is a pseudo I*4 value. This version will disappear once all application code has been converted.

```
      ZMIO (OP, LUN, FIND, BLKNO, NBYTES, BUFF, IBUFF, IERR)
   Inputs:
      OP       R*4  Operation - 'READ', 'WRIT'. ASCII - 4 characters.
      LUN      I*2  Logical unit number of a previously opened file.
      FIND     I*2  Pointer to FTAB returned by ZOPEN.
      BLKNO    I*4  One relative beginning block number. The size of a
                    block is given by NBPS in COMMON/DCHCOM/.
      NBYTES   I*2  Number of bytes to transfer.
      BUFF     R*4  The i/o buffer.
      IBUFF    I*2  Buffer number to be used - 1 or 2.
   Outputs:
      IERR     I*2  Error return code:
                       0 = Success.
                       1 = File not open.
                       2 = Operation incorrectly specified.
                       3 = I/O error.
                       4 = end of file (no messages)
```

15.9.2.8 ZMSGCL - closes message file associated with LUN removing any exclusive use state and clears up the FTAB.

```
      ZMSGCL (LUN, FIND, IERR)

   Inputs:  LUN  I*2  logical unit number (6 or 12)
   Output:  IERR I*2  error code: 0 -> no error
                                  1 -> Deaccess or Deassign error
                                  2 -> file already closed in FTAB
                                  3 -> both errors
                                  4 -> erroneous LUN
```

15.9.2.9 ZMSGDK - reads or writes one 512-byte logical record between the buffer BUF and disk unit LUN. Special version for message writing.

```
ZMSGDK (OPER, LUN, FIND, NREC, BUF, IERR)

Inputs:
   OPER   R*4  Operation = 'READ' or 'WRIT'
   LUN    I*2  logical unit number (12)
   FIND   I*2  pointer to file area in FTAB
   NREC   I*2  record number in file: starts with 1
   BUF    I*2  (256) array to hold record
Output:
  IERR    I*2  error code: 0 => ok
                           1 -> file not open
                           2 => input error
                           other -> I/O error
```

15.9.2.10 ZMSGOP - opens message files.

```
ZMSGOP (LUN, IND, IVOL, PNAME, MAP, EXCL, WAIT, IERR)

Inputs:
   LUN    I*2    Fortran Logical file number. (6 or 12)
   IVOL   I*2    Disk volume containing file, 1,2,3,...
   PNAME  R*4(2) 8 Character physical file name,left justified
   MAP    L*2    Is this a map file.
   EXCL   L*2    Desire exclusive use.
   WAIT   L*2    I will wait.
Output:
   IND    I*2   Index into FTAB for the file control block.
   IERR   I*2   error code
                 0 = No error
                 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
```

15.9.2.11 ZOPEN - opens logical files, fills FTAB entries and performs full open on disk files. Tape units are assigned an I/O channel.

```
ZOPEN (LUN, IND, IVOL, PNAME, MAP, EXCL, WAIT, IERR)

Inputs:
   LUN    I*2      Logical unit number.
```

```
      IVOL    I*2       volume number, 1,2,3,...
      PNAME   R*4(6)    24-character physical file name,left justified,
                        packed, and padded with blanks.
      MAP     L*2       is this a map file ?
      EXCL    L*2       desire exclusive use?
      WAIT    L*2       I will wait?
   Output:
      IND     I*2       Index into FTAB for the file control block.
      IERR    I*2       Error return code:
                            0 = no error
                            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
```

15.9.2.12 ZPHFIL - constructs a physical file name in PNAM from ITYPE, IVOL, NSEQ, and IVER. New version designed either for public data files or user specific files. This routine contains the logical assignment list for Graphics devices. Numerical values are encoded as hexidecimal numbers.

```
   EXAMPLE: If ITYPE='MA', IVOL=8, NSEQ=801, IVER=153, NLUSER=768 then
               PNAME='DA08:MA832199;1' for public data or
               PNAME='DA08:MA832199.300;1' for private data
   ITYPE = 'MT' leads to special name for tapes
   ITYPE = 'TK' leads to special name for TEK4012 plotter CRT
   ITYPE = 'TV' leads to special name for TV device
   ITYPE = 'ME' leads to special logical for POPS memory files

      ZPHFIL (ITYPE, IVOL, NSEQ, IVER, PNAM, IERR)
   Inputs:
      ITYPE    I*2  Two characters denoting type of file. For example,
                    'MA' for map file.
      IVOL     I*2  Number of the disk volume to be used.
      NSEQ     I*2  User supplied sequence number. 000-999.
      IVER     I*2  User suppplied version number. 00-255.
   Outputs:
      PNAM  R*4(6)  >= 24-byte field to receive the physical file name,
                    left justified (packed) and padded with blanks.
      IERR     I*2  Error return code.
                    0 = good return.   1 = error.
```

15.9.2.13 ZRENAM - Renames a disk file.

```
      ZRENAM (IVOL, NAME1, NAME2, IERR)

   Inputs:
      IVOL   I*2     Volume number (1 relative).
      NAME1  R*4(6)  Old file name. 24 char., left justified, padded
                     on the right with blanks, and packed.
      NAME2  R*4(6)  New file name. like NAME1.
   Outputs:
      IERR   I*2     error code.
                  0 - successful completion
                  2 - old file not found
                  3 - volume not found or not ready
                  6 - new file name already exists in directory.
                  7 - other errors.
```

15.9.2.14 ZWAIT - waits until an I/O operation started by ZMIO is complete.

```
      ZWAIT (LUN, IND, IBUF, IERR)

   Inputs:
      LUN  I*2  logical unit number
      IND  I*2  Pointer to FTAB
      IBUF I*2  Wait for 1st or 2nd buffer in double buffered I/O
   Output:
      IERR I*2  Error return  0 => ok
                              1 => LUN not open
                              3 => I/O error
                              4 => end of file
                              7 => wait service error
```

15.9.3 System Functions

15.9.3.1 ZACTV8 - activates the specified task. This routine is normally in the AIPS program library (for AIPS and other standalone programs but not tasks).

```
      ZACTV8 (NAME, INPOPS, VERSON, PID, IERR)
```

Inputs:	NAME	I*2(3)	root task name. (2 char / integer)
	INPOPS	I*2	Pops # to be used by task.
	VERSON	R*4(5)	Logical name or absolute name of device/directory for area containing the executable module.
Output:	PID	I*2(4)	Process "ID" code of activated task for use by directly subsequent ZTACTQs PID(1) = user number on systems which use that (= 0 otherwise and on all AIPSB invocations) PID(2-4) process ID number (as needed)
	IERR	I*2	error code: 0 => ok. 1 => name invalid or not task. 2 => activation error.

15.9.3.2 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*4	Current CPU accumulation in seconds
	IOCNT	I*4	I/O count

15.9.3.3 ZDATE - returns local time of day.

```
      ZDATE (ID)
```

Output:	ID(1)	year since 0.
	ID(2)	month (1-12).
	ID(3)	day (1-31).

15.9.3.4 ZDELAY - causes the calling program to suspend itself for a specified length of time.

```
      ZDELAY (SECS, IERR)

   Input:
      SECS  R*4  Number of seconds to delay.
   Output:
      IERR  I*2  Error code. 0 = ok,  1 = error.
```

15.9.3.5 ZGNAME - returns the actual task/process name.

```
      ZGNAME (NAME, IERR)

   Outputs: NAME    I*2(3)   Actual name (2 chars / word)
            IERR    I*2      Error code : 0 => ok
```

15.9.3.6 ZMYVER - determines the default version (OLD or NEW or TST). This routine is normally in the AIPS program library (for AIPS and other standalone programs but not tasks).

```
      ZMYVER

   Output: in Common /DCHCOM/ variable VERNAM
                Unpacked String containing 'OLD:','NEW:','TST:'
```

15.9.3.7 ZPRIO - changes the current program's machine priority between that of batch programs and that of interactive programs. This routine is used by tasks using true array processors.

```
      ZPRIO (OP, IERR)

   Inputs: OP    R*4    'UPPP' to inter., 'DOWN' to batch
           IERR  I*2    Error code: 0 => ok
                                    1 => bad OP
                                    2 => illegal request
                                    3 => other failures
```

15.9.3.8 ZPRPAS - prompts the user on his terminal with the prompt string "Password: " and then reads back a 12-character "password" without echoing on the screen.

```
      ZPRPAS (PASS, BUFF, IERR)

   Outputs: PASS   R*4(3)    Password - 12 unpacked characters: left
                                justified and blank filled.
            BUFF   I*2(256)  scratch buffer (if needed)
            IERR   I*2       error code: 0 => ok
                               ??? => I/O error of some sort
```

15.9.3.9 ZTACTQ - determines if a specified task is active.

```
      ZTACTQ (NAME, ACTIVE, IERR)

   Inputs:  NAME      I*2(3)     actual task name.(2 char/integer)
   Output:  ACTIVE    L*2        T => task active.
            IERR      I*2        error number:
                                     0 => ok.
                                     1 => invalid task name.
```

15.9.3.10 ZTIME - returns the local time of day.

```
      ZTIME (IT)

   Output:  IT(1) I*2  hours  (0-23)
            IT(2) I*2  min    (0-59)
            IT(3) I*2  sec    (0-59)
```

15.9.3.11 ZFREE - This routine will calculate the number of free 512 byte blocks that are available on the disks used for AIPS data and print the information on the screen. This routine is normally in the AIPS program library (for AIPS and other standalone programs but not tasks).

```
      ZFREE (IERR)

   Inputs:
       From common /DCHCOM/
       NVOL   I*2  Number of AIPS disks.
   Output:
       IERR   I*2  0 = ok, 1=error in disk logical name.
```

15.9.3.12 ZSTAIP - performs any operations needed to normalize the local operating system at the conclusion of an interactive AIPS session. This routine is normally kept in the AIPS program library (not for tasks).

ZSTAIP (SCRTCH)

Outputs: SCRTCH I*2(256) Scratch buffer

15.9.3.13 ZTKILL - will delete the task/process specified by NAME. This routine is normally in the AIPS program library (for AIPS and other standalone programs but not tasks).

ZTKILL (NAME, IERR)

Inputs: NAME I*2(3) actual task name.(2 char/integer)
Output: IERR I*2 error number:
0 => ok.
1 => error.

15.9.3.14 ZTQSPY - obtains entire list of AIPS-originated tasks now running in system and prints info about them via MSGWRT. This routine is normally in the AIPS program library (for AIPS and other standalone programs but not tasks).

ZTQSPY (TLIST)

Output: TLIST I*2(256) Scratch buffer

15.9.3.15 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 common /DCHCOM/. It checks for remote entries at this stage and uses the true device numbers (set by ZDCHIN) to do the assignments. This routine is normally in the AIPS program library (for AIPS and other standalone programs but not tasks).

ZWHOMI (IERR)

Output: IERR I*2 error code: 0 ok.
1 => task is AIPS, but NPOPS illegal.
2 => task is not AIPS.

15.9.4 Non-disk I/O Routines

15.9.4.1 ZDOPRT - reads a bit map such as produced by PRTDRW and converts it into a FORTRAN file that can be spooled to the printer-plotter as a plot.

```
      ZDOPRT (IVOL, IBMLUN, NCOPY, FILNAM, DESTRY, ISIZE,
     *   INBLK, IERR)

     Inputs:
        IVOL      I*2     volume no. of bit map disk (1 rel)
        IBMLUN    I*2     bit map logical unit number.
        NCOPY     I*2     Number of copies of the plot to make.
        FILNAM    R*4(6)  physical file name of bit map.
        DESTRY    L*2     destroy bit file when done?
        ISIZE     I*2     size of INBLK in words.
      In/Out:
        INBLK     I*2(*)  scratch buffer
     Outputs:
        IERR      I*2     error return code.
                          0 => OK, otherwise failed.
```

15.9.4.2 ZENDPG - advances the line printer to avoid "burn-out" on electrostatic type printers.

```
      ZENDPG (LINE)

   Inputs:  LINE    I*2    # lines printed on page so far
```

15.9.4.3 ZQMSIO - opens a file for printing a plot on the QMS Lasergraphix using the name QMS.PLT (OP = 'OPEN') or writes data to the QMS device (or temp file) (OP = 'WRIT').

```
      ZQMSIO (OP, QMSLUN, N, LINE, IERR)

   Inputs: OP        I*2     'OPEN', 'WRIT'
           QMSLUN    I*2     LUN to use
           N         I*2     Number of characters in LINE (WRIT only)
           LINE      L*1(N)  Characters to go to QMS (WRIT only)
   Output: IERR      I*2     Error code: 0 => ok
                                 1 => bad OP
                                 2 => OPEN can't find logical name queue
                                 3 => OPEN can't assign logical name
                                 6 => OPEN or WRIT I/O error
```

15.9.4.4 ZTAPE - Performs standard tape manipulating funotions.

```
      ZTAPE (OP, LUN, FIND, COUNT, IERR)

   Inputs:
       OP  R*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. Works for VMS 3.0 & later.
                    'MONT' = mount tape. Works for VMS 3.0 and later.
                    '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*2  logical unit number
       FIND I*2  FTAB pointer.  Drive number for MOUNT/DISMOUNT.
       COUNT I*2 Number of records or file marks to skip. On MOUNT
                 this value is the density.
     Outputs:
       IERR  I*2  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
```

15.9.4.5 ZTKBUF - puts the low order byte of IN into the proper byte of the TEKTRONIX output buffer (TKBUFF). The "Z" is to allow other conversions as required locally.

```
      ZTKBUF (IN, IT, FIND, IERR)

   Input:  IN      I*2  the low order byte of this word is put into
                        DRBUFF.
           IT      I*2 Type of data: 1 control, 2 position, 3 char
           FIND    I*2 FTAB position of TEK 4012 data.
   Output: IERR    I*2 error code. 0-ok, 1-write error.
   COMMON: TKPOS   Byte position in TKBUFF to place IN.
           TKBUFF  TEKTRONIX output buffer.
```

15.9.4.6 ZTKCLS - closes a TK (Tektronix) device.

```
      ZTKCLS (LUN, IND, IERR)

   Inputs: LUN   I*2    Logical unit number
           IND   I*2    Pointer to FTAB
   Output: IERR  I*2    Error code: 0 -› ok, else from ZCLOSE.
```

15.9.4.7 ZTKOPN - opens a TK (Tektronix) device.

```
      ZTKOPN (LUN, IND, IERR)

   Inputs: LUN   I*2    Logical unit number for TK device
   Output: IND   I*2    pointer to FTAB
           IERR  I*2    error code: 0 =› ok, else failed.
```

15.9.4.8 ZTTYIO - performs I/O to a terminal.

```
      ZTTYIO (OPER, LUN, FIND, NBYTES, BUFFER, IERR)

   Inputs: OPER    R*4     'READ' or 'WRIT'
           LUN     I*2     LUN of open device (usually 5 or 6)
           FIND    I*2     Pointer to FTAB for open device
           NBYTES  I*2     # bytes (characters) to transmit (‹= 132)
   In/out: BUFFER  R*4(*)  I/O buffer
   Output: IERR    I*2     Error code: 0 =› ok
                                       1 =› file not open
                                       2 =› input parameter error
                                       3 =› I/O error
                                       4 =› end of file
```

15.9.4.9 ZPRMPT - prompts user on CRT screen and reads a line. This routine is normally in the AIPS program library (for AIPS and other standalone programs but not tasks).

```
      ZPRMPT (IPC, BUFF, IERR)

   INPUT:    IPC     I*2      prompt character.
   OUTPUT:   BUFF    I*2(40)  line of user input.
             IERR    I*2      error code:   0 =› ok.
                                            1 =› read/write error.
```

15.9.5 Directory And Text File

15.9.5.1 ZTCLOS - closes a text file.

```
      ZTCLOS (LUN, FIND, IERR)
  Inputs:   LUN     I*2  logical unit number.
            FIND    I*2  Not used with this routine.
  Output:   IERR    I*2  Error code.
                           0 => no error.
                           1 => RMS error.
                           2 => file not open.
```

15.9.5.2 ZTOPEN - opens a text file.

```
      ZTOPEN (LUN, FIND, IVOL, PNAME, MNAME, VERSON, WAIT,
     *    IERR)

  Inputs:   LUN      I*2    logical unit number.
            IVOL     I*2    disk drive number.
            PNAME    R*4(6) disk-file type.  Only type ('HE' ect)
                            used. Should be generated by ZPHFIL.
            MNAME    R*4(2) file name.
            VERSON   R*4(5) Version (determines in which dir/subdir
                            to look for the file).
            WAIT     L*2    T  => wait until file is available.
  Output:   IERR     I*2    error 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
                                6  => Other open errors.
            FIND     I*2 pointer to FTAB location.
```

15.9.5.3 ZTREAD - reads the next sequential card image from a text file.

```
      ZTREAD (LUN, FIND, BUF, IERR)

   Inputs:   LUN      I*2 logical unit number
             FIND     I*2 FTAB pointer for LUN
```

```
Output:   BUF(*)   I*2 array card image.(> = 80 chars packed)
          IERR     I*2 Error code:
                         0  => No error
                         1  => File not open.
                         2  => End of file.
                         4  => Other.
```

15.9.5.4 ZTXMAT - opens the directory for a source file area and returns a list of member names whose first NCH characters match the first NCH characters of MNAME.

```
      ZTXMAT ( IVOL, PNAME, MNAME, NCH, VERSON, NAMES,
     *   NNAM, IERR)

Inputs: IVOL    I*2        Volume number.
        PNAME   R*4(6)     File name: 24 packed chars
        MNAME   I*2(4)     Text file member name
        NCH     I*2        Number of characters to compare
        VERSON  R*4(5)     Tells which dir to get names from.
Output: NAMES   I*2(4,64)  Names which match NCH chars of MNAME
                           (unpacked, 2 per integer)
        NNAM    I*2        Number of names in NAMES
        IERR    I*2        Error code: 0 => ok
                           1 => none
                           2 => error in inputs or Open
                           3 => I/O error
```

15.9.5.5 ZGTDIR - gets alphabetized list of members of text files.

```
      ZGTDIR (ITYPE, LNAME, HNAME, VERSON, NUM, NAMES, IERR)

Inputs:  ITYPE   I*2     type of file (HE, SO, etc).
         LNAME   I*2(4)  lowest name to include.
         HNAME   I*2(4)  include names lower than this one.
         VERSON  R*4(5)  Version. Set in AIPS as the adverb VERSION.
Output:  NUM     I*2      number of names found.
         NAMES   I*2(4,1000)  sorted file names.
         IERR    I*2      error code.
```

15.9.6 Miscellaneous

15.9.6.1 ZDCHIN - initializes the disk characteristics common. If NDISK < 0, ZDCHIN uses ABS (NDISK) but skips reading parameters from the parameter disk file. Otherwise, ZDCHIN starts by hardcoded parameter values and then resets some based on values on an alterable disk file.

```
ZDCHIN (NDEV, NDISK, NMAP, IOBLK)

Inputs: NDISK I*2  max number regular disk files open at once
        NMAP  I*2  max number of map (double buf) files open at once
        NDEV  I*2  max number of devices open at once
        IOBLK I*2(256) I/O buffer for reading values off disk.
```

15.9.6.2 ZMATH4 - does I*4 arithmetic on pseudo I*4 arguments.

```
ZMATH4 (ARG1, OP, ARG2, RESULT)

Inputs:
   ARG1 P I*4  First P I*4 argument
   OP     I*2  OPeration ='PL'(+);'MI'(-);'MU'(x);'DI'(/)
                         'MN'(min); 'MX'(max)
   ARG2  P I*4 Second P I*4 argument
Outputs:
   RESULT P I*4 Result
```

15.9.6.3 ZKDUMP - dumps portions of an array in INTEGER*2, char*4, hex*2, and REAL*4: i.e. in as many forms as possible ZKDUMP is called a Z routine because the formats may not be acceptable on all machines. This routine is normally in the AIPS program library (for AIPS and other standalone programs but not tasks).

```
   ZKDUMP (I1, I2, K, C)

Inputs: I1    I*2     start subscript in integer array
        I2    I*2     end subscript in integer array
        K     I*2(*)  integer array
        C     R*4(*)  real array equivalenced to K
```

15.9.6.4 ZTFILL - fills in initial values in FTAB.

```
      ZTFILL (FIND, MAP)

   Inputs: FIND    I*2   location in FTAB
           MAP     I*2   T => map part of FTAB
```

INDEX