Chapter 9

Devices

9.1 Overview

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

9.2 Tape Drives

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

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

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

9.2.1 Opening Tape Files (TAPIO)

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

9.2.2 Positioning Tapes

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

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

9.2.3 I/O to Tape Files (TAPIO)

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

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

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

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

VBOUT

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

Sequential External Files

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

9.2.4 Tape Data Structure

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

9.2.5 Tape Data Formats

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

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

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

10. ZRLR32 converts local floating point to 32-bit IEEE floating.
11. ZRLR64 converts local double precision floating point to 64-bit IEEE floating format.

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

9.3 Graphics Displays

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

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

9.3.1 Opening the Graphics Terminal

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

9.3.2 Writing to the Graphics Terminal

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

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

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

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

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

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

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

9.3.3 Activating and Reading the Cursor

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

9.3.4 Updating the Image Catalog

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

9.3.5 An Example

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

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

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

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

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

9.4 INCLUDEs

9.4.1 DTKS.INC

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

9.4.2 DDCH.INC

```
      INC /DDCH.INC
```

9.5 Routines

9.5.1 TAPIO

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

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

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

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

Tapes: $31 \leq$ LUN $\leq 30 +$ NTAPED

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

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

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

9.5.2 TEKFLS

Will write the output buffer TKBUFF to the TEKTRONIX 4012.

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

9.5.3 TEKVEC

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

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

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

9.5.4 TKCATL

Read, write, init the Graphics image catalog

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

9.5.5 TKCHAR

Will write characters to a TEKTRONIX 4012.

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

9.5.6 TKCLR

Will clear the screen for a Tektronix 401n.

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

9.5.7 TKCURS

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

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

9.5.8 TKDVEC

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

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

9.5.9 VBOUT

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

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

9.5.10 ZC8CL

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

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

9.5.11 ZCLC8

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

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

9.5.12 ZI8IL

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

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

9.5.13 ZI16IL

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

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

9.5.14 ZILI16

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

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

9.5.15 ZI32IL

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

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

9.5.16 ZILI32

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

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

9.5.17 ZR32RL

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

The IEEE format is:

```
             1         2         3
01234567890123456789012345678901
seeeeeeeemmmmmmmmmmmmmmmmmmmmmmm
```

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

The value is given by:

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

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

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

9.5.18 ZRLR32

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

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

9.5.19 ZR64RL

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

The IEEE format is:

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

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

The value is given by:

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

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

9.5.20 ZRLR64

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

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

9.5.21 ZTAPE

Performs standard tape manipulating functions.

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

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

9.5.22 ZTKCLS

Close a Tektronix device

```
  ZTKCLS (LUN, FIND, IERR)
  Inputs:
     LUN      I   Logical unit number
     FIND     I   Index in FTAB to file control block for LUN
  Output:
     IERR     I   Error return code: 0 => no error
                     1 => Non-zero ZTKCL2 error
                     2 => Non-zero LSERCH error
                     3 => both 1 and 2
                     4 => invalid LUN
```

9.5.23 ZTKOPN

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

```
  ZTKOPN (LUN, FIND, IERR)
  Inputs:
     LUN    I   Logical unit number
  Output:
     FIND   I   Index in FTAB to file control block for LUN
     IERR   I   Error return code: 0 => no error
                   1 => LUN already in use
                   2 => no such logical device
                   3 => device not found
                   4 => exclusive use denied
                   5 => no room for LUN in FTAB
                   6 => other open errors
```

9.5.24 ZTPCLS

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

```
ZTPCLS (LUN, FIND, IERR)
Inputs:
   LUN    I   Logical unit number
   FIND   I   Index in FTAB to file control block for LUN
Output:
   IERR   I   Error return code: 0 => no error
                 1 => close error
                 2 => non-zero LSERCH error
                 3 => both 1 and 2
                 4 => invalid LUN
```

9.5.25 ZTPMIO

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

```
ZTPMIO (OPER, LUN, FIND, NBYTES, BUFF, IBUFF, IERR)
Inputs:
   OPER     C*4    Operation code 'READ' or 'WRIT'
   LUN      I      Logical unit number
   FIND     I      Index in FTAB to file control block for LUN
   NBYTES   I      Number of 8-bit bytes to transfer
   BUFF     I(*)   I/O buffer
   IBUFF    I      Buffer number to use (1 or 2)
Output:
   IERR     I      Error return code: 0 => no error
                      1 => file not open
                      2 => input error
                      3 => I/O error
                      4 => end of file (no messages)
```

9.5.26 ZTPOPN

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

```
ZTPOPN (LUN, FIND, IVOL, PNAME, OPER, IERR)
Inputs:
   LUN     I       Logical unit number (30 < LUN <= 30 + NTAPED
                      => tape, else disk)
   IVOL    I       Tape drive or disk volume containing file
   PNAME   C*48    tape disk physical file name
   OPER    C*4     'READ' => read only or 'WRIT' => read/write
Output:
   FIND    I       Index in FTAB to file control block for LUN
   IERR    I       Error return code: 0 => no error
                      1 => LUN already in use
```

```
                         2 => file not found
                         3 => volume not found
                         4 => exclusive use denied
                         5 => no room for LUN in FTAB
                         6 => other open errors
```

9.5.27 ZTPWAT

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

```
ZTPWAT (LUN, FIND, IBUFF, LBYTES, IERR)
Inputs:
   LUN      I   Logical unit number
   FIND     I   Index in FTAB to file control block for LUN
   IBUFF    I   Buffer # to wait for (1 or 2)
Output:
   LBYTES   I   Number 8-bit bytes read/written (+1 if tape tape
                record longer than requested)
   IERR     I   Error return code: 0 => no error
                   1 => LUN not open in FTAB
                   3 => I/O error
                   4 => end of file
                   7 => wait service error
```