Writing Fortran Applications Using Tapes [4]

This chapter describes how you can use the tape subsystem from Fortran programs. You can use Fortran programs while working with the tape subsystem and IBM compatible or ER90 devices.

For the examples in this chapter, it is assumed that you understand the `assign`(1) command. For more information, see the `assign`(1) command in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011 or in the *UNICOS/mk User Commands Reference Manual*, Cray Research publication SR–2611.

4.1 IBM compatible tape processing

This section briefly describes how to use the tape subsystem from Fortran programs using IBM compatible devices.

4.1.1 Reading and writing to tape

The following example illustrates the use of a Fortran program to read and write to tape:

1. Reserve a tape by using the `rsv`(1) command. In this example, the tape has a device group name of `TAPE` and the number of devices requested is `1`:

   ```
   rsv TAPE 1
   ```

2. Request a tape mount with the `tpmnt`(1) command. In this example, it is a new, standard labeled tape with a volume identifier of `SCRSL`, a device group name of `TAPE`, a density of `6250`, a write ring specified to be on the reel, a block size of `32768` bytes, and the `-P` option overwriting the existing path name of `fort.20` with the newest version of `fort.20`.

   ```
   tpmnt -n -l sl -v SCRSL -g TAPE -d 6250 -r in -b 32768 -P fort.20
   ```

3. Use the `assign`(1) command to specify that file `fort.20` is a tape.

   ```
   assign -s tape f:fort.20
   ```

4. Compile and load the Fortran write program `tapewr.f`, directing the binary output to executable file `tapewr`:

   ```
   f90 -o tapewr tapewr.f
   ```

5. Execute `tapewr`, using the data from file `input` and appending the output to `tapewr.l`:

   ```
   tapewr < input >> tapewr.l
   ```

6. Compile, load, and execute the Fortran read program by repeating steps 4 and 5, using files `taperd.f`, `taperd`, and `taperd.l`:

   ```
   f90 -o taperd taperd.f
   taperd >> taperd.l
   ```

7. Release the resources:

   ```
   rls -a
   ```

Note the `write(20)` to `unit 20` in `tapewr` and the `read(20)` from `unit 20` in `taperd` reference steps 2 and 3, in which `fort.20` is used in the `tpmnt` and `assign` commands.

Figure 21 shows the Fortran write program, called `tapewr`. You must supply the `COMPUTE` routine:

```
PROGRAM TAPEWR
     INTEGER IBUF(10)
     REAL RNUM(5)
     CHARACTER*21 CDATA
     COMPLEX CNUM(3)
C
C     Write 5 records. Each record contains a mix of data types.
C
      DO 10 I=1,5
       CALL COMPUTE(I,IBUF,RNUM,CDATA,CNUM) ! Compute
       WRITE(20) IBUF,RNUM,CDATA,CNUM       ! Write them out.
  10 CONTINUE
     END
```

Figure 21. Writing an unlabeled tape

Figure 22 shows the Fortran read program, called `taperd`. You must supply the `ANALYZE` routine:

```
PROGRAM TAPERD
     INTEGER IBUF(10)
      REAL RNUM(5)
      CHARACTER*21 CDATA
      COMPLEX CNUM(3)
C
C     Read 5 records.
C
      DO 10 I=1,5
         READ(20) IBUF,RNUM,CDATA,CNUM        ! Read and convert data.
         CALL ANALYZE(I,IBUF,RNUM,CDATA,CNUM) ! analyze...
   10 CONTINUE
      END
```

Figure 22. Reading an unlabeled tape

4.1.2 Reading and writing tape marks

The following example illustrates the use of a Fortran program to read and write tape marks:

1. Reserve a tape by using the `rsv`(1) command:

   ```
   rsv TAPE 1
   ```

2. Compile and load Fortran program `tapemk.f`, directing the binary output to executable file `tapemk`:

   ```
   f90 -o tapemk tapemk.f
   ```

3. Request a tape mount by using the `tpmnt`(1)command. In this example, the tape has standard labels, a path name of `fort.1`, and a volume identifier of `SCRSL;` it uses the `-T` option to let you read or write tape marks:

   ```
   tpmnt -P fort.1 -l sl -v SCRSL -T -g TAPE -n
   ```

4. Use the `assign`(1) command to specify that file `fort.1` is a tape:

   ```
   assign -s tape f:fort.1
   ```

5. Execute `tapemk`:

   ```
   tapemk
   ```

6. Release the resources:

   ```
   rls -a
   ```

Figure 23 shows the Fortran program that reads and writes a tape mark, called `tapemk.`

```
PROGRAM TAPEMK
      INTEGER BLOCK(1000)
C
C     Write 5 tape blocks, each followed by an end-of-file tape
C     mark (EOF).
C
      DO 10 I=1,5
         WRITE(1) BLOCK              ! Write out a tape block/record.
         ENDFILE(1)                  ! Write an EOF.
  10  CONTINUE

      REWIND 1

C
C     Read back the 5 tape blocks (records) and after each, read
C     the end-of-file tape mark (EOF).
C
      DO 20 I=1,5
        READ(1)         BLOCK           ! Read in a tape block/record.
        READ(1, END=20) TPMK           ! Read the EOF.
        PRINT *,’ Error - no EOF.’     ! If no EOF, then error.
        STOP ’error’
      20 CONTINUE
      END
```

Figure 23. Reading and writing tape marks

4.1.3 Positioning a tape by blocks

The following example illustrates the use of a Fortran program to position a tape by blocks:

1. Reserve a tape by using the `rsv`(1) command:

   ```
   rsv TAPE 1
   ```

2. Compile and load Fortran program `pos.f`, directing the binary output to the executable file `pos`:

   ```
   f90 -o pos pos.f
   ```

3. Request a tape mount with the `tpmnt`(1) command. In this example, the tape has standard labels, a path name of `fort.1`, and a volume identifier of `SCRSL:`

   ```
   tpmnt -p fort.1 -l sl -v SCRSL -g TAPE -n
   ```

4. Use the `assign`(1) command to specify that file `fort.1` is a tape:

5. Execute `pos`:

   ```
   assign -s tape f:fort.1
   ```

   ```
   pos
   ```

6. Release the resources:

   ```
   rls -a
   ```

Figure 24 shows the Fortran program that positions a tape by blocks, called `pos`.

```
PROGRAM POS
      INTEGER BLOCK(1000)
C
C     Write 5 records to the tape. (records 1-5)
C
      DO 10 I=1,5
        WRITE(1) BLOCK
   10 CONTINUE
C
C     Backspace the tape over the fifth record to position the
C     file after the fourth record on the tape.
C
      BACKSPACE 1
C
C     Rewrite record 5 with new data, and add records 6-9.
C
      DO 20 I=1,5
        WRITE(1) BLOCK
   20 CONTINUE

      END
```

Figure 24. Positioning by blocks

4.1.4 Positioning a tape by using the SETTP(3) library call

The following examples illustrate the use of the `SETTP`(3) library call in the positioning of a tape. For more information, see the *Application Programmer's Library Reference Manual*, Cray Research publication SR–2165.

4.1.4.1 Example 1

The following example illustrates the positioning of a tape on a multivolume file. The `tpos` program positions your tape to block number 50 on the second volume of a multivolume file.

1. Reserve a tape by using the `rsv`(1) command. In this example, the tape has a device group name of `TAPE` and the number of devices requested is `1`:

   ```
   rsv TAPE 1
   ```

2. Request a tape mount by using the `tpmnt`(1) command. In this example, the tape is an old one with an IBM standard label, with three volume identifiers

of `VSN1:VSN2:VSN3`, a device group name of `TAPE`, a record format of fixed length, a block size of `64000` bytes, and a path name of `fort.1`:

```
tpmnt -o -l sl -v VSN1:VSN2:VSN3 -g TAPE -F F -b 64000 -p fort.1
```

3. Use the `assign`(1)) command to specify that file `fort.1` is a tape.

```
assign -s tape f:fort.1
```

4. Compile and run the program shown in Figure 25:

```
     PROGRAM TPOS
IMPLICIT INTEGER (A - Z)
  DIMENSION BUFF(4096)
  RECLEN=4096
C
C  IN THE SETTP CALL BELOW, THE FIELDS CORRESPOND TO THE FOLLOWING:
C  FIELD 1: UNIT NUMBER.
C      2: BLOCK # REQUEST SIGN. '1H ' INDICATES THAT THE THIRD
C          FIELD (50) IS AN ABSOLUTE BLOCK # RELATIVE TO THE
C          BEGINNING OF THE VOLUME.
C      3: INTEGER BLOCK NUMBER
C      4: VOLUME # REQUEST SIGN. '1H ' INDICATES THAT THE FIFTH
C          FIELD (2) IS AN ABSOLUTE VOLUME # RELATIVE TO THE
C          BEGINNING OF THE VOLUME IDENTIFIER LIST SPECIFIED ON THE
C          TPMNT COMMAND.
C      5: INTEGER VOLUME NUMBER.
C      6: NAME OF VOLUME IDENTIFIER TO BE MOUNTED.
C           0 INDICATES THAT THIS PARAMETER IS IGNORED.
C      7: SPECIFIES WHETHER THE TAPE SHOULD BE SYNCHRONIZED
C          (0=NO).
C      8: INTEGER RETURN STATUS. ON EXIT, INDICATES WHETHER
C          POSITIONING WAS SUCCESSFUL OR NOT. 0 = SUCCESS;
C          NONZERO=ERROR OR WARNING.
C  POSITION TO THE 50TH BLOCK OF VOLUME 2:
  CALL SETTP(1,1H ,50,1H ,2,0,0,STAT)
C
  IF (STAT .NE. 0) THEN
    PRINT *,'SETTP ERROR: STAT = ',STAT
    CALL ABORT
  ENDIF
C
C  READ THE 50TH BLOCK ON VOLUME 2
C
  READ(1) (BUFF(N),N=1,RECLEN)
C
C  PROCESS THE DATA
C
  END
```

Figure 25. SETTP(3) positioning, example 1

4.1.4.2 Example 2

The following example shows you how to position forward nine blocks relative to the current position, and then read one block.

1. Reserve a tape by using the `rsv`(1) command. In this example, the tape has a device group name of `TAPE` and the number of devices requested is `1`:

   ```
   rsv TAPE 1
   ```

2. Request a tape mount by using the `tpmnt`(1)command. In this example, the tape is an old one with an IBM standard label, a volume identifier of `SCRNL`, a device group name of `TAPE`, a record format of fixed length, a block size of `64000` bytes, and a path name of `fort.1`:

   ```
   tpmnt -o -l sl -v SCRNL -g TAPE -F F -b 64000 -p fort.1
   ```

3. Use the `assign`(1) command to specify that file `fort.1` is a tape.

   ```
   assign -s tape f:fort.1
   ```

4. Compile and run the program shown in Figure 26:

```
PROGRAM TPOS
     IMPLICIT INTEGER (A - Z)
     DIMENSION BUFF(8000)
     RECLEN=8000
     NBLKS=200
C
     DO 500 I=1,NBLKS,10
C
C    SKIP 9 BLOCKS
C
     CALL SETTP(1,1H+,9,0,0,0,1,STAT)
     IF (STAT .NE. 0) THEN
       PRINT *,'SETTP ERROR: STAT = ',STAT
       CALL ABORT
     ENDIF
C
C    READ THE 10TH BLOCK
C
     READ(1) (BUFF(N),N=1,RECLEN)
C
C    PROCESS THE DATA
C
 500 CONTINUE
     END
```

Figure 26. SETTP(3) positioning, example 2

4.1.5 Reading and writing tapes containing foreign data

This section shows you how to convert foreign data to Cray Research data or convert Cray Research data to foreign data. Currently, the Fortran libraries support the translation and conversion of IBM, VAX/VMS, NOS/VE, IEEE, and CDC foreign data types. The two methods of foreign data conversion are the following:

- Explicit
- Implicit

4.1.5.1 Converting foreign data explicitly

This section shows you how to convert foreign data explicitly by using Fortran library data conversion routines to read tapes written on foreign computer systems. For a complete list of Cray Research Fortran library data conversion routines, see the *Application Programmer's Library Reference Manual*, Cray Research publication SR–2165.

The following library conversion routines translate data in a foreign format into Cray Research format:

Routine	Description
`IBM2CRAY`(3)	Converts IBM data to Cray format
`IEG2CRAY`(3)	Converts IEEE or generic 32-bit data to Cray format
`NVE2CRAY`(3)	Converts NOS/VE data to Cray format
`VAX2CRAY`(3)	Converts VAX data to Cray format
`CDC2CRAY`(3)	Converts CDC 60-bit data to Cray format

The following library conversion routines translate data in Cray Research format into a foreign format:

Routine	Description
`CRAY2IBM`(3)	Converts Cray data to IBM format
`CRAY2IEG`(3)	Converts Cray format to IEEE or generic 32-bit data
`CRAY2NVE`(3)	Converts Cray data to NOS/VE format
`CRAY2VAX`(3)	Converts Cray data to VAX format
`CRAY2CDC`(3)	Converts Cray data to CDC 60-bit format

With explicit conversion you must specify the type and size of all data conversions before you can set up structures for the converted data; therefore, you must know the type and size of the data originally written on a tape.

Note: Be careful when specifying foreign record translation with the `-F` option on the `assign`(1) command. With most values of the `-F` options, you can only read, write, backspace, and rewind your tape. With the `-F ibm.u,tape`, `-F ibm.vbs,tape`, `-F ibm.vb,tape`, or `-F ibm.v,tape` option you can also use the `-d skipbad` option to request that bad data be automatically skipped. See the `assign`(1) man page for more details on the `-d` option. When you use the `-F bmx` or `-F tape` option, and do not specify any other FFIO layers, you can also use the Fortran tape positioning routines, process bad data, and do end-of-volume processing.

4.1.5.2 Example 1

This example illustrates the handling of data by using library conversion routines:

1. Reserve a tape by using the `rsv`(1) command. In this example, the device group name is `CART` and the number of devices requested is `1`:

   ```
   rsv CART 1
   ```

2. Compile Fortran program `ibmcvt.f`:

   ```
   f90 ibmcvt.f
   ```

3. Load the `ibmcvt.o` file, directing the output to executable file `ibmcvt`:

   ```
   segldr -o ibmcvt imbcvt.o
   ```

4. Request a tape mount by using the `tpmnt`(1) command. In this example, the tape has an IBM standard label, a volume identifier of `ISCSL`, a device group name of `CART`, a block size of `32768` bytes, and a path name of `fort.29`:

   ```
   tpmnt -l sl -v ISCSL -g CART -b 32768 -p fort.29 -n
   ```

5. Use the `assign`(1) command to specify that file `fort.1` is a tape.

   ```
   assign -s tape f:fort.29
   ```

6. Execute `ibmcvt`:

   ```
   ibmcvt
   ```

7. Release the reserved resource:

   ```
   rls -a
   ```

Figure 27 shows the Fortran program, called `ibmcvt.f`:

```
PROGRAM IBMCVT
 INTEGER CRAY2IBM,IBM2CRAY
C
C  REALA and REALB are the arrays that hold the CRAY format numbers.
C  IBMA and IBMB hold the converted IBM data. Note that they are half as
C  large as the CRAY real numbers, because the IBM real format is only 32
C  bits long. The type of the IBM array is not important, because no
C  computations will be performed on them. Only the proper amount of
C  space is important.
C
   REAL REALA(10000)
   REAL REALB(10000)
C
   INTEGER IBMA(5000)
   INTEGER IBMB(5000)
C
   CALL GENERATE(REALA)  ! REAL data is generated and converted to IBM format
C
C  The data produced is converted to IBM internal format and placed in
C  array IBMA. See the man pages for the CRAY2IBM(3) and IBM2CRAY(3)
C  routines.
C
   ISTAT = CRAY2IBM(    2,   ! data type, 2=REAL
   +       10000,   ! number of items to convert
   +        IBMA,   ! ’foreign’ array
   +         0,   ! bit offset in IBMB
   +       REALA,   ! CRAY data
   +         1)   ! stride
   IF (ISTAT.LT.0) STOP ’error 1’  ! Check for conversion error.
C
C  Write the converted data to unit 29. No ’foreign’ assign options should
C  be present on this unit, or the data will be converted twice!
C
   WRITE(29) IBMA
   REWIND 29
C
   READ(29) IBMB         ! Read the IBM format data back from the file.
```

```
C
   ISTAT = IBM2CRAY(   2,   ! data type, 2=REAL
   +      10000,   ! number of items to convert
   +       IBMB,   ! ’foreign’ array
   +         0,   ! bit offset in IBMB
   +      REALB,   ! CRAY data
   +         1)   ! stride
   IF (ISTAT.LT.0) STOP ’error 1’ ! Check for convert error.
C
   CALL PROCESS(REALB)
   END
```

Figure 27. Converting data to an IBM format

The data generated on Cray Research systems, converted to IBM format by the data conversion routines, is altered to fit the storage capabilities of IBM computer systems because the system storage limits of precision have been exceeded. The following list describes the way data is handled when it exceeds the limits of precision for IBM computer systems:

- Cray Research positive numbers (if they exceed IBM computer systems’ limits of precision) are assigned the largest positive values allowed for integer or floating-point real numbers that can be expressed on IBM systems.
- Cray Research negative numbers with absolute values that exceed IBM computer systems’ limits of precision are assigned the most negative value that can be expressed on IBM systems.
- Cray Research positive and negative numbers approaching zero, which are more precise than the smallest positive or negative fractional value that can be expressed on IBM computer systems, are assigned a value of 0.

If the data is generated on IBM computer systems and read on Cray Research systems, you do not need to be concerned with loss of precision in the conversion process. This is true of any computer system that has both a smaller exponent and a smaller mantissa size than that of Cray Research systems.

4.1.5.3 Example 2

The Fortran program fragment shown in Figure 28 illustrates how you might read an unknown number of records that are all of the same length and contain the same data type:

```
   .
       .
       .
       ICT = 0
10     CONTINUE
       READ(29,ERR=20,END=30) (ARRAY(I),I=1,LENGTH)
       ICT = ICT + 1
       CALL IBM2CRAY(ITYPE,LENGTH,ARRAY,0,NARRAY,1)
       CALL PROCESS(NARRAY)
       GOTO 10

20     print *,’ Error in read on record ’,ICT+1
       STOP ’error’

30     print *’ End of File encountered on record ’,ICT+1
       .
       . (continue program)
       .
```

Figure 28. Reading an unknown number of records

4.1.5.4 Example 3

The partial Fortran program shown in Figure 29 reads data records consisting of floating-point, integer, or character data (or any combination of these):

```
C  Read in IBM data records.
C
     READ(29,END=20,ERR=15) (A(I),I=1,NUM)
     READ(29,END=30,ERR=25) (B(I),I=1,CNT)
     READ(29,END=40,ERR=35) (C(I),I=1,NUMBER)
     READ(29,END=50,ERR=45) (D(I),I=1,COUNT)
C
C Now convert IBM data to Cray format. The fields of IBM2CRAY are:
C    Field 1: Type code. Indicates format of IBM data.
C          2: Number of data items to convert.
C          3: IBM array from which you are converting.
C          4: Bit number to begin conversion.
C          5: Array to contain the converted data.
C
     CALL   IBM2CRAY(7,NUM,A,1,SPR)       ! 7 indicates short integer
     CALL   IBM2CRAY(1,CNT,B,1,INT)       ! 1 indicates long integer
     CALL   IBM2CRAY(2,NUMBER,C,1,DPR)    ! 2 indicates short real
     CALL   IBM2CRAY(6,COUNT,D,1,CHR)     ! 6 indicates char (EBCDIC)
```

Figure 29. Reading mixed data types

Note: To correctly convert the data to be read, you must know the data types of the contents of the tape.

4.1.5.5 Converting foreign data implicitly

This section shows you how to translate the blocking structures and convert foreign data implicitly to Cray Research data and vice versa by using the `assign`(1) command. Currently, implicit conversion supports the translation and conversion of IBM, VAX/VMS, CDC (60-bit), NOS/VE, ULTRIX, and IEEE foreign data types.

The following example shows you how to read foreign data with Fortran programs using the `assign`(1) command:

1. Reserve a tape by using the `rsv`(1) command. In this example, the device group name is `TAPE` and the number of devices requested is `1`:

   ```
   rsv TAPE 1
   ```

2. Compile and load Fortran program `impread.f`, directing the output to the executable file `impread`:

   ```
   f90 -o impread impread.f
   ```

3. Request a tape mount by using the `tpmnt`(1) command. In this example, the tape has an IBM standard label, a volume identifier of `SCRSL`, a device group name of `TAPE`, a block size of `32768` bytes, and a path name of `fort.29`:

   ```
   tpmnt -l sl -v SCRSL -g TAPE -b 32768 -p fort.29
   ```

4. Request implicit data conversion. In this example, specify (by using the `-F` option) that the blocking of the tape is in the IBM `VBS` format, specify (by using the `-N` option) that the numeric data to be converted is IBM data, and use the `assign`(1) command to specify that file `fort.29` is a tape:

   ```
   assign -F ibm.vbs,tape -N ibm f:fort.29
   ```

5. Execute `impread`:

   ```
   impread
   ```

6. Release the reserved resource:

   ```
   rls -a
   ```

Figure 30 shows the Fortran program, called `impread.f`:

```
PROGRAM IMPREAD
      INTEGER INT
      REAL RL(4)
      COMPLEX COM(3)
C
C     THIS PROGRAM READS DATA IN FROM THE DATA FILE AND PRINTS IT.
C     THE RECORD FORMAT OPTION (-F) AND DATA CONVERSION OPTIONS (-N)
C     MUST BE SPECIFIED IN THE ASSIGN COMMAND.
C
C     THE RUN-TIME LIBRARY WILL DEBLOCK THE FOREIGN RECORD(S) AND
C     CONVERT THE DATA ITEMS AS REQUESTED IN THE ASSIGN COMMAND.
C
      DO 10 I=1,10
        READ(29) INT,RL,COM
        PRINT *,’REC=’,INT
        PRINT *,’ RL=’,RL
        PRINT *,’COM=’,COM
 10   CONTINUE
      END
```

Figure 30. Converting foreign data

4.1.6 Using the bad data recovery routines

This section shows you how to use the bad data recovery routines, which allow you to skip or accept bad data. These routines check the status of the I/O function. If an error has occurred, these routines call a Fortran error-handling routine. For more information on the Fortran error handling routines, see the *Application Programmer’s Library Reference Manual*, Cray Research publication SR–2165.

The Fortran error-handling routines available are as follows:

Routine	Description
`SKIPBAD`(3)	Skips bad data

`ACPTBAD`(3)	Makes bad data available

4.1.6.1 Example 1

Figure 31 shows the following example, called `skipbdxmp`. This example illustrates how to use the Fortran error-handling routine `SKIPBAD`, which skips bad data on the read operation:

```
PROGRAM SKIPBDXMP
      IMPLICIT INTEGER (A-Z)
      PARAMETER (MAXSIZ=50000)
      DIMENSION BUFFER(MAXSIZ),UDA(512)

      NUMBLKS=1000000
      NWORDS=4096

      DO 5000 NBLK=1,NUMBLKS
      CALL READ(99,BUFFER,NWORDS,STATUS)
      IF(STATUS .EQ. 0) GO TO 5000

      IF(STATUS .EQ. 4) THEN
        PRINT*,’****PARITY ERROR ON READ AT RECORD’,NBLK
        NPAR=NPAR+1
        GO TO 2500
      ENDIF

      IF(STATUS .EQ. 2) THEN
        PRINT*,’*****END OF FILE DETECTED,RECORDS=’,NBLK
        STOP
      ENDIF

      IF (STATUS .NE. 0) THEN
        PRINT*,’UNEXPECTED RETURN CODE FROM READ=’,STATUS
        CALL ABORT
      ENDIF

2500 CALL SKIPBAD (99,BLKS, TERMCND)
PRINT*,’SKIPBAD-BLOCKS SKIPPED’,BLKS
      PRINT*,’STATUS EQUALS’,TERMCND

      IF (TERMCND .EQ. 0) GO TO 5000
      IF (TERMCND .EQ. 1) THEN
        PRINT*,’****END OF FILE DETECTED, RECORDS READ=’,NBLK
        PRINT*,’****NUMBER OF PARITY ERRORS=’,NPAR
        STOP
      ENDIF
```

```
     IF (TERMCND .LT. 0) THEN
        PRINT*,'****NOT ON A RECORD BOUNDARY,ABORTING'
        CALL ABORT
      ENDIF
5000 CONTINUE
      STOP
      END
```

Figure 31. Using the `SKIPBAD`(3) routine

4.1.6.2 Example 2

Figure 32 shows you how to use the Fortran error-handling routine called `ACPTBAD`(3), which accepts bad data on the read operation):

```
PROGRAM ACPTBAD

      IMPLICIT INTEGER (a-z)
      PARAMETER (NUMBLKS=10000)
      PARAMETER (MAXSIZE=50000)
      PARAMETER (RECLEN=4096)
      DIMENSION BUFFER(MAXSIZE),UDA(MAXSIZE)

      NPAR = 0
      DO 5000 NBLK=1,NUMBLKS
         NWORDS=RECLEN
         CALL READ(1,BUFFER,NWORDS,STATUS)
         IF (STATUS .EQ. 0) GO TO 5000
IF (STATUS .EQ. 4) THEN
           PRINT *,’***PARITY ERROR ON READ AT RECORD ’,NBLK
           NPAR = NPAR+1
           GO TO 2500
      ENDIF

      IF ((STATUS .EQ . 2) .OR. (STATUS .EQ. 3))THEN
         PRINT *,’END OF FILE/DATA DETECTED, RECORDS= ’,NBLK
         STOP ’COMPLETE’
      ENDIF

      IF (STATUS .NE. 0)THEN
         PRINT *,’UNEXPECTED RETURN CODE FROM READ = ’,STATUS,NBLK
         CALL ABORT
      ENDIF
```

```
2500 CALL ACPTBAD(1,UDA,CNT,TERMCND,UBC,MAXSIZE)

C
C     BUILD UP USER RECORD
C
      IX = 0
      DO 3500 I = (NWORDS+1),(NWORDS+CNT)
         IX=IX+1
         BUFFER(I)=UDA(IX)

3500  CONTINUE

      IF (TERMCND .LT. 0)THEN
         PRINT *, 'END OF RECORD NOT REACHED'
      ENDIF
      IF (TERMCND .EQ. 1)THEN
         PRINT *,'**** END OF FILE DETECTED,RECORDS = ',NBLK
         PRINT *,'**** NUMBER OF PARITY ERRORS = ',NPAR
      ENDIF
5000 CONTINUE
      END
```

Figure 32. Using the ACPTBAD(3) routine

4.1.6.3 Example 3

Figure 33 shows you how to reserve, mount, and release tapes from inside your Fortran program, using the ISHELL(3) routine. If you use this routine, you must make sure that the tape file is closed before the tape is released. Use the Fortran CLOSE statement to close a file.

```
PROGRAM TP1
 DIMENSION IBUF(500)
 INTEGER ISHELL

C RESERVE A CART DEVICE

 ISTAT = ISHELL(’rsv CART 1’)
 IF (ISTAT.NE.0) GOTO 100

C REQUEST MOUNT OF DESIRED CART

 ISTAT = ISHELL(’tpmnt -l al -v ISCAL -p fort.10 -g CART -n’)
 IF (ISTAT.NE.0) GOTO 200

C  WRITE TO THE TAPE. YOU MUST HAVE PREVIOUSLY USED THE ASSIGN
C COMMAND TO IDENTIFY UNIT 10 AS A TAPE.

 DO 10 I = 1,500
 WRITE(10)IBUF
 10 CONTINUE

C BEFORE RELEASING THE TAPE, THE FILE MUST BE CLOSED.
 CLOSE(10)

C RELEASE THE TAPE, BUT KEEP THE CART RESOURCE.

 ISTAT = ISHELL(’rls -k -p fort.10’)
 IF (ISTAT.NE.0) GOTO 300

C  REQUEST MOUNT OF ANOTHER TAPE

 ISTAT =ISHELL(’tpmnt -l sl -v ISCSL -p fort.10 -g CART -n’)
 IF (ISTAT.NE.0) GOTO 200
C WRITE TO TAPE

 DO 20 I = 1,500
 WRITE(10) IBUF
 20
```

```
C CLOSE THE FILE AND RELEASE ALL TAPE RESOURCES

 CLOSE(10)
 ISTAT = ISHELL('rls -a')
 IF (ISTAT.NE.0) GOTO 300
 STOP

100 PRINT *,'RSV FAILED'
 STOP

 200 PRINT *,'TPMNT FAILED'
 ISTAT = ISHELL('rls -a')
 STOP

 300 PRINT *,'RLS FAILED'
 PRINT *,'ISTAT = ',ISTAT
 STOP
 END
```

Figure 33. Using the `ISHELL`(3) routine

To execute the preceding program, `tp1`, type the following:

```
f90 -o tp1 tp1.f
assign -s tape f:fort.10./tp1
```

4.1.7 Using end-of-volume processing requests

This section describes user end-of-volume (EOV) processing from a Fortran program. For information about user EOV processing from a C program, see Chapter 5, page 81.

Normally, volume switching is handled by the tape subsystem and is transparent to you. However, when user EOV processing is requested, you gain control at the end-of-tape and your program may perform special processing. For more information on the Fortran interface routines used in EOV processing, see the *Application Programmer's Library Reference Manual*, Cray Research publication SR–2165. The library interface routines for EOV processing from a Fortran program are as follows:

Routine	Description
SETSP(3)	Enables or disables EOV processing
STARTSP(3)	Starts special tape processing
ENDSP(3)	Ends special tape processing
CHECKTP(3)	Checks tape position
CLOSEV(3)	Closes volume and mounts next volume specified in the volume identifier list

When using EOV processing for online tape files on the tape subsystem, make sure that data is flushed from the library and system buffers before calling certain routines as discussed in the following. Failure to flush the buffers and check for EOV can result in lost data at the end of the tape volume.

To instruct the system to perform EOV processing, call the SETSP routine with the appropriate parameter set to ON after a tape file is opened.

Check for EOV by calling the CHECKTP macro. To test whether a tape is at EOV, you must call CHECKTP after each WRITE, ENDFILE, or READ operation. In addition, for an output dataset, call CHECKTP after each GETTP or GETPOS call to see if EOV was encountered.

For output datasets, you should also ensure that the library and system have flushed their buffers, and then test whether the tape is at EOV, before issuing any of these statements:

CLOSE

REWIND

BACKSPACE

and before calling any of these routines:

SETTP(3)

SETPOS(3)

CLOSEV(3)

SETSP(3) (OFF)

To flush the buffers, call GETTP(3) with the SYNCH parameter set to ON. Then call CHECKTP(3) to see if EOV was reached.

4.1.7.1 Example 1

The following example shows you how to use EOV processing by using the library interface routines:

1. Reserve a tape by using the `rsv`(1) command:

   ```
   rsv CART 1
   ```

2. Compile and load Fortran program `teov.f`, directing the output to executable file `teov`:

   ```
   f90 -o teov teov.f
   ```

3. Request a tape mount by using the `tpmnt`(1) command. In this example, the tape has no label, a path name of `fort.9`, and volume identifiers of `x` and `y`:

   ```
   tpmnt -l nl -P fort.9 -v x:y -g CART -n
   ```

4. Use the `assign`(1) command to specify that file `fort.9` is a tape:

   ```
   assign -s tape f:fort.9
   ```

5. Execute `teov`:

   ```
   teov
   ```

6. Release the reserved resource:

   ```
   rls -a
   ```

Figure 34 shows the Fortran program, called `teov.f`:

```
 PROGRAM TEOV
      IMPLICIT INTEGER(A-Z)
      PARAMETER (BLKLEN = 512)

C     MAXREC = total number of records to be written
      PARAMETER (MAXREC = 10000)
      DIMENSION BLK(BLKLEN)
      INTEGER IPA(45)

C     Set up for special EOV processing
      CALL SETSP(9, 1, ISTAT)
      IF (ISTAT .NE. 0) GOTO 20

C     Write MAXREC records. Check for end-of-volume after each write.
C     If end-of-volume is detected, call the subroutine EOVPROC
      DO 10 I = 1, MAXREC
        WRITE (9)BLK
        CALL CHECKTP(9, ISTAT, ICBUF)
        IF (ISTAT. EQ. 0) THEN
C       At EOV
        CALL EOVPROC()
        ENDIF
10    CONTINUE
C     We have written all of the records. Call GETTP with the
C     sync parameter set to flush the library’s and system’s buffers.
      CALL GETTP(9, 40, IPA, 1, IREPLY)
      IF (IREPLY.NE. 0)GOTO 20
      CALL CHECKTP(9, ISTAT, ICBUF)
      IF (ISTAT. EQ. 0) THEN
C       At EOV
        CALL EOVPROC()
      ENDIF
C     Stop end-of-volume processing
      CALL ENDSP (9, ISTAT)
      IF (ISTAT .NE. 0)GOTO 20
C     Close the file
      CLOSE(9)
C     STOP
20    PRINT *,’ERROR’
      END
```

```
SUBROUTINE EOVPROC()
      DIMENSION TBLK(512)
      DIMENSION HBLK(512)

C     Start special processing at eov.
      CALL STARTSP(9, ISTAT)
      IF (ISTAT. NE. 0) GOTO 20

C     Write a special block at the end of the tape
C     and close the volume.
      WRITE (9) TBLK
      CALL CLOSEV(9, ISTAT)
      IF (ISTAT. NE. 0) GOTO 20

C     Write a special block at the beginning of the next tape.
      WRITE (9) HBLK

C     Stop special processing
      CALL ENDSP (9, ISTAT)
      IF (ISTAT. NE. 0) GOTO 20
      RETURN
20    PRINT *,’ERROR’
      STOP
      END
```

Figure 34. Using Fortran library routines for EOV processing

4.1.7.2 Example 2

The following example illustrates EOV processing when writing a tape file. The program writes until end-of-volume is reached. It then reads the last two blocks on the first volume and the blocks buffered in the IOS, and writes these on the second tape volume:

1. Reserve a tape by using the `rsv`(1) command:

   ```
   rsv CART 1
   ```

2. Compile and load Fortran program `teov2.f`, directing the output to executable file `teov2`:

   ```
   f90 -o teov2 teov2.f
   ```

3. Request a tape mount by using the `tpmnt`(1) command. In this example, the tape has no label, a path name of `fort.9`, and volume identifiers of `VOL1` and `VOL2`:

   ```
   tpmnt -l nl -v VOL1:VOL2 -p fort.9 -g CART -r in -n -T
   ```

4. Use the `assign`(1) command to specify that file `fort.9` is a tape.

   ```
   assign -s tape f:fort.9
   ```

5. Execute `teov2`:

   ```
   teov2
   ```

6. Release the reserved resources:

   ```
   rls -a
   ```

Figure 35 shows the Fortran program, called `teov2.f`:

```
 PROGRAM TEOV2

c    Example of EOV processing. Assumes that fort.9 is a tape file

     IMPLICIT INTEGER(A-Z)
     PARAMETER (BLKLEN = 512, PALEN = 30)
     DIMENSION BLK(BLKLEN), PA(PALEN)

c    Set up for special EOV processing.

     CALL SETSP(9,1,ISTAT)
     IF (ISTAT.NE.0) GOTO 100

c    Fill the first volume. CHECKTP returns a status that indicates
c    whether end-of-volume has been reached.

10   CONTINUE
     WRITE(9)BLK
     CALL CHECKTP(9,ISTAT,ICBUF)
     IF (ISTAT.LT.0) GOTO 10

c    Determine number of blocks buffered in the IOS.
C    Note that we do not request synch here.

     CALL GETTP(9, PALEN, PA, 0, ISTAT)
     IF (ISTAT.NE.0) GOTO 100

c    Start special processing
     CALL STARTSP(9,ISTAT)
     IF (ISTAT.NE.0) GOTO 100

c    Backspace 2 blocks. Read these 2 blocks from tape + blocks from
c    the IOS + blocks in the library buffer and store them in a
c    temporary file (fort.10)

     BACKSPACE(9)
     BACKSPACE(9)
     NBLK=PA(12)+2+PA(11)            ! blocks in IOS + 2 from tape
                                  ! + blocks in library buffer
     DO 20 I = 1,NBLK
        READ(9)BLK
```

```
20   WRITE(10)BLK
c    Backspace 2 blocks before closing the volume, because these
c    2 blocks will be rewritten on the second volume.

     BACKSPACE(9)
     BACKSPACE(9)

c    The programmer wants to write a tape mark at EOV
     ENDFILE(9)

c    Close the volume and request mount of the next volume

     CALL CLOSEV(9,ISTAT)
     IF (ISTAT.NE.0) GOTO 100

     CALL ENDSP(9,ISTAT)            ! stop special processing
     IF (ISTAT.NE.0) GOTO 100

c    Disable special processing

     CALL SETSP(9, 0, ISTAT)
     IF (ISTAT.NE.0) GOTO 100

     REWIND(10)                    ! rewind temporary file

c    Write the blocks in fort.10 onto the second volume

     DO 30 I = 1,NBLK
        READ(10)BLK
30   WRITE(10)BLK

     CLOSE(9)                ! close the file
     STOP
100  CALL ABORT()
     END
```

Figure 35. Using EOV processing when writing a file

4.1.7.3 Example 3

The following example shows how to use EOV processing to detect the end-of-volume when reading a multivolume file.

1. Reserve a tape by using the `rsv`(1) command:

   ```
   rsv
   ```

2. Compile and load the Fortran program `eovr.f` directing the output to executable file `eovr`:

   ```
   f90 -o eovr eovr.f
   ```

3. Request a tape mount by using the `tpmnt`(1) command. In this example, the file has a standard label, a path name of `fort.10`, and volume identifiers of `x` and `y`:

   ```
   tpmnt -l sl -p fort.10 -v x:y
   ```

4. Use the `rsv`(1) command to specify that file `fort.10` is a tape:

   ```
   assign -s tape f:fort.10
   ```

5. Execute `eovr`:

   ```
   eovr
   ```

6. Release the reserved resource:

   ```
   rls -p fort.10
   ```

Figure 36 shows the Fortran program, called `eovr.f`:

```
  PROGRAM EOVR

      IMPLICIT INTEGER (A-Z)
      PARAMETER (BLKLEN=4096)

      DIMENSION BLK(BLKLEN)

      CALL SETSP(10,1,ISTAT)
      IF (ISTAT.NE.0) THEN
        PRINT *,’BAD STATUS FROM SETSP ’,ISTAT
        GOTO 100
      ENDIF

c     Read until we get to the end of a volume or the end of data

10    CONTINUE

      IC = BLKLEN
      CALL READ(10, BLK, IC, ISTAT)
      IF ((ISTAT.EQ.2).OR.(ISTAT.EQ.3)) THEN
        PRINT *,’END OF FILE/DATA’
        GOTO 100
      ELSEIF (ISTAT.NE.0) THEN
        PRINT *,’UNEXPECTED STATUS FROM READ ’,ISTAT
        GOTO 100
      ENDIF

c     Check for end of volume

      CALL CHECKTP(10,REPLY,CB)
      IF (REPLY .LT. 0 ) GOTO 10
      IF (REPLY .GT. 0 ) THEN
        PRINT *,’UNEXPECTED STATUS FROM CHECKTP ’,REPLY
        GOTO 100
      ENDIF

c     At EOV. Start special processing, and call CLOSEV to mount the
c     next tape
```

```
CALL STARTSP(10,ISTAT)
      IF (ISTAT.NE.0)THEN
        PRINT *,'BAD STATUS FROM STARTSP ',ISTAT
        GOTO 100
      ENDIF
      CALL CLOSEV(10,ISTAT)
      IF (ISTAT.NE.0)THEN
        PRINT *,'BAD STATUS FROM CLOSEV ',ISTAT
        GOTO 100
      ENDIF
      CALL ENDSP(10,1,ISTAT)
      IF (ISTAT.NE.0) THEN
        PRINT *,'BAD STATUS FROM ENDSP ',ISTAT
        GOTO 100
      ENDIF
      GOTO 10

c     Disable EOV processing
100   CALL SETSP(10,0,ISTAT)
      CLOSE(10)
      END
```

Figure 36. Using EOV processing when reading a multivolume file

4.2 ER90 tape processing

You can access ER90 devices through Fortran. Unformatted I/O is currently supported. You can select either the byte-stream mode or block mode of the device.

Note: The ER90 format is not available on systems that run the UNICOS/mk operating system or that have GigaRing support.

In byte-stream mode, two processing classes are available:

- Pure data mode
- COS blocking

To select the processing class, use the `assign`(1) command.

In block mode, select the FFIO tape layer by using the following command:

```
assign -F tape
```

With this processing class, each Fortran record corresponds to a block. Because ER90 devices require that each block be the size specified by the `-b` option of the `tpmnt`(1) command, all Fortran records must be the same size. An exception to this rule is the last record written before a tape mark or the end-of-file. This record may be smaller than the size specified by the `-b` option. When you choose this processing class, EOV processing routines, user tape marks, and the `SETTP`(3) routine are available.

4.2.1 Using pure data mode

In pure data mode, no record control words are written to the file. This indicates that the user must know the size of the records being read. Reading, writing, and rewinding are allowed in this mode.

Use the `assign`(1) command to select this mode, as follows:

```
assign -F er90 assign.object
```

Pure data mode does not support EOV processing, `SETTP`(3), `SKIPF`(3), and multiple `ENDFILES`. `GETTP` is allowed, but the meaning of some fields that are returned in the information array differs from that returned when using round or cartridge tapes and the following `assign`(1) command:

```
assign -[F,s] [tape,bmx]
```

`GETPOS`(3) and `SETPOS`(3) are supported when using this processing class. The `len` parameter for these routines must be at least 4. The values returned by `GETPOS`(3) in the `pa` array contain device specific information; it may be used in a subsequent call to `SETPOS`(3). For ER90 files, the information returned does not include the volume serial number (VSN) or partition information. Before using `SETPOS`(3), verify that the correct VSN and partition is in position.

When using round or cartridge tapes, each Fortran record corresponds to a physical tape block. When using the ER90 device in byte-stream mode, each byte is considered a block. Therefore, the `ipa(10)`, `ipa(11)`, and `ipa(12)` fields return byte counts.

When the file is assigned with `-F er90`, each Fortran read or write results in one or more system calls. The `bufa` layer may be used to provide asynchronous buffering, potentially reducing the number of system calls and improving performance. It may be combined with the `er90` layer as follows:

```
assign -F bufa,er90 assign.object
```

The following example illustrates the use of pure data mode with the ER90 device.

1. Reserve a device by using the`rsv`(1) command:

   ```
   rsv ER90
   ```

2. Compile the Fortran program `tpwr1.f`v, resulting in the relocatable file `ctpwr1.o`:

   ```
   f90 tpwr1.f
   ```

3. By default, the ER90 flexible file I/O (FFIO) layer is disabled. It can be enabled by the system administrator, or by specifying the `ff_er90` loader directives file.

   Load the relocatable file `tpwr1.o`, directing the binary output to the executable file `tpwr1` using the following `segldr`(1) command:

   ```
   segldr -o tpwr1 tpwr1.o -j ff_er90
   ```

4. Request a tape mount by using the `tpmnt`(1) command. In this example, the tape has no label, a path name of `fort.1`, and a volume identifier of `00011`:

   ```
   tpmnt -p fort.1 -l nl -v 00011 -g ER90
   ```

5. Use the `assign`(1) command to specify that file `fort.1` is an ER90 file with asynchronous buffering:

   ```
   assign -F bufa,er90 fort.1
   ```

6. Execute `tpwr`:

   ```
   ./tpwr1
   ```

7. Release the reserved resources:

   ```
   rls -a
   ```

Figure 37 shows the Fortran program, called `tpwr1.f`:

```
program tpwr1
      integer buf(2000)
      integer ipa(4)
c     write 100 records
      do 10 i = 1,100
        do 5 j = 1,2000
          buf(j) = i
5     continue
      write(1)buf
      if (i.eq.50)then
        call getpos(1, 4, ipa, istat)
        if (istat.ne.0)then
          print *,’bad stat ’, istat
          stop ’error’
        endif
10    endif
      rewind(1)

c     read the records and verify
      do 20 i = 1,100
        read(1) buff
        do 15 j = 1,2000
          if (buf(j).ne.i)then
            print *,’bad data ’, buf(j),i
            stop ’error’
          endif
15    continue
20    continue
c     now position back to the point where we did the getpos
      call setpos(1, 4, ipa, istat)
      if (istat.ne.0)then
        print *,’bad stat ’,istat
         stop ’error’
      endif
      read(1)buf
      if (buf(1).ne.51)then
        print *,’bad data after setpos ’, buf(1)
      endif
      end
```

Figure 37. Using pure data mode

4.2.2 Using COS blocking mode

In COS blocking mode, users can read, write, rewind, and backspace.

Use the `assign`(1) command to select COS blocking for the ER90, as follows:

`assign -F cos,er90` *assign.object*

The COS blocking mode does not support EOV processing, `SKIPF`(3), `CLOSEV`(3), `SETTP`(3), and concatenated tape files. `GETTP`(3) is allowed, but some of the fields that are returned in the information array differ from those returned when using round or cartridge tapes. The COS blocking layer buffers data, and it is not included in the value returned by `GETTP` in `ipa(11).`

`GETPOS`(3) and `SETPOS`(3) are supported when using this processing class. The `len` parameter for these routines must be 6 or higher. The values returned by `GETPOS` in the `pa` array contain device specific information.

The following example illustrates the use of COS blocking mode with the ER90 device.

1. Reserve a device by using the `rsv`(1) command:

   ```
   rsv ER90
   ```

2. Compile the Fortran program `tpwr2.f`, resulting in the relocatable file `tpwr2.o`:

   ```
   f90 tpwr2.f
   ```

3. By default, the ER90 flexible file I/O (FFIO) layer is disabled. It can be enabled by the system administrator, or by specifying the `ff_er90` loader directives file.

   Load the relocatable file `tpwr2.o`, directing the binary output to the executable file `tpwr1` using the following `segldr`(1) command:

   ```
   segldr -o tpwr2 tpwr2.o -j ff_er90
   ```

4. Request a tape mount by using the `tpmnt`(1) command. In this example, the tape has no label, a path name of `fort.1`, and a volume identifier of `00011`:

   ```
   tpmnt -p fort.1 -l nl -v 00011 -g ER90
   ```

5. Use the `assign`(1) command to specify that file `fort.1` is an ER90 file:

   ```
   assign -F er90 fort.1
   ```

6. Execute `tpwr`:

   ```
   ./tpwr2
   ```

7. Release the reserved resources:

   ```
   rls -a
   ```

Figure 38 shows the Fortran program `tpwr2.f`:

```
 program tpwr2
      real rbuf(2000)
c     write 100 records
      do 10 i = 1,100
        do 5 j = 1,2000
          rbuf(j)=i
5     continue
      write(1) rbuf
10    continue
      rewind(1)
c     read the records and verify
      do 20 i = 1,100
         read(1) rbuf
        do 15 j=1,2000
          if (rbuf(j).ne.i)then
            print *,' bad data ', rbuf(j),i
            stop
        endif
15    continue
20    continue
      end
```

Figure 38. Using COS blocking mode