CERN Program Library Long Writeup I101

EPIO

Experimental Physics Input Output Package

User's Guide

Application Software and Databases Group

Computing and Networks Division

CERN Geneva, Switzerland

Requests for information should be addressed to:

```
CERN Program Library Office
CERN-CN Division
CH-1211 Geneva 23
Switzerland
Tel.  +41 22 767 4951
Fax.  +41 22 767 7155
Bitnet:  CERNLIB@CERNVM
DECnet:  VXCERN::CERNLIB (node 22.190)
Internet:  CERNLIB@CERNVM.CERN.CH
```


Contact Person:	Ian McLaren /CN	(mclareni@cernvm.cern.ch)
Technical Realization:	Michel Goossens /CN	(goossens@cernvm.cern.ch)

Edition – December 1993

History

Hans Grote/SL was one of the original two authors of the EPIO package.

The EPIO offline package is intended to be used for all data (at least on tape) in an experiment, in such a way that from the raw data tape to the DST, the tape (or file) format is identical. This makes the transport of data between computers trivial, and simplifies the task of passing the files or tapes at different stages of the production chain through any other part of the production chain.

The package has been developed at CERN as a part of `packlib`.

Version 1.66 of EPIO introduced a standard Fortran 77 version which is the base for the Unix versions.

We would like to encourage all users to contact us in case:

- they detect mistakes in the manual or the package,
- they would like to see something to be added to the manual or described in more detail,
- they would like to see features added to the package.

Remember that your collaboration in this is vital, and that the poor man in (Saclay, Rome, Vienna, Hamburg, etc.) may desperately be trying to find and fix a bug that you have just corrected, but told nobody (and vice versa, of course).

Preliminary remarks

This manual serves at the same time as a **Reference manual** and as a **User Guide** for the EPIO package.

In this manual examples are in `monotype face` and strings to be input by the user are `underlined`. In the index the page where a routine is defined is in **bold**, page numbers where a routine is referenced are in normal type.

In the description of the routines a * following the name of a parameter indicates that this is an **output** parameter. If another * precedes a parameter in the calling sequence, the parameter in question is both an **input** and **output** parameter.

This document has been produced using LaTeX [1] with the `cernman` style option, developed at CERN. A compressed PostScript file `epio.ps.Z`, containing a complete printable version of this manual, can be obtained from any CERN machine by anonymous ftp as follows (commands to be typed by the user are underlined):

```
ftp asis01.cern.ch
Trying 128.141.201.136...
Connected to asis01.cern.ch.
220 asis01 FTP server (Version 6.10 ...) ready.
Name (asis01:username): anonymous
Password: your_mailaddress
230 Guest login ok, access restrictions apply.
ftp> cd cernlib/doc/ps.dir
ftp> binary
ftp> get epio.ps.Z
ftp> quit
```

Table of Contents

List of Tables

Chapter 1: Introduction

1.1 Usage

The package is designed as to make almost all features of the very flexible EP format (see section 5.6) available to the user. One can and should however, use it in as simple a way as possible. In the following, we give a series of applications ranging from the simplest use to the most complex one. It should be stressed that for the user wanting to use the package as a black box,

– no knowledge of the actual format is required
– the only routines he has to use and understand are

```
EPINIT
EPREAD(EPFHDR,EPFRD)
EPOUTS
EPRWND
EPCLOS
EPEND
```

Obviously, the more advanced user will have to learn more.

After the examples, we give a description of all routines in the package, and of the utility routines going with it (which are, however, independent of the package and are intended to fulfil the most common user needs).

While it should not concern most users, the following overview of the format might help to clarify some problems. The basic "units" of information are 16-bit words. However, as of version 1.56, all information including the physical header, maybe given in 32-bit words. The header of each physical record or block is usually 12 words. The user's logical data (or events) can be specified by the user to be in "units" of 16 or 32-bit words. Each logical record has a header (normally 4 units long) which specifies the record length and header length in "units" and other general identifiers. However, irrespective of the value of "unit" the user can treat the logical data part of the record as 16-bit, 32-bit or packed quantities, by specifying different "MODES" to `EPREAD` and `EPOUTS`, etc. Packed quantities are applicable to other types of data, for example CDC 60-bit words or 24-bit Camac, but the user has to look after the unpacking himself. The complete description of the format can be found in section 5.6.

It should be stressed that the package does not perform automatic number format conversion, but only packs and unpacks (16 and 32-bit words). However, there are conversion routines from and to IBM floating and integer formats and from and to ASCII. To simplify the portability of data between different computer types, 32-bit headers are written in IBM format.

Users are invited to study the examples carefully concerning initialisation, termination, and definition of buffer sizes.

1.2 Usage on UNIX

The Unix version is restricted to fixed length blocks with full padding which is in any case the EPIO default. If a different blocksize than the EPIO default (1800 16-bit words) is required, it must be specified correctly in status word 1 by a call to `EPSETW`, e.g. for a blocksize of 1800 bytes on unit 11

```
CALL EPSETW(11,1,900,IERR).
```

Unfortunately, this means that you can not give an arbitrarily large blocksize to allow your program to read files with different block sizes. Otherwise, existing applications should be portable to Unix workstations. The choice of C or Fortran for the basic I/O can be selected using the status word 33, which is set by the user as detailed in the table below.

Value	Type of I/O	Default filename (`lun=nn`)
0	Fortran sequential	`for0nn`
1	Fortran direct access	`epionn`
2	C using CFIO	`epionn`

By default, status word 33 is set to 2. Although it is the old default, Value 0 is not portable and seems to give more problems. Values 1 and 2 give identical formats on most systems and are portable.
The association with real filenames can be made in several ways:

1. Code the filename in the EPIO routine `EPDEFU` e.g.,

   ```
   CALL EPDEFU(LUN,'epdata/test1',IERR)
   ```

2. Use the UNIX `ln` command before execution, e.g.

   ```
   ln epdata/test epio01
   ```

   and follow it by:

   ```
   CALL EPREAD(1,MODE,NW,IREC,IBUF,IERR)
   ```

3. Rename (`mv`) or copy (`cp`) the file to an EPIO name `epionn`.

1.2.1 Transferring files

EPIO files for use on or created on mainframes should have fixed length blocks. On VAX/VMS you can use the default settings. On CERNVM the user should issue a filedef specifying `RECFM F` or `RECFM U` and the correct blocksize, which most applications are doing already. If not, a correct filedef is

```
FILEDEF IOFILE11 DISK MYEPIO TESTDATA A1 ( RECFM F BLOCK 3600
```

The files can then be transferred using FTP. The most convenient way is to use the ZFTP utility and a good description of a session is given in Computer Newsletter 200. This avoids the need to convert the data file or to modify the EPIO application. The actual transfer is performed by the command `putb`, e.g.-

```
putb myepio.testdata test1 3600
```

If ZFTP is not available on your system then you must use ftp directly specifying the qualifier “binary”. For files created on a workstation for transfer to a mainframe it is essential to create them as a byte stream. Although it is clearly inefficient, conversion utilities between the two formats will be provided.
A typical CERNVM ftp session would be

```
tcpipibm ftp hostnm
 > hostname
 > password
 > cd wdir
 > binary fixed 3600            for a blocksize of 3600 bytes
 > put file.iofile epio11       or > get epio11 file.iofile
 > quit
```

On VAX, the session is similar except that the EPIO file names are of the form `FOR0nn.DAT`.

1.3 Usage on VM/CMS

The VM version uses `IOPACK`, and both packages are available from the standard CERN program library PACKLIB, by using the command

```
CERNLIB
```

The `IOFILEnn` data sets read and written by EPIO must have filemode 4 (i.e. OS simulation) with options `"RECFM U"` and an appropriate blocksize.
For example, to access a disk data set `"RAWDATA DATA"` on logical unit 10 for reading or writing an appropriate FILEDEF might be:

```
FILEDEF IOFILE10 DISK RAWDATA DATA A4 (RECFM U BLKSIZE 3600
```

At present, concatenation of data sets is not supported under VM, but at CERN the FILEDEF option `"APPEND"` is available for disk files.

1.4 Usage on VAX

1.4.1 Disk files

The usage of disk files does not require any particular action, they are connected to logical units in the standard VMS way. If the logical name `FORnnn` is defined then its translation will be used as file name to connect, otherwise the file `FORnnn.DAT` will be used. However, the file name may also be changed at run-time via a call to the routine `EPDEFU`.

1.4.2 Tape files

The current version of EPIO will recognize when it reads or writes a tape instead of disk, provided the device name is of the standard form `MTAn:`
For example: Suppose you want to mount a tape with blocks of length 23040 bytes on unit 0, for the logical unit no. 1:

```
MOUNT /FOR/BLOC=23040/REC=23040 MTAn: LABEL FOR001
```

where `LABEL` is just a place holder. For unlabelled tapes, on the CENTRAL 8600 at CERN the command should read

```
SETUP/BLOC=23040/REC=23040/NOLABEL XUTnnn XUTnnn FOR001
```

for a non labelled tape, where the second XUTnnn is again a place holder. Should the tape be labelled the qualifier `/LABEL=ASCII` or `/LABEL=EBCDIC` is required. If the tape is to be written the qualifier `/WRITE` must be inserted before the `/[NO]LABEL` qualifier. Byte swapping to/from tape may be done automatically when using EPIO, if this is allowed by the drivers' firmware/hardware. Tapes with EBCDIC labels should be positioned correctly using e.g.:

```
SET MAGTAPE/SKIP=FILES:1 FOR001
```

Alternately label groups must be treated by the user by recovering from EPIO error 5. The file name may be changed at run-time via a call to the routine `EPDEFU`.

1.5 Usage on the Apollo/Domain

The Apollo version uses the stream I/O routines available in the system library. The files written by EPIO are of type `REC` (variable record length) but even `UASC` files can be handled provided the logical block length is fixed and the first status word is properly set.

The default file name is `FORnnn.DAT` like on the VAX where nnn is the logical unit number. This can be changed by calling `EPDEFU`.

Tapes are accessed via a magnetic tape descriptor file. The EPIO routines address the file which in turn redirect the I/O operation to the physical device. A magnetic tape descriptor file is created by the command:

```
edmtdesc for011.dat -c -s lab no reo no civ yes spos yes
edmtdesc for011.dat    -s f 1 rf u bl 3600 rl 3600 ascni no
```

This is the case for a standard EPIO unlabelled tape recorded with a blocksize of 3600 bytes. In case of label or of different blocksize the parameters should be changed accordingly. Type `Help EDMTDESC` for more details. The name of the Magnetic tape descriptor file can be different from `FORnnn.DAT` provided the user calls `EPDEFU`.

1.6 Usage on IBM (Wylbur MVS)

The IBM version of EPIO uses the input/output package `IOPACK` (Z300 in the CERN Program Library). DD names for the EPIO units are of the form

```
IOFILEnn    nn = 1 to 99
```

with `DCB=(RECFM=U,BLKSIZE=iiii)`

The default version of EPIO does not allow concatenation of data sets but this can easily be changed by setting status word 25 to non-zero;

```
CALL EPSETW(LUN,25,1,IERR)
```

To access multiple logical units on the same physical unit using `"UNIT=AFF"` requires a `"CALL EPRWND"` in addition to `EPDROP`, so that the unit is properly closed. The details for reading multifile labelled tapes are in the description of `EPREAD`.

1.7 Usage on CDC

The EPIO routines are available from the standard CERN Program libraries, and use FORTRAN BUFFER IN/OUT to perform the I/O.

Magnetic tape files should be accessed specifying record type `RT=U`. EPIO disk file default format on NOSBE:

```
MFA,MFB      RT=S,BT=C
```

1.8 Other Considerations

1.8.1 Recommended Block Sizes

For reasons of compatibility with different word lengths, the block size should be a multiple of 480 bits = 60 bytes. The default block size is 3600 bytes.

In addition, there are restrictions on the recommended physical record sizes for certain devices. In adition, the "standard Fortran" version, on which the UNIX version is based, is restricted to block sizes in multiples of machine words.

1.8.2 Byte swapping

With the status word 27 set to 1 fully automatic byte swapping is the default. This works of course only with the EP format as it requires two special control words in the physical header.

To read EP files created in the `"OLD"` format (with a 6 word physical header) the user must call `EPSETW` to set status word 27 to 0. The user can force EPIO to read his data as new format by setting status word 27 to 2, avoiding peculiar side effects when there are parity errors for example.

Chapter 2: Examples

2.1 Write EPIO file on a 32-bit computer

Floating point numbers are generated and written to unit 20 in the standard EP format using default values. Note the dimension of `IBUF(900)` to accept the default blocksize of 1800 16-bit units. The user's data is stored in the array `A`.

```
      PROGRAM EPEXA1
      DIMENSION IBUF(900),A(300)
      CALL EPINIT
      DO 10  I=1,20
      CALL USER(A,NW)
C
C   ROUTINE -USER- HAS STORED NW VALUES IN A TO BE WRITTEN OUT
C
      CALL EPOUTS(20,3,NW,A,IBUF,IERR)
      IF(IERR.NE.0)STOP 1
   10 CONTINUE
C **********************************************************
C
C                THE FOLLOWING CALL IS ESSENTIAL
C
C **********************************************************
      CALL EPCLOS(20,IBUF,IERR)
C
C   FOLLOWING CALL ONLY NECESSARY ON SOME COMPUTERS (UNIVAC,IBM)
C
      CALL EPEND(20,IBUF,IERR)
      IF(IERR.NE.0)STOP 2
      STOP
      END
```

2.2 Write on a 60-bit computer

Essentially similar to the previous example, a file is created using all defaults. Note the dimension of `IBUF(480)`.

```
      PROGRAM EPEXA2
      DIMENSION IBUF(480),IA(300)
      CALL EPINIT
      DO 10  I=1,30
      NW=10*I
      CALL USER(IA,NW)
C
C    ROUTINE -USER- HAS STORED NW VALUES IN A TO BE WRITTEN OUT.
C    .....
      CALL EPOUTS(20,3,NW,IA,IBUF,IERR)
      IF(IERR.NE.0)STOP 1
   10 CONTINUE
C **********************************************************
C
C                  THE FOLLOWING CALL IS ESSENTIAL
C
C **********************************************************
      CALL EPCLOS(20,IBUF,IERR)
C
C     FOLLOWING CALL ONLY NECESSARY ON SOME COMPUTERS (IBM, UNIVAC)
C
      CALL EPEND(20,IBUF,IERR)
      IF(IERR.NE.0)STOP 2
      STOP
      END
      SUBROUTINE USER(IA,NW)
      DIMENSION IA(NW)
      DO 5 J=1,NW
      IA(J)=J
 5    CONTINUE
      RETURN
      END
```

2.3 Read Long Block Tape Format

This example treats data written with "long" blocks with a maximum size of 14400 bytes. Note the call to `EPSETW` to specify the buffer size correctly for a 32-bit computer.

```
      PROGRAM EPEXA3
      DIMENSION IA(300),BUF(3600)
      CALL EPINIT
      CALL EPSETW(10,1,7200,IERR)
C --- LOOP UNTIL EOF, OR ANY OTHER ERROR
    1 CONTINUE
      CALL EPREAD(10,3,NW,IA,BUF,IERR)
      IF(IERR.NE.0)  GOTO 20
C --- PROCESS CONTENTS OF A
C
      CALL USER(IA,NW)
C
C --- LOOP
      GOTO 1
   20 STOP
      END
      SUBROUTINE USER(IA,NW)
      DIMENSION IA(NW)
      PRINT*,NW,(IA(I),I=1,5)
      RETURN
      END
```

2.4 Read a File in Default Format.

Read the file of section 2.1. Note the the conversion routine `CFRIBM`.

```
      PROGRAM EPEXA4
      DIMENSION A(300),BUF(480)
      CALL EPINIT
 C--- LOOP UNTIL EOF, OR ANY OTHER ERROR
     1 CONTINUE
      CALL EPREAD(10,3,NW,A,BUF,IERR)
      IF(IERR.NE.0)  GOTO 20
      CALL CFRIBM(A,NW,3)
 C--- PROCESS CONTENTS OF A
 C
      CALL USER(A,NW)
 C
 C--- LOOP
      GOTO 1
    20 STOP
      END
```

2.5 Write a file with long blocks on a 32-bit computer

Do the same as in the example in section 2.1, but this time with a bigger buffer and convert data to IBM format for portability (`CTOIBM`).

```
      PROGRAM EPEXA5
      DIMENSION IBUF(4500),A(3000)
      CALL EPINIT
C--- INCREASE OUTPUT BUFFER = PHYSICAL BLOCK LENGTH TO 9000 16-BIT
C     WORDS (18000 BYTES), CORRESPONDING TO 4500 32-bit words
      CALL EPSETW(20,1,9000,IERR)
      IF(IERR.NE.0)  STOP 1
      DO 10  I=1,20
      CALL USER(A,NW)
C
C    ROUTINE -USER- HAS STORED NW VALUES IN A TO BE WRITTEN OUT.
C    .....
C--- NOW CONVERT TO IBM FLOATING POINT FORMAT + WRITE
      CALL CTOIBM(A,NW,3)
      CALL EPOUTS(20,3,NW,A,IBUF,IERR)
      IF(IERR.NE.0)STOP 2
   10 CONTINUE
C ***********************************************************
C
C                THE FOLLOWING CALL IS ESSENTIAL
C
C ***********************************************************
      CALL EPCLOS(20,IBUF,IERR)
      IF(IERR.NE.0)STOP 3
      STOP
      END
```

2.6 Reading back the file from the previous example

Read the file written in the previous example (EPEXA5) on a 32-bit computer, using CFRIBM to convert the real numbers to the local representation.

```
      PROGRAM EPEXA6
      DIMENSION A(3000),BUF(4500)
      CALL EPINIT
C--- INCREASE INPUT BUFFER = PHYSICAL BLOCK LENGTH TO 9000 16-BIT
C    WORDS (18000 BYTES), CORRESPONDING TO 4500 32-bit words
      CALL EPSETW(10,1,9000,IERR)
      IF(IERR.NE.0)  STOP 1
C--- LOOP UNTIL EOF, OR ANY OTHER ERROR
    1 CONTINUE
      CALL EPREAD(10,3,NW,A,BUF,IERR)
      IF(IERR.NE.0)  GOTO 20
C--- THE FOLLOWING CALL HAS NO EFFECT ON IBM (MAY BE LEFT
C    IN FOR CODE COMPATIBILITY)
      CALL CFRIBM(A,NW,3)
C--- PROCESS CONTENTS OF A
C
      CALL USER(A,NW)
C
C--- LOOP
      GOTO 1
   20 STOP
      END
```

2.7 16-bit Input and 32-bit Output

Read a tape on unit 10, with a maximum event size of 1000 words, unpacked as 16-bit words. After some reduction results are stored in array IRECO with a limit of 300 words. These results are then output to unit 20 packed 32-bits/word. The block size for both units is the EP package default (1800 16-bit words). Note the mandatory call to EPCLOS for the output unit at the end.

```
      PROGRAM EPEXA7
      DIMENSION IBUF(900),IBUFO(900),IRECI(1000),IRECO(300)
      CALL EPINIT
C--    SET INPUT STATUS WORDS  SETTING WORD 2 PREVENTS PROGRAM
C--    OVERWRITING SHOULD YOU HAVE BAD DATA
      CALL EPSETW(10,2,1000,IERR)
      IF(IERR.NE.0)STOP 1
C         THE FOLLOWING CALL IS OPTIONAL; USEFUL TO VERIFY OPTIONS
      CALL EPSTAT
C--    READ A RECORD, UNPACK AS 16-BIT WORDS
 50   CALL EPREAD(10,2,NW,IRECI,IBUF,IERR)
      IF(IERR.NE.0)GOTO 500
      PRINT*,NW
C--    DATA REDUCTION PART  NWO WORDS OF DATA ARE STORED IN IRECO
C--    NWO CAN VARY BUT THE MAXIMUM IS 300, DEFINED BY EPSETW CALL
C--     NWO WORDS
C--
      CALL USER(NWO,IRECO)
C--     THE OUTPUT IS PACKED 32-BITS PER WORD
      CALL EPOUTS(20,3,NWO,IRECO,IBUFO,IERR)
      IF(IERR.NE.0)GOTO 600
      GOTO 50
C--    READ ERRORS
C--      PARITY AND OTHER
 500  IF(IERR.NE.1)GOTO 50
C--      EOF
      CALL EPCLOS(20,IBUFO,IERR)
      CALL EPEND(20,IBUFO,IERR)
      STOP
C--    WRITE ERRORS
 600  STOP 600
      END
```

2.8 Read and Write ASCII Hollerith Strings

The package provides utility routines to convert to and from ASCII 8-bit character code, which is the way we recommend to write text.

Since the package uses a 16-bit word as basic unit, the user should read and write Hollerith strings as bit strings (`MODE=1`), and pad with a blank to an even number of characters if necessary.

Of the computers for which this package exists, only IBM and CDC do not use the ASCII code internally. IBM uses 8-bit EBCDIC, CDC uses 6-bit DISPLAY code. However, when reading and writing on VAXes, the bytes in successive pairs of bytes have to be swapped in order to come out correctly, since they get swapped again when reading or writing, to make 16-bit words look correct.

Consequently, there exist two routines, `SFRASC` and `STOASC` on IBM, CDC, and VAX which perform the necessary conversion or swapping. The following examples show how they are used.

The previous routines `CFRASC` and `CTOASC`, working on blown-up strings will continue to exist.

The following example shows how to write Hollerith strings using both the old and new routines.

```
      PROGRAM EPEXA9
      DIMENSION IBIG(1000),ISMALL(1000),IBUF(1000),STRING(22)
      DATA STRING/
      1 'ABCD','EFGH','IJKL','MNOP','QRST','UVWX','YZab','cdef',
      2 'ghij','klmn','opqr','stuv','wxyz','0123','4567','89+-',
      3 '*/()','$=,.','#[]:','"_!&',4H'?<>,'@---'/
C
C    NW IS THE NUMBER OF CHARACTERS
C
      NW= 88
      CALL EPINIT
C
C    USE PREVIOUS ROUTINES ( BLOW, CONVERT, BUNCH )
C
      CALL BLO8W(STRING,1,IBIG,1,NW)
      CALL CTOASC(IBIG,NW)
      CALL BUN8W(IBIG,1,ISMALL,1,NW)
C
C    WRITE AS-BIT STRING (NOTE NUMBER OF 16-BIT WORDS ! )
C
      CALL EPOUTS(11,1,(NW-1)/2+1,ISMALL,IBUF,IERR)
C
C--- String Conversion Routines  add same string behind first, write again
C
      CALL STOASC(STRING,1,ISMALL,NW+1,NW)
      CALL EPOUTS(11,1,NW,ISMALL,IBUF,IERR)
      CALL EPCLOS(11,IBIG,IERR)
      CALL EPEND(11,IBIG,IERR)
      STOP
      END
```

2.9 Read and convert text strings

This example shows the use of SFRASC to read and convert the character strings from the file created in the example in section 2.8.

```
      PROGRAM TEST
      DIMENSION IBUF(1000),IASC(1000)
      CALL EPINIT
   10 CONTINUE
      CALL EPREAD(11,1,NW,IASC,IBUF,IERR)
      IF(IERR.NE.0)  STOP
      CALL SFRASC(IASC,1,IASC,1,2*NW)
      NW=NW/2
      PRINT 2001,NW,IERR
 2001 FORMAT(/" NO. OF WORDS =",I5,"  ERROR =",I3)
      PRINT 2002,(IASC(I),I=1,NP)
 2002 FORMAT(/1X,25A4)
      GOTO 10
      END
```

2.10 Read old Format Tapes

This example shows typical values to use for OLD format HP or NORD tapes read on the CDC. Note `IBUF(512)` and the corresponding number of 16-bit words is 1920, and the call to `EPREAD` with `MODE=2`. Long events up to 5000 16-bit words are expected. Most data acquisition systems were converted to the new format in the early eighties.

```
      PROGRAM EPTEST(TAPE10,OUTPUT)
      DIMENSION IBUF(512),IRECI(5000)
      CALL EPINIT
C        SET STATUS WORDS FOR INPUT
      CALL EPSETW(10,1,1920,IERR)
      IF(IERR.NE.0)STOP 1
      CALL EPSETW(10,2,5000,IERR)
      IF(IERR.NE.0)STOP 2
C     SUPPRESS BYTE SWAPPING
      CALL EPSETW(10,27,0,IERR)
      IF(IERR.NE.0)STOP 3
C         THE FOLLOWING CALL IS OPTIONAL; USEFUL TO VERIFY OPTIONS
      CALL EPSTAT
C        READ A RECORD, UNPACK AS 16-BIT WORDS
 50   CALL EPREAD(10,2,NW,IRECI,IBUF,IERR)
      IF(IERR.NE.0)GOTO 500
      JJ=NW-5
      PRINT*,NW,(IRECI(I),I=1,5),(IRECI(I),I=JJ,NW)
      GOTO 50
C        READ ERRORS
C          EOF
 500  IF(IERR.EQ.1)STOP
C          PARITY AND OTHER
      GOTO 50
      END
```

2.11 Data Selection and User Defined Header.

This example starts by skipping to a selected block on the input unit `10` using `MODE=30` in `EPREAD`. Then a further selection is made on the logical record header (looking for `ID=0`) using `MODE=20`. The data for these events are then unpacked as 16-bit words, and only the first 100 words are written to the output unit (`20`) and packed in 32-bits. In addition a user defined header (`IH`) is used where `IH(1-4)` are the default package header, `IH(5)` is the original number of data words and `IH(6)` the original record sequence number.

```
      PROGRAM EPTEST
      DIMENSION IBUF(900),IBUFO(900),IRECI(1000),IH(6)
      CALL EPINIT
 C       SET STATUS WORDS FOR INPUT
      CALL EPSETW(10,2,1000,IERR)
      IF(IERR.NE.0)STOP 1
 C--    SKIP TO BLOCK 15
    5 CALL EPREAD(10,30,NW,IRECI,IBUF,IERR)
      IF(IERR.NE.0)STOP 5
      IF(IRECI(3).LT.15)GOTO 5
 C-- READ THE RECORD HEADER AND SELECT RECORDS WITH TYPE ID=0
   50 CALL EPREAD(10,20,NW,IRECI,IBUF,IERR)
      IF(IERR.NE.0)GOTO 500
      IF(IRECI(2).NE.0)GOTO 50
      IH(6)=IRECI(4)
 C-- READ DATA OF CURRENT RECORD, UNPACK AS 16-BIT WORDS
      CALL EPREAD(10,12,NW,IRECI,IBUF,IERR)
      IF(IERR.NE.0)GOTO 500
      JJ=NW-4
      PRINT*,NW,(IRECI(I),I=1,5),(IRECI(I),I=JJ,NW)
      IH(5)=NW
 C-- WRITE AS 32-BIT WORDS
      CALL EPOUTL(20,3,6,IH,100,IRECI,IBUFO,IERR)
      IF(IERR.NE.0)GOTO 600
      GOTO 50
 C       READ ERRORS
 C         PARITY AND OTHER
  500 IF(IERR.NE.1)GOTO 50
 C         EOF
      CALL EPCLOS(20,IBUFO,IERR)
      STOP
 C       WRITE ERRORS
  600 STOP 600
      END
```

2.12 Writing 32 bit physical Headers, logical Headers and Data.

This example also shows the control language to run in VM/CMS.

```
/* Test 32 bit headers */
'exec cernlib'
'vfort out32 (noprint '
'filedef iofile20 disk out32 data a4 ( recfm u lrecl 32400 block 32400'
'load out32 ( start '
'erase load map '
'erase out32 text a1'
/*BEGIN OUT32 FORTRAN A3 */
       DIMENSION IBUF(8100),A(50000)
       CALL EPINIT
       CALL EPSETW(20, 1,16200,IERR)
       CALL EPSETW(20, 2,100000,IERR)
       CALL EPSETW(20, 3,  32 ,IERR)
       CALL EPSETW(20,29,   1 ,IERR)
C   OUTPUT 4 long LOGICAL RECORDS
C
       DO 10  I=1,4
        NW=10000 * (I+1)
        CALL USER(A,NW,I)
C  ..................................................
C     ROUTINE -USER- HAS STORED NW VALUES IN A TO BE WRITTEN OUT.
C   .........................................
        CALL EPOUTS(20,3,NW,A,IBUF,IERR)
C       IF(IERR.NE.0)STOP 1
   10  CONTINUE
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
C               THE FOLLOWING CALL IS ESSENTIAL
C
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
       CALL EPCLOS(20,IBUF,IERR)
C  ..........................................................
C   FOLLOWING CALL ONLY NECESSARY ON SOME COMPUTERS (UNIVAC,IBM)
C    ON VM (AND MVS?) WE THINK THIS SHOULD BE EPRWND(20,IBUF,IERR)
C    WITH APPROPRIATE FILEDEF (IODD IN MVS)
C  ..........................................................
C      CALL EPEND(20,IBUF,IERR)
       CALL EPRWND(20,IBUF,IERR)
       IF(IERR.NE.0)STOP 2
       STOP
       END
       SUBROUTINE USER(A,NW,I)
       INTEGER A(1)
       DO 10 J=1,NW
       A(J)=I
   10  CONTINUE
       RETURN
       END
```

Chapter 3: User Routines

3.1 Initialisation

CALL **EPINIT**

Must be called once per job, in the root. Pulls common block /EPCOMM/ in and initializes the package. More than 10 units can be handled simultaneously as follows:

1. Load a common block /EPCOMM/ that is big enough (see $dim as dimension of LIST, table 5.1)
2. After the call to EPINIT set NMUNIT to the new maximum.

3.2 Reading

CALL **EPREAD** (LUNIT,MODE,NW*,*IREC*,IBUF*,IERR*)

Input parameters:

LUNIT	logical unit number
MODE	=j, j=1,2,3 : get next logical record data j=1 : data transferred in packed form j=2 : data unpacked as 16-bit bytes/word j=3 : data unpacked as 32-bit bytes/word =10+j, j=1,2,3 as above: get data of current logical record (only possible after a previous call with j=20). =20 : get header of next logical record. The header will always be unpacked in units (16 or 32-bit words). =30 : get next physical header. The header will always be unpacked in units.

Input/Output buffer:

IBUF	user provided unit buffer. Must not be touched by user. Minimum size = (16*status word 1 - 1)/(bits/word) + 1

Output parameters:

NW	no. of words transferred into IREC. For j=1 no. of units, for j=2 or j=3 no. of 16 or 32-bit words, i.e., number of machine words occupied in IREC
IREC	user provided area to store the header or data requested
IERR	error flag. See separate list.

Remarks:

1. For reading the old EP format status word 27 must be set to zero by a

```
CALL EPSETW(LUN,27,0,IERR)
```

and the previous call in the old “EVENT” package

```
CALL EVENT(IARRAY,ISTAT)
```

has to be replaced by a call to EPREAD with j=2. In this case, as for EVENT, the complete record (including the header) will be transferred. Please note that the data now start in IREC(1), instead of IARRAY(2) as before and the data are unpacked as 16-bit bytes/word only.

Old format "special records" are not specifically decoded by EPREAD but return IERR=7. The user is left to do the unpacking himself, e.g.:

```
 CALL EPREAD(LUN,2,NW,irec,IBUF, ierr)
 :
 :
 IF (IERR.NE.7) GO TO 10
 NW=IWD16(IBUF,1)
 CALL BLO16W(IBUF,1,IREC,1,NW)
10 CONTINUE
```

2. To read consecutive files of multifile tapes the user has only to make further calls to EPREAD after each end of file (signified by IERR=1). In the case of labelled tapes the labels will be treated as data and EPREAD will return a length error on reading (IERR=5). The user program should choose to continue reading in this case. A combination of these techniques with MODE=30 can provide the same facilities as EVENTIN in the old package. To prevent tapes running off the end of reel users should stop reading at end of information (IERR=3), signified by 2 consecutive EOF's, or by using the file or record count.

CALL **EPFHDR** (LUNIT,MLUSER,IHEAD*,*IBUF*,IERR*)

Fast logical record header reading routine.

Input parameters:

LUNIT user unit number

MLUSER no. of header words transferred to user

Input/Output buffer:

IBUF user buffer

Output parameters:

IHEAD MLUSER words of logical record header (regardless of actual length or of status word 26)

IERR error number

Conditions for use:

1. 16-bit units only (i.e. logical record header consists of 16-bit words)
2. No spanned headers
3. Always MLUSER words transferred to user (word 3 is header length). This may exceptionally lead to a program range error if the input buffer IBUF coincides with the end of the user program. Remedy: increase size of IBUF by 16*MLUSER/(no. of bits per word)
4. No headerless blocks
5. No old EP format
6. No unknown length records

In addition, no checks performed whether input unit, or whether header cut. Conditions 2. to 6. are always fulfilled when writing with EPIO, condition 1. is default when writing with EPIO.

Calls to this routine are entirely compatible with `EPREAD` calls.

`CALL` **EPFRD** `(LUNIT,MODE,NW,IREC,IBUF,IERR)`

Fast logical record data reading routine.

Input parameters:

`UNIT`	user unit number
`MODE`	one of 11, 12, 13 (see `EPREAD`)

/Input/Output buffer:

`IBUF`	user buffer

Output parameters:

`NW`	no. of words in `IREC`
`IREC`	record transferred to user
`IERR`	error number

Conditions for use:

1. 16-bit units only (i.e. logical record header consists of 16-bit words)
2. no headerless blocks
3. no old EP format
4. no unknown length records
5. modes 11, 12, 13 only (otherwise error 8)

In addition, no checks performed whether input unit. Conditions 2. to 4. are always fulfilled when writing with EPIO, condition 1. is default when writing with EPIO. User data will be truncated at the value in status word 2, but no error will be signalled.

The routine is poorly protected against bad user data on input, and may lead to program aborts in cases where `EPREAD` would recover. In particular `EPFRD` does not test for the condition which leads to `IERR=18` from `EPREAD`, and performs unpredictably in the case of this condition.

Therefore, in case the user wants to be better protected (e.g., after a fatal error in `EPFRD`), he should use `EPREAD`.

Calls to this routine are entirely compatible with `EPREAD` calls.

3.3 Writing

`CALL` **EPOUTS** `(LUNIT,MODE,NW,IREC,*IBUF*,IERR*)`

Writes one record with standard logical record header (4 words, see important remark on logical record headers in section 5.3.

Input parameters:

`LUNIT`	logical unit number

`MODE` transfer more

- `1:` transfer data as they are (bit string)
- `2:` pack data before writing, given as 16-bit right adjusted
- `3:` pack data before writing, given as 32-bit right adjusted

`NW` no. of words to be written.
For `MODE=1` in units (as defined by physical header word 11), for `MODE=2` or `=3` in 16 or 32-bit words respectively, i.e. the number of machine words occupied

`IREC` area containing the user data.

Input/Output buffer:

`IBUF` user provided unit buffer. Must not be touched by user.
Minimum size = `(16*status word 1 - 1)/(bits/word) + 1`

Output parameters:

`IERR` error flag. See separate list.

`CALL` **EPOUTL** `(LUNIT,MODE,NH,IH,NW,IREC,*IBUF*,IERR*)`

This routine writes a user specified logical record header in front of the data, instead of the header produced automatically from the status words. Words 1,3 (and possibly 2 and 4) will be set by the writing routine in any case (basic protection).

The logical record data are given in the same call.

Input parameters:

`LUNIT` logical unit number

`MODE` transfer more

- `1:` transfer data as they are (bit string)
- `2:` pack data before writing, given as 16-bit right adjusted
- `3:` pack data before writing, given as 32-bit right adjusted

`NH` header length in words

`IH` array containing header (in unpacked form)

`NW` no. of words to be written.
For `MODE=1` in units (as defined by physical header word 11), for `MODE=2` or `=3` in 16 or 32-bit words respectively, i.e. the number of machine words occupied

`IREC` area containing the user data.

Input/Output buffer:

`IBUF` user provided unit buffer. Must not be touched by user.
Minimum size = `(16*status word 1 - 1)/(bits/word) + 1`

Output parameters:

`IERR` error flag. See separate list.

`CALL` **EPCLOS** `(LUNIT,*IBUF*,IERR*)`

Routine to close output units (write existing physical buffer out - writing may continue afterwards). The physical block count will continue from its current value at re-opening.

Closing an input unit has no effect.

Mandatory at end of job for all output units (or `EPEND`)

Input parameters:
`LUNIT` logical unit number
Input/Output buffer:
`IBUF` user provided unit buffer. Must not be touched by user.

Output parameters:
`IERR` error flag. See separate list.

3.4 Auxiliary Unit Operations—Rewind

```
CALL EPRWND  (LUNIT,*IBUF*,IERR*)
```

Rewinds to the start of the current file in use on this unit.

Rewinding an output unit triggers writing the last buffer out and writes an end of file. After a rewind operation (and only in that case) the user may switch from reading to writing or vice versa.

The physical header is reset to the standard header after a rewind.

Input parameters:
`LUNIT` logical unit number
Input/Output buffer:
`IBUF` user provided unit buffer. Must not be touched by user.

Output parameters:
`IERR` error flag. See separate list.

Note:

On the Apollo rewinding a magnetic tape file is done by explicitly closing the stream.

3.5 Drop

```
CALL EPDROP  (LUNIT,IERR*)
```

Drops an existing unit. User should call `EPCLOS` beforehand when writing. This concerns only this package, i.e. the space in common block `/EPCOMM/` becomes free for another unit. For the system, the unit still exists, and may therefore be “opened” again later. The physical block count will be reset at re-opening.

Input parameters:
`LUNIT` logical unit number
Output parameters:
`IERR` error flag. See separate list.

3.6 Write EOF mark

```
CALL EPEND (LUNIT,*IBUF*,IERR*)
```

Mandatory at end of job for all output units (or EPCLOS)

Acts like EPCLOS, but in addition writes an end of file mark on LUNIT if output unit. On IBM and UNIVAC it must be called to properly terminate output units.

Note that on IBM this has the same effect as EPCLOS - the filemark written is backspaced over so a new write to the same logical unit will not result in a new physical file but will merely continue the existing file. Users who really want to write a multi-file tape on IBM within the same job step must call EPEND and then IODD [3].

Input parameters:

LUNIT logical unit number

Input/Output buffer:

IBUF user provided unit buffer. Must not be touched by user.

Output parameters:

IERR error flag. See separate list.

3.7 Utility Routines

```
CALL EPUREF (LUNIT)
```

Returns internal reference number for logical unit in LREF in the common block /EPCOMM/. If LUNIT is invalid or there are no units open, then LREF=0.

Input parameters:

LUNIT logical unit number

```
CALL EPDEFU (LUNIT,CHNAME,IERR*)
```

Allows to specify a file name other than the EPIO default name.

Input parameters:

LUNIT logical unit number

CHNAME Character variable, specifying the file name.

Output parameters:

IERR error flag. See separate list.

3.7.1 Get Status Words

```
CALL EPGETW (LUNIT,NUMBER,IW*,IERR*)
```

Input parameters:

LUNIT logical unit number

NUMBER Number of the desired status word.

Output parameters:

`IW` value of the status word.

`IERR` error flag. See separate list.

```
CALL EPGETA (LUNIT,NW,IWS*,IERR*)
```

Input parameters:

`LUNIT` logical unit number

`NW` Number of desired status words (always start at word 1).

Output parameters:

`IWS` value of status words `1..NW`.

`IERR` error flag. See separate list.

```
CALL EPGETC (NWCONT*,IWCONT*)
```

Output parameters:

`NWCONT` no. of overall control words (word 3 of common `/EPCOMM/`).

`IWCONT` control words (words `1..NWCONT` of list of common `/EPCOMM/`)

3.7.2 Set Status Words

These routines enable the user to change defaults kept in the status words, such as the maximum length of the physical block he wants to read, the maximum length of the logical record, and so on.

Normally, control words should be changed before any writing on that unit takes place. Otherwise, it is strongly recommended to make a close call to the corresponding unit (`EPCLOS`) beforehand in order to avoid problems. The user should be fully aware of what he is doing, and foresee the possible effects of any change.

For the use of `EPSETA`, and `EPSETC` (setting more than one word), it is recommended to get hold of the existing status words by calls to `EPGETA` or `EPGETC`, modify where desired, and restore.

```
CALL EPSETW (LUNIT,NUMBER,IWORD,IERR*)
```

Input parameters:

`LUNIT` logical unit number

`NUMBER` Number of the status word to change.

`IWORD` new value of the status word.

Output parameter:

`IERR` error flag. See separate list.

```
CALL EPSETA (LUNIT,NW,IWORDS,IERR*)
```

Input parameters:

`LUNIT` logical unit number

`NW` Number of status words to set (always start at word 1).

`IWORDS` new value for the status words `1..NW`.

Output parameter:

`IERR` error flag. See separate list.

`CALL` **EPSETC** `(NW,IW)`

Input parameters:

`NW` no. of overall control words to set (from 1 up to `NCONT`)

`IW` new overall control words.

Handle with care!

`CALL` **EPADDH** `(LUNIT,NH,IH,*IBUF*,IERR*)`

Adds a user portion to the standard physical header. The current block is written out beforehand if not empty.

The physical header is reset to the standard header after a rewind.

Input parameters:

`LUNIT` logical unit number

`NH` number of 16-bit words to add (0 resets the standard physical header).

`IH` user physical header words to add (16-bit, right adjusted, 0 filled)

Input/Output buffer: User provided unit buffer. Must not be touched by the user.

Output parameter:

`IERR` error flag. See separate list.

`CALL` **EPUPDH** `(LUNIT,NH,IH,IBUF,IERR)`

Updates the (optional) user portion of the physical block header. The necessary space for this optional portion must have been reserved by the user through a previous call to `EPADDH` (typically once at the start of the job). Whereas a call to `EPADDH` causes the existing (if any) block to be written out, `EPUPDH` does not have this effect.

Input parameters:

`LUNIT` logical unit number

`NH` no. of user words in header to be updated. It is cut at the actual no. of extra user words available in the physical block header.

`IH` Array containing the new values. Words `1..NH` of `IH` will replace the current words `1..NH` of the user portion in the header.

Input/Output buffer: User provided unit buffer. Must not be touched by the user.

Output parameter:

`IERR` error flag. See separate list.

3.7.3 Print Overall Status

`CALL` **EPSTAT**

Prints the overall control words, and all unit status words in their
actual state. Useful for debugging.

`CALL` **EPIOT** `(LUN)`

Print PAM title on `LUN`.

Chapter 4: Auxiliary routines used by EP read/write

These routines will have to be provided by the user when he installs the package on his computer, if no version for his computer yet exists (see list in chapter 1).

Important remark:

The source and target word positions in some of these routines are given in 16-bit word counts.

It should be stressed that, due to different representation in different computers, floating and integer type conversion have to be called separately.

4.1 Move, Blow, and Bunch

```
CALL W16MOV (SOURCE,N1,TARGET,N2,N3)
```

Moves 16-bit words in packed form.

Input parameters:

`SOURCE`	source array
`N1`	starting 16-bit word in source
`TARGET`	target array
`N2`	starting 16-bit word in target
`N3`	no. of 16-bit words to move

```
CALL BLO8W (SOURCE,N1,TARGET,N2,N3)
```

Blows (unpacks) 8-bit words into machine words.

Input parameters:

`SOURCE`	array containing 8-bit words in packed form
`N1`	first 8-bit word in source to blow
`TARGET`	target array for unpacked 8-bit words
`N2`	`TARGET(N2)` will contain first unpacked word
`N3`	no. of 8-bit words to unpack

```
CALL BUN8W (SOURCE,N1,TARGET,N2,N3)
```

Bunches (packs) 8-bit words (right adjusted, zero filled) into a continuous bit string.

Input parameters:

`SOURCE`	array containing 8-bit words in unpacked form
`N1`	first 8-bit word in source to packed
`TARGET`	target array for packed 8-bit words
`N2`	first 8-bit word in target to pack into
`N3`	no. of 8-bit words to pack

```
CALL BLO16W (SOURCE,N1,TARGET,N2,N3)
```

Blows (unpacks) 16-bit words into machine words.

Input parameters:

`SOURCE` array containing 16-bit words in packed form
`N1` first 16-bit word in source to blow
`TARGET` target array for unpacked 16-bit words
`N2` `TARGET(N2)` will contain first unpacked word
`N3` no. of 16-bit words to unpack

`CALL` **BUN16W** `(SOURCE,N1,TARGET,N2,N3)`

Bunches (packs) 16-bit words (right adjusted, zero filled) into a continuous-bit string.

Input parameters:

`SOURCE` array containing 16-bit words in unpacked form
`N1` first 16-bit word in source to packed
`TARGET` target array for packed 16-bit words
`N2` first 16-bit word in target to pack into
`N3` no. of 16-bit words to pack

`CALL` **BLO32W** `(SOURCE,N1,TARGET,N2,N3)`

Blows (unpacks) 32-bit words into machine words. ***Important remark:*** Counts partly in 16-bit words.

Input parameters:

`SOURCE` array containing 32-bit words in packed form
`N1` first 32-bit word in source to blow
`TARGET` target array for unpacked 32-bit words
`N2` `TARGET(N2)` will contain first unpacked word
`N3` no. of 32-bit words to unpack

`CALL` **BUN32W** `(SOURCE,N1,TARGET,N2,N3)`

Bunches (packs) 32-bit words (right adjusted, zero filled) into a continuous-bit string. ***Important remark:*** Counts partly in 16-bit words.

Input parameters:

`SOURCE` array containing 32-bit words in unpacked form
`N1` first 32-bit word in source to packed
`TARGET` target array for packed 32-bit words
`N2` first 32-bit word in target to pack into
`N3` no. of 32-bit words to pack

`IVAL =` **IWD16** `(A,N)`

Function returns the 16-bit word no. `N` of string `A`. The first 16-bit word coincides with the start of `A`.

4.2 Conversion

The following routines are optional, but useful when data are written in IBM integer or floating format.

`CALL` **CTOIBM** `(*ARRAY*,NW,MODE)`

Input/Output buffer:

`ARRAY` On input contains words to be converted to IBM format. After conversion it will contain the IBM formatted words right adjusted, zero filled after the call.

Input parameters:

`NW` number of words to be converted

`MODE` Conversion mode

- `1:` convert to 16-bit (signed) IBM integers. Conversion of unsigned integers before writing is not necessary (they must not be longer than 16-bit, of course)
- `2:` convert to 32-bit IBM integers.
- `3:` convert to 32-bit IBM floating.

```
CALL CFRIBM (ARRAY,NW,MODE)
```

The routine converts from IBM format to local machine format. The input has to be right adjusted, zero filled in `ARRAY`. The arguments are the same as for `CTOIBM` above.

```
CALL CFRASC (ARRAY,NW)
```

The routine converts from ASCII character to local character format. `ARRAY` contains `NW` ASCII characters, in the form of one character (per word, right adjusted, zero filled). The conversion is performed in place.

```
CALL CTOASC (ARRAY,NW)
```

Inverse action of `CFRASC`.

```
CALL SFRASC (SOURCE,N1,TARGET,N2,NCH)
```

IBM version

Converts a Hollerith string from ASCII to EBCDIC (IBM internal)

CDC version

`SFRASC` converts a string of 8-bit ASCII characters into a string of 6-bit display characters.

VAX version

Converts a Hollerith string from external ASCII to internal ASCII, i.e. on VAX performs byte swapping only. By making source and target, and `N1` and `N2` identical, the routine can be used for byte swapping only (identical to `STOASC`).

Parameters

Input parameters:

`SOURCE` array containing the ASCII string

`N1` first character in source to convert

`TARGET` array receiving the EBCDIC string

`N2` first character position of converted string in target

`N3` number of characters to convert

The routine allows the simultaneous conversion and concatenation of a string. On VAX, IBM (not CDC !), source and target may overlap as long as N2≤N1.

```
CALL STOASC (SOURCE,N1,TARGET,N2,NCH)
```

IBM version

Converts a Hollerith string from EBCDIC (IBM internal) to ASCII

CDC version

`STOASC` converts a string of CDC display characters into a string of 8-bit ASCII characters.

VAX version

Converts a Hollerith string from external ASCII to internal ASCII, i.e. on VAX performs byte swapping only. By making source and target, and `N1` and `N2` identical, the routine can be used for byte swapping only.

Parameters

Input parameters:

`SOURCE`	array containing the EBCDIC string
`N1`	first character in source to convert
`TARGET`	array receiving the ASCII string
`N2`	first character position of converted string in target
`N3`	number of characters to convert

The routine allows the simultaneous conversion and concatenation of a string. On VAX, IBM (not CDC !), source and target may overlap as long as N2≤N1.

Chapter 5: Technical details

5.1 Status words

One set of status words per unit will be kept in a common block (which should reside in the root for overlayed programs). The status words contain all information the routines have to know in order to operate. The shorthand notations 'physical header' and 'record header' refer to the physical and logical headers respectively of the EP format description.

Words marked '*' can be defined and changed by the user through calls to special routines (the user should not write into the status word area directly).

r, w, and r/w under 'use' means reading or writing only, or both.

Table 5.1: Overview of the status words

1	*	r/w	Physical block length in 16-bit words. Used for writing; should always be a multiple of the "magic" 180. For the UNIX and "standard Fortran" versions it is the actual block length. For reading, it is the upper limit of the EPIO buffer `IBUF`. The user has to provide sufficient space in `IBUF`. Example: for a buffer length of 1800 16-bit words, `IBUF(480)` is needed on a 60-bit machine. Default=1800
2	*	r	Maximum logical record length (in machine words). Default=999999
3	*	r/w	Reading : as read from physical header word 11 Writing : Logical record word length (16 or 32-bits); a word of this length will be called a "unit" in the following. Default=16
4	*	r/w	P.h. word 5 (run number), read or written Default=10101
5	*	w	If ≥ 0, logical record type identifier (will be used as record header word 2). If ≤ 0, for user purposes. Default=0
6	*	w	If ≥ 0, logical record count will be placed in record header word 4 If < 0, record header word 4 will not be set (i.e. for user purposes). Default=0
7	*	r/w	Reading : as read from physical header word 2, or 0 if no physical header Writing : Actual physical header length. Is automatically set at call to `EPADDH`. Default=12 (see control word 6), 24 for 32 bit p.h.
8	*	w	Output padding flag, values `i+j` `i=0` : span logical records over blocks `i=10`: do not span logical records Remark: headers will never be spanned on writing

			`j=1` : pad physical block with zeros to full length (i.e. write fixed length blocks)
			`j=2` : pad up to next magic multiple (see control word 5)
			`j=3` : do not pad at all
			Default=1
9	*	r/w	Physical header word 6 (physical record type)
			Default=0 (16 bit p.h.), 1 for 32 bit p.h.
10		r/w	Logical unit number
11		r/w	Number of blocks read or written
12		r/w	Number of records read or written
13		r/w	Number of read or write parities
14		r/w	Actual occupation of I/O buffer (16-bit words)
15		r/w	Displacement to start of first logical record (physical header)
16		r/w	Status indicator: 0 at start, 1 writing, 2 reading.
17		r	Flag for old (=1) or new (=0) EP format. Recognized when reading.
18		r	Number of headerless blocks still following (from physical header word 9)
19		r	Normally 0, 1 if last read access resulted in E.O.F.
20		r	Actual number of units read, including header, in current logical record (logical record header word 1)
21		r	Logical record header length (logical record header word 3)
22		r	Position indicator - Internal in `EPREAD`
			`=0` : pointer at start of logical record (header)
			`=1` : pointer at start of logical record (data)
			`=2` : pointer at end of physical header
23		r	Internal pointer `IP1` in `EPREAD`
24		r/w	Internal unit name UNIVAC and Apollo only
25	*	r/w	VAX: Channel address (magnetic tape)
			UNIVAC: -1 if tape, otherwise sector address
			IBM: if 0 (default) `ULP` option, else `NULP` (see `IOPACK`)
			`NULP` allows concatenation of files, `ULP` does not
			Apollo: must be set before first read `Stream _$ID`
			UNIX C I/O: stream address (not user settable!)
26	*	r	Maximum logical record header length, for calls to `EPREAD` with `MODE=20` (default 999999)
27	*	r	User wants automatic byte-swapping if 1 (default) or suppression of it (=0).
			User forces new format if 2
28		r	Internal usage
29	*	r/w	Select physical header record format
			writing: 16 bit headers = 0 (default) or 32 bit headers = 1
			reading: as deduced from header words 7 and 8
30	*	w	Vax tape append option,

			= 0 (default) new tape file,= 1 append.
31		r	=Pointer to logical reader header, for random access
32		r	=Internal usage
33	*	r/w	Select type of I/O in the UNIX version; see section 1.2 “Usage on UNIX”.

The status words are kept in a common block

```
COMMON/EPCOMM/NMUNIT,NWUNIT,NCONT,ISTART,LASTUT,LREF,LIST($dim)
```

with

`NMUNIT`	max. no. of units supported simultaneously, reading plus writing (default = 10).
`NWUNIT`	number of status words per unit according to list above (default = 32).
`NCONT`	number of overall control words for package according to list below (default = 8).
`ISTART`	internal reference flag
`LASTUT`	internal reference flag
`LREF`	internal reference flag
`LIST`	status word list, with `$dim = NCONT+NWUNIT*NMUNIT`, this allows the support of more units than the default 10 (see below) default value of `$dim = 350`.

Important note

These defaults may change in a later version of the package. Users who want to use or modify these values (e.g.: to support more than 10 simultaneous units) are urged to check the current values by printing the start of above COMMON block after a call to `EPINIT`.

5.2 Control words

These control words are kept in the first 8 locations of the array `LIST` in common block `/EPCOMM/`.

1 no. of units actually in use
2 max. no. of error print messages (default=100)
3 actual number of error print messages
4 no. bits/machine word
5 no. of 16-bit words / recommended unit (default=180)
6 standard physical header length (default=12) in words (16 or 32 bit)
7 number of unit control words accessible to user mods
8 logical print output unit

5.3 Important remark on logical record headers

The logical record header consists of 3 words minimum, but the standard form has four words, being:

1. the logical record length
2. the logical record type identifier
3. the logical record header length
4. the logical record sequence number.

Of these, words 1 and 3 will always be provided by the output routines when writing (basic protection).
Words 2 and 4 of the logical record will always be assigned the values described above when the user writes via calls to `EPOUTS`.
For use with `EPOUTL`, two status words, number 5 and 6, are used to assign values in the following way:

let the status words be `S5`, `S6`, `S12` and the header words be `H2`, `H4`,

then

`if S5` ≥ 0 `H2` is set to `S5` by `EPOUTL`
`if S5` < 0 `H2` is not set by `EPOUTL`, i.e. the user should have set it before calling `EPOUTL`.
`if S6` ≥ 0 `H4` is set to `S12` (!) by `EPOUTL`
`if S6` < 0 `H4` is not set by `EPOUTL`, i.e. the user should have set it before calling `EPOUTL`.

5.4 Remark on padding

Full padding, partial padding, and no padding have certain consequences on the different computers, and should therefore be considered beforehand.

1. Full padding: all blocks on the output file have the same (user specified) length, which should be a multiple of 180 16-bit words. This mode will probably work on any computer for reading and writing.
2. Partial padding: ensures the blocks to have an integer length when counted in machine words (no trailing bits, or incomplete words on reading, no extra-bits when writing). This should normally work except on computers which can only read and write fixed length blocks.
3. No padding: since in the current package, full machine words are written on all computers, this may lead to problems when reading, on another machine but normally only if the physical or logical lengths are not known. For example, due to extra padding during the transfer, records of unknown length may not be processed correctly when received through a network.

5.5 List of errors

Table 5.2 explains the meaning of the integer `IERR` returned as the last parameter of most EPIO subroutine calls, zero meaning no error. For each error two types of routines are quoted: those called by the user, and in which the error condition was detected, and those in which the error occurred.

Control is always returned to the user, but some of the errors (marked by * in first column) are so serious that it becomes meaningless to continue reading or writing on the unit concerned, at least after a limited number of them.

Table 5.2: Overview of the error codes

Error	**Routine**	**User routine**	**Meaning**
1	EPBLIN	EPREAD	End of file on reading or open failure on IBM
	EPBOUT	EPOUTL	reading or writing
		EPOUTS	
2	EPBLIN	EPREAD	r/w parity, or I/O error (IBM)
	EPBOUT	EPOUTL	
		EPOUTS	
		EPRWND	
3	EPBLIN	EPREAD	end of information on reading, or in some cases after an open error on IBM
4*	EPBLIN	EPREAD	physical record length ≤ 0
5*	EPBLIN	EPREAD	physical record length of record just read > actual length of block read or user buffer too small
6	see ⇒	EPREAD	user record chopped (`IREC` too small, status word 2) the actual length (including header) can be retrieved from status word 20 using `EPGETW`
7*	see ⇒	EPREAD	physical header error; could be a record in the old format
8	see ⇒	EPOUTS	invalid mode specified in call
		EPOUTL	
		EPREAD	
9	see ⇒	EPREAD	call to `EPREAD` with mode 11, 12, or 13, without prior call with `MODE=20`
			You can also get this error reading OLD format, with `MODE=20` followed by 11, 12 or 13
10	EPBLIN	EPREAD	end-of-run (logical record length = 0)
11*	EPBOUT	EPOUTS	unit not declared on JCL card,
		EPOUTL	or wrong `BLKSIZE` (only IBM)
	EPBLIN	EPREAD	
12	EPBOUT	EPOUTS	end of volume, or unrecovered write
		EPOUTL	parity (only IBM [3])
		EPCLOS	

Table 5.2: Overview of the error codes

Error	Routine	User routine	Meaning
		EPRWND	
13	EPUNIT	EPOUTS	maximum number of units reached
		EPOUTL	
		EPRWND	
		EPADDH	
		EPSETW	
		EPSETA	
		EPREAD	
		EPGETA	
		EPGETW	
14	see ⇒	EPDROP	unit does not exist
		EPADDH	
		EPEND	
15	see ⇒	EPOUTS	logical record header (or complete record)
		EPOUTL	too long to fit in physical block
16	see ⇒	EPGETW	status word address out of range
		EPGETA	
		EPSETW	
		EPSETA	
17	see ⇒	EPADDH	user switches from reading to writing
		EPREAD	or vice versa, without rewinding unit
		EPOUTL	
		EPOUTS	
18	see ⇒	EPREAD	displacement to start of logical record inconsistent with current logical record length, not tested by EPFRD
19	EPOUTL	EPOUTL	number of words to write < 0
		EPOUTS	
20	EPOUTL	EPOUTL	Negative record length or negative or
		EPOUTS	zero header length in user call
21	EPREAD	EPREAD	Old format and MODE≠2
22	EPBLIN	EPREAD	Wrong 32 bit physical header read
23	EPREAD	EPREAD	Logical record unit neither 16 nor 32 bit (status word 3).
24	EPOUTL	EPOUTL	Logical record or header length
		EPOUTS	> 65536 for 16 bit headers
25	EPBLIN	EPREAD	Error from `cfseek` reading in random access mode

5.6 Format Description

5.6.1 Tapes

Logical records may be of fixed or variable length. A logical record may be entirely contained in one physical record or may overflow into one or more physical records. Normally an event or trigger corresponds to one logical record but it may equally well consist of a sequence of related logical records.

Physical records may be of fixed or variable length.

A fixed record length is recommended (for ease of reading using standard system and particularly FORTRAN, input routines). A fixed record length is particularly important if the record has no header specifying the length.

A record length which is a multiple of 360 bytes is recommended (corresponding to an integral number of 16, 18, 24, 32, 36, 48, 60, 64-bits).

The following table indicates the increase in tape utilisation with record length up to the maximum of 32K bytes for IBM 370 [4].

Usage(%)	67	80	88	90	94
Record length (bytes)	3600	3600*2	3600*4	3600*5	3600*8

Table 5.3: Tape usage at 6250 bpi

5.6.2 Files

It is recommended that logical records relating to the same experimental conditions or 'run' are grouped in one file.

For unlabelled tapes the end-of-file (and end-of-run) is indicated by one EOF mark.

For labelled tapes the end-of-file (and end-of-run) is indicated by an EOF mark and the associated EOF records [5].

End-of-data on the tape (and end-of-file) is indicated by two consecutive EOF marks. It is NOT recommended that files (runs) span from one tape to another.

END-OF-TAPE. It is NOT recommended to write past the EOT reflective marker, therefore, on detection of the EOT marker, the software should backspace over a sufficient number of data records to allow the required end-of-run and EOF records to be written.

RECORD FORMATS. Logical records and physical records, (i.e. physical tape blocks) normally consist of a number of header words, followed by data. In some cases the number of header words in a physical record may be 0.

WORD LENGTHS. All lengths in the physical header are expressed in words where a word is defined as a 16-bit unsigned integer. The logical record wordlength is defined in the physical record header.

`byte 0` is defined as the highest order 8-bits of the word,

`byte 1` is defined as the next highest order 8-bits, etc.

`byte 0` is written to tape before byte 1.

Logical Record Format

A logical record normally has a minimum of three header words followed by data. The logical record wordlength is defined in the physical record header. Logical record headers with 32 bit words should have their 16 bit components in “IBM” order.

Table 5.4: Logical Record Format

Word	Definition
	Header
`1`	The logical record length (`LRL`), which should always correspond to the actual number of words in the logical record, except in the following special cases: `LRL=1` The total length of the logical record is unknown at the time of writing (the logical record header to tape). `LRL=0` End-of-data in this run. `LRL=`$2^{16}-1$ End-of-data in this physical record (16-bit word all-bits set). Note that even when the logical record wordlength is 16-bits the 16-bits following the last word of the current logical record will,when set, be an unambiguous indication of the end-of-data in the physical record.
`2`	Logical Record Type Identifier It is mandatory,except in the above mentioned special cases (but may simply be set to zero). No attempt is made to standardise on record types as they are rather application dependent.
`3`	Logical Record Header Length (`LHL`), whose definition is strongly recommended. The offline package writes this word and uses it when unpacking.
`4`	Logical record sequence number. Its use is recommended. (It has been found useful online but is not required by the offline unpack routines.)
`...`	
	Data
`LHL+1`	First Data Word
`...`	
`LRL`	Last Data Word

Physical Record Format

Each physical record normally has a variable length physical record header, followed by data. However, certain records, following a record with a suitable header may have no header. The word length may be 16 or 32 bits but in both cases the values of lengths and pointers are in units of 16 bits.

The physical header identifier fields (7 and 8) are used by the off-line package to identify the header word length, to help in error detection and recovery, and to decide if byte swapping is required. For the 32 bit header words, the least significant 16 bits are in the leftmost 16 bits of the word, i.e. "IBM format". The off-line package achieves this by calling `CTOIBM` before the header is output.

Table 5.5: Physical Record Format

Word	Definition
	Header
1	Physical Record Length (`PRL`)
2	Physical Record Header Length (`PHL>SHL`) (see below)
3	Physical Record Number (`NR`) This number identifies each record on tape. Each physical record counts in the numbering even if there is no physical record header. `NR` is modulo 65536 (2^{16}) for 16 bit headers.
4	Displacement (`NS` from first header word to first word of first logical record in present physical record (or to end-of-data indicator if no logical record starts in current physical record). `LS=PHL` if logical record starts immediately after header `LS=0` if ALL data in this record belong to current logical record
5	Run Number or other Run Identifier. They can be selected by Run number or other Run identifier.
6	Physical Record Type. It is application dependent and is foreseen to readily select data from different sources which may be interleaved on tape.
7	Physical Header Identifier Fields 1 =29954 (16 bit), 522144444 (32 bit)
8	Physical Header Identifier Fields 2 =31280 (16 bit), 522144444 (32 bit) These Header Identifier fields are present to identify extensions to the 'EP' format [2] and to recognise the physical header during error recovery.
9	Number of Physical Records following this one, with 0 length headers. 0 if next record has a header. Physical records without headers are **not** recommended, mainly to help error recovery.
10	Tape Format Version Number = 8012
11	Logical Record word length (in-bits, 16 or 32) This is the word length (of header and data words) for all logic records which start in this physical record. Initially the unpack routines will be written to handle 16 or 32-bit words.
12	Length of Standard Physical Record Header (`SHL=12` for 16 bit or 24 for 32 bit) This length (`SHL`) is to be used as a displacement to access the user header words (independently of any future addition to the standard header).
	Data
`SHL+1`	Start of User Header Words
...	
`PHL+1`	Start of data from previous logical record (if any)
...	
`LS+1`	First word of first logical record in this physical record (if any)
...	
...	Logical record(s) data
...	
`PRL`	Last word in physical record

Bibliography

[1] L. Lamport. LaTeX *A Document Preparation System*. Addison-Wesley, 1986.

[2] J. Ogilvie, *Standard Format for data recorded on Magnetic Tape* CERN-DD Division, DD/OC/79-1

[3] R. Matthews *CERN Program Library IOPACK* Long Write-Up Z-300, DD/US/45

[4] *IBM Users Guide*, DD/US/1

[5] E.M. Rimmer, J. Ogilvie, *On-Line Support for Labelled Magnetic Tapes* CERN-DD Division, DD/OC/79-2

Index