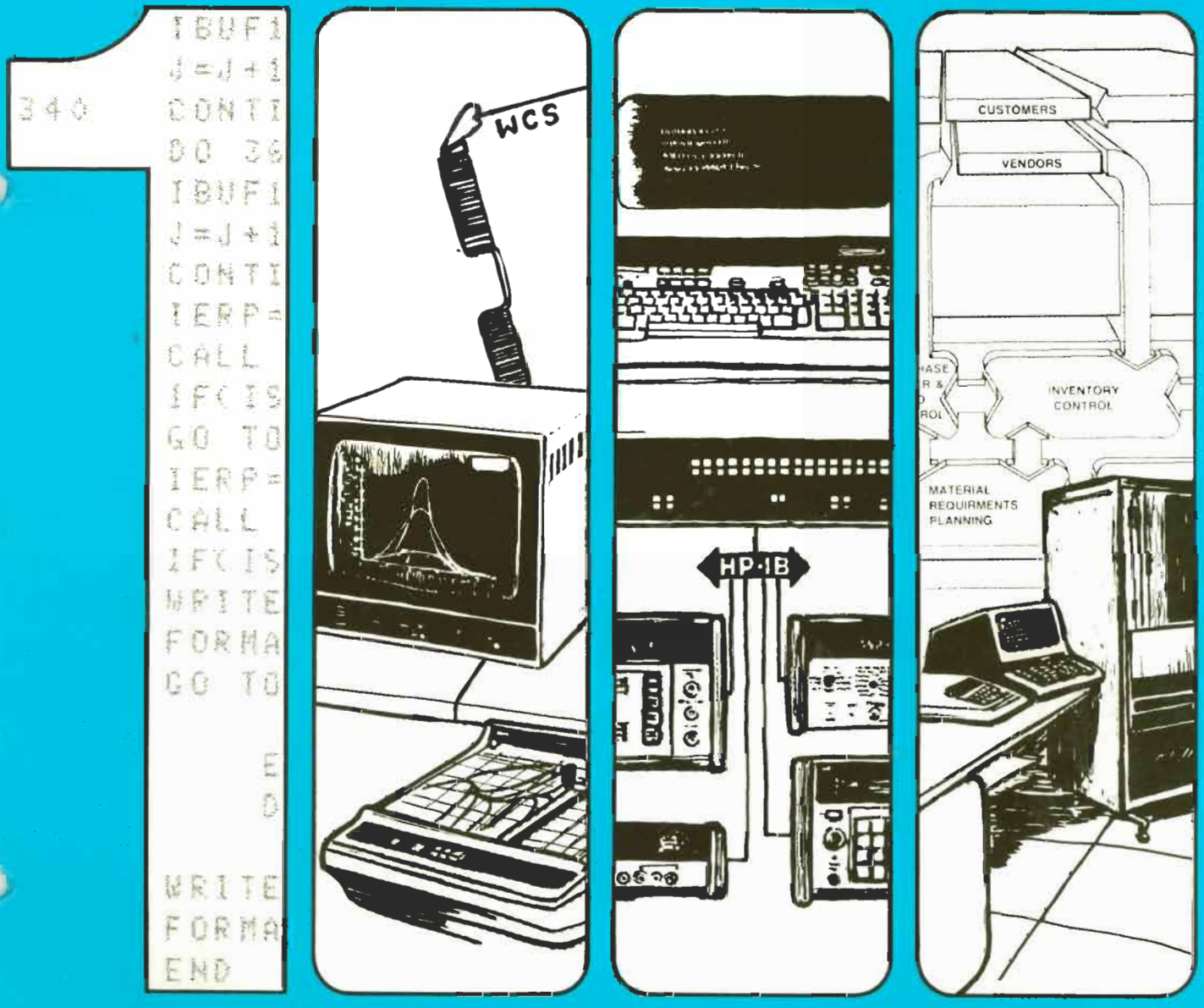
HEWLETT PACKARD

Hewlett-Packard Computer Systems

COMMUNICATOR



1982

Volume V Issue 5

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

HEWLETT-PACKARD
COMPUTER SYSTEMS

Volume V
Issue 5

COMMUNICATOR/1000

Departments

ABOUT THIS ISSUE

Vol. 5, Issue 5 is a departure from our normal format of centering around feature articles. Due to a lack of feature-length articles, but a building catalog of bit bucket contributions, we decided to print an "all Bit-Bucket" issue. The tips included are rather wide-ranging in subjects. I hope most of you can find something useful in at least one of these areas.

And for those of you who are not happy with this format, don't complain! Take some action and write a feature for a future issue.

Thanks for your support.

Best Regards,

Ms. Editor

EDITOR'S DESK

BECOME A PUBLISHED AUTHOR IN THE COMMUNICATOR/1000 . . .

The COMMUNICATOR is a technical publication designed for HP 1000 computer users. Through technical articles, the direct answering of customers' technical questions, cataloging of contributed user programs, and publication of new product announcements and product training schedules, the COMMUNICATOR strives to help each reader utililize their HP 1000's more effectively.

The Feature Articles are clearly the most important part of the COMMUNICATOR. Feature Articles are intended to promote a significant cross-fertilization of ideas, to provide in-depth technical descriptions of application programs that could be useful to a wide range of users, and to increase user understanding of the most sophisticated capabilities designed into HP software. You might think of the COMMUNICATOR as a publication which can extend your awareness of HP 1000's to include that of thousands of users worldwide as well as that of many HP engineers in Data Systems factories at Cupertino, California and Grenoble, France.

To accomplish these goals, editors of the COMMUNICATOR actively seek technical articles from HP 1000 customers, HP Systems Engineers in the Field, and Marketing and R&D Engineers in the factories. Technical articles from customers are most highly valued because it is customers who are closest to real-world applications.

WIN AN HP-32E CALCULATOR!

Authoring a published article provides a uniquely satisfying and visible feeling of accomplishment. To provide a more tangible benefit, however, HP gives away three free HP-32E hand-held calculators to Feature Article authors in each COMMUNICATOR/1000 issue! Authors are divided into three categories. A calculator is awarded to the author of the best Feature Article in each of the author categories. The three author categories are:

1. HP 1000 Customers;

2. HP field employees;

3. HP division employees.

Each author category is judged separately. A calculator prize will be awarded even if there is only one entry in an author category.

Feature Articles are judged on the following bases: (1) quality of technical content; (2) level of interest to a wide spectrum of COMMUNICATOR/1000 readers; (3) thoroughness with which subject is covered; and, (4) clarity of presentation.

What is a Feature Article? A Feature Article meets the following criteria:

1. Its topic is of general technical interest to COMMUNICATOR/1000 readers;

2. The topic falls into one of the following categories —

   OPERATING SYSTEMS
   DATA COMMUNICATIONS
   INSTRUMENTATION
   COMPUTATION
   OPERATIONS MANAGEMENT
   LANGUAGES

3. The article covers at least two pages of the COMMUNICATOR/1000, exclusive of listings and illustrations (i.e., at least 1650 words).

There is a little fine print with regard to eligibility for receiving a calculator; it follows. No individual author will be awarded more than one calculator in a calendar year. In the case of multiple authors, the calculator will be awarded to the first listed author of the winning article. An article which is part of a series will compete on its own merits with other articles in the issue. The total of all articles in the series will not compete against the total of all articles in another series.

All winners of calculators will be announced in the issue of the COMMUNICATOR/1000 in which their articles appear. Again, all Feature Articles are judged by an impartial panel of three DSD Technical Marketing Engineers.

A SPECIAL DEAL IN THE OEM CORNER

When an HP 1000 OEM writes a Feature Article that is not only technically detailed and insightful but also application-oriented as opposed to theoretical, then that OEM may ask that the article be included in THE OEM CORNER. A Feature Article included in THE OEM CORNER may contain up to 150 words of pure product description as well as a picture or illustration of the OEM'S product or its unique contribution. HP's objective is twofold: (1) to promote awareness of the capabilities HP 1000 OEMs' products among all HP 1000 users; and, (2) to publish an article of technical interest and depth.

IF YOU'RE PRESSED FOR TIME . . .

If you are short of time, but still have that urge to express yourself technically, don't forget the COMMUNICATOR/1000 BIT BUCKET. It's the perfect place for a short description of a routine you've written or an insight you've had.

THE MECHANICS OF SUBMITTING AN ARTICLE

If at all possible please submit an RTE File containing the text of your article recorded on a Minicartridge (preferrably) or on a paper tape along with the line printer or typed copy of your article. This will help all of us to be more efficient. The Minicartridge will be returned to you promptly. Please include your address and phone number along with your article.

All articles are subject to editorship and minor revisions. The author will be contacted if there is any question of changing the information content. Articles requiring a major revision will be returned to the author with an explanatory note and suggestions for change. We hope not to return any articles at all; if we do, we would like to work closely with the author to improve the article. HP does, however, reserve the right to reject articles that are not technical or that are not of general interest to COMMUNICATOR/1000 readers.

Please submit your COMMUNICATOR/1000 article to the following address:

Editor, COMMUNICATOR/1000
Data Systems Division
Hewlett-Packard Company
11000 Wolfe Road
Cupertino, California 95014
USA

The Editor looks forward to an exciting year of articles in the COMMUNICATOR/1000.

With best regards,

The Editor

EDITOR'S DESK

LETTERS TO THE EDITOR

Dear Ms. Editor:

John Pezzano's article "Using Memory behind your FORTRAN Program" (Vol. 5, Issue 2) gave me the idea of trying a solution on my own.

My proposition is to place the extendable memory at the end of the program during the loading process. This can be done as shown in the LOADR-Comand-File. The supported routine LIMEM returns the available memory in LEN, which can be added to the dimension of your variable.

Variations of this procedure could be applied to load any type of program, even segmented programs.

Yours truly,

Ernst Stelzer
AG GRELL
Max-Planck-Institut fur Biophysik
Heinrich-Hoffmann-StraBe Nr. 7
D-6000 Frankfurt am Main 71
West - Germany (BRD)

Dear Mr. Stelzer,

Thanks for your input. For a related bit bucket article, see the contribution from George Wynne of the U.S. Naval Ordnance Station.

Regards,

Ms. Editor

```
&PROG  T=00004 IS ON CR00031 USING 00003 BLKS R=0000

0001   FTN4X,L
0002         PROGRAM PROG()
0003         COMMON /MEM/ MEM(10)
0004         INTEGER FWAM
0005         LU = LOGLU(I)
0006         CALL LIMEN(0,FWAM,LEN)
0007         WRITE(LU,5010) LU, FWAM, LEN
0008         DO 10 I = 1, LEN + 10
0009           MEM(I) = 32767 - I
0010   10    CONTINUE
0011         DO 20 I = LEN + 10, 1, -5
0012           WRITE(LU,5030) I, (MEM(I+J), J = 0, 4)
0013   20    CONTINUE
0014   5010  FORMAT("LU"I3"  FWAM"I7"  LEN"I7)
0015   5030  FORMAT(6(I5,2X))
0016         END
```

```
&PROGD T=00004 IS ON CR00002 USING 00002 BLKS R=0000

0001  FTN4X,L
0002          BLOCK DATA
0003          COMMON /MEM/ MEM(10)
0004          END



/PROG  T=00004 IS ON CR00031 USING 00002 BLKS R=0000

0001  SZ,22
0002  REL,%PROG
0003  SEARCH
0004  REL,%PROGD
0005  /E


What to do !

   - Compile each element
   - RUN,LOADR,/PROGD
   - RUN,PROG

See what happens !


PROG    26042 26233

LOGLU   26234 26311   92067-1X297 REV.2013 790228
.EIO.   26312 27526   24998-1X329 REV.2101 800929
FMTIO   27527 30760   24998-1X328 REV.2101 800929
.FMCV   30761 33223   24998-1X333 REV.2101 800709
.IOER   33224 33337   24998-1X321 REV.2101 800731
.UFMP   33340 33352   24998-1X296 REV.2101 800731
.EXIT   33353 33426   24998-1X320 REV.2101 800731
PNAME   33427 33474   92068-1X035 REV.2013 771121
.IOCL   33475 33576   24998-1X305 REV.2101 800731
.IOCM   33577 33642   24998-1X327 REV.2101 801007
.FIOI   33643 33730   24998-1X322 REV.2126 810326
LIMEM   33731 33751   92067-1X477 REV.2013 790126
REIO    33752 34076   92067-1X275 REV.2013 790316
ER0.E   34077 34077   24998-1X249 REV.2001 750701
.OPN?   34100 34123   24998-1X325 REV.2101 800803
PAU.E   34124 34124   24998-1X254 REV.2001 750701
MEM     34125 34136


 5 PAGES RELOCATED     22 PAGES REQ'D     NO PAGES EMA        NO PAGES MSEG
LINKS:BP     PROGRAM:BG    LOAD:TE    COMMON:NC
 /LOADR:PROG    READY AT  9:55 AM  WED., 14  OCT., 1981

 /LOADR:$END
```

EDITOR'S DESK

Dear Ms. Editor:

I read with interest the article by the group in Pisa in Volume 5, Issue 3 regarding user written I/O routines for HP 1000 computers. We have been using similar techniques at Stanford for many years and have found them to be most useful. There are a class of devices which operate asynchronously on a demand-response basis for which this technique can be slightly modified and used in a multiprogramming environment. Instead of disabling the operating system (by a call to $LIBR) for the duration of the entire transaction, we do so only long enough to exchange some unit of information such as a byte or line. Careful programming is necessary to assure that no interrupts will occur when the operating system is re-enabled. Under such circumstances, it is possible to overlap I/O from/to a specialized device with system I/O to/from discs, etc. This allows the user to take advantage of system facilities such as the file manager, as well as providing protection of the other system resources. If real-time requirements are not stringent, multi-user operation of the system also is still possible.

An example application of this technique is the the optimum usage of the Versatec printer/plotter which we use extensively in several of our systems. When operated with a driver, the Versatec operates at approximately one-half speed while plotting due to the system overhead entailed in processing interrupts. We discovered that by turning off RTE via a call to $LIBR and outputting directly to the printer/plotter, we could operate it at close to full speed. If one is careful to make sure that no interrupts are left pending at the time $LIBX is called to complete a unit of input/output processing, then the device can be operated in a full system context.

We have also used this technique to operate real-time devices such as a Summagraphics digitizing tablet and a DeAnza image array processor. These devices preclude multiprogramming but full operating system resources are used, which simplified the programming task necessary to interface these special devices.

The principal discovery which I would like to report is a way to greatly decrease the overhead incurred when using this technique. The call from a user program to $LIBR requires the system to process a memory protect interrupt with the associated overhead. The call to $LIBR results in several changes in system status including the disabling of interrupts and memory protect which in turn allows input/output instructions to be executed by the user's program. If it were possible to execute input/output instructions in the user program without calling $LIBR, then direct input/output could be accomplished with absolutely no overhead.

In this regard, the 12892B Memory Protect Module Installation and Service manual makes extremely interesting reading. There is a jumper labelled "SEL 1" which is normally absent from the board. In this condition, input/output instructions are only allowed to select code 1 when memory protect is enabled. By inserting a jumper in this position, input/output instructions to all select codes are allowed, even when memory protect is enabled. The net result of installing this jumper is the possibility to do input/output from a user program without drivers and without the overhead of calling $LIBR.

Certain caveats need to be observed unless the interrupt system is explicitly disabled (which is perfectly feasible using a CLF 0 command). It will be necessary to prevent interrupts for some input/output interfaces. This is possible by never issuing an STC command. Sometimes it is necessary to use an STC to strobe data to or from the interface board (e.g., the microcircuit interface board). If the STC is *immediately* followed by a CLC command no interrupt will be generated because the hardware prevents an interrupt immediately following the STC command. In reading data from the 12966 board, it is necessary to check for the flag being set with control also set. In this circumstance, I programmed a short loop which turned off the entire interrupt system for a few microseconds each time this test was performed.

The uses of this technique are only limited by the programmers imagination. We have used it on several systems with quite different applications over a period of two years with absolutely no problems. I hope that it will be useful to some of your readers.

Sincerely,

William Sanders
Cardiology Division
Stanford University Medical Center

P.S. I also have a simple hardware modification that can be performed to a board in the 2100 computer that accomplished the same result. I will be happy to communicate it to any intrested readers.

Dear Mr. Sanders,

Thank you for the additional information.

Regards,

Ms. Editor

EDITOR'S DESK

Dear Ms. Editor:

The following is a solution to a problem concerning use of 2 or more HP 2240A Measurement and Control Processors on the same HP-IB interface card using DVR37 W/SRQ.

Program to initialize one of the 2240's to its power on state would intermittently hang up on the call to SRQ (dummy interrupt handler).

The hung up program could be simulated by:

1. Turn power OFF on 2240 not to be initialized.

2. Turn power ON on 2240 not to be initialized.

3. Run Program (ABORX)

The solution to this problem seems to be provided mostly by HP SE Todd Field as follows:

1. Initialize HP-IB 2240 LUs as follows:

```
:CN,LU,157000B
```

2. Set up dummy interrupt handler program that does nothing but has an ID segment.

3. Run following initialization program (see attachment for listing).

The key to the solution seems to be the call to (STATS) which is part of the HP-IB library. This subroutine is mentioned in passing but not documented in the 2240 User's Manual (P/N 2240-93001 Dec 1979).

Very truly yours,

Lewis J. Metzger
Abex Corporation
Mahwah, New Jersey

LISTING OF 2240 POWER ON INITIALIZATION PROGRAM


```
FTN4X,L
      PROGRAM ABORX
      INTEGER  HPIB,MACS1,MACS2,INTPG
      DATA INTPG/5,'INTPG '/
C
      CALL ABRT(HPIB,2)           !ABORT ALL ACTIVITY ON THIS HP-IB
C
      CALL CLEAR(MACS1,1)          !TERMINATE ANY PENDING
      CALL CLEAR(MACS2,1)          !REQUEST ON EACH 2240
C
      CALL STATS(MACS1,ISTAT)      !SERIAL POLL EACH 2240
      CALL STATS(MACS2,ISTAT)      !FOR INTERRUPT
C
      CALL SRQ(MACS1,16,INTPG)     !SET UP DUMMY
      CALL SRQ(MACS2,16,INTPG)     !INTERRUPT HANDLER PROGRAM
      END
```

Dear Mr. Metzger,

Thank you for your solution and thanks to HP System Engineer Todd Field, Woodbury, New York.

Sincerely,

Ms. Editor

WHO'S LOGGED ON?

by Dan Wagner/E. I. Dupont,
New Cumberland, PA

Listed below is a FORTRAN utility used to determine who is logged on to the system in a Session environment. The information that is reported is retrieved from the system accounts file (+@CCT!). The program will default to the local node number, but it can also list the users logged on to a remote node using DS/1000-IV.

Compile using FTN4X:

```
FTN4X,&WHO,1,%WHO
```

Loader command must include SSGA:

```
LOADR,,%WHO,,,SS
```

Programmer: Dan Wagner
E. I. Dupont
Berg Electronics Div.
515 Fishing Creek Road
New Cumberland, Pa. 17070

```
FTN4X
      PROGRAM WHO (),Who is logged on the system.
C=====================================================================C
C                                                                     C
C WW          WW  HH          HH  OOOOOOOOOOOO                       C
C WW          WW  HH          HH  OOOOOOOOOOOO                       C
C WW          WW  HH          HH  OO          OO                     C
C WW          WW  HH          HH  OO          OO                     C
C WW          WW  HH          HH  OO          OO                     C
C WW          WW  HHHHHHHHHHHHH   OO          OO                     C
C WW    WW    WW  HHHHHHHHHHHHH   OO          OO                     C
C WW   WWWWW  WW  HH          HH  OO          OO                     C
C WW WW  WW WW  HH          HH  OO          OO                     C
C WWWW     WWWW  HH          HH  OO          OO                     C
C WWW       WWW  HH          HH  OOOOOOOOOOOO                       C
C WW          WW  HH          HH  OOOOOOOOOOOO                       C
C                                                                     C
C=====================================================================C
C                                                                     C
C REVISION LIST:                                                      C
C                                                                     C
C --DATE-- --BY-- -- D E S C R I P T I O N --                         C
C                                                                     C
C 10/14/81 D.A.W. -ORIGINAL VERSION-                                  C
C 11/26/81 D.A.W. -ADD REMOTE CAPABILITY USING DS/1000-IV             C
C                                                                     C
C=====================================================================C
```

```
C                                                                       C
C          This program will display a brief report of the users        C
C    logged on the system.  This is done by accessing the system        C
C    accounts file (+@CCT!).  The program will default to a local       C
C    DS/1000 node, but will also report the users on a remote node.     C
C    The syntax for executing the program is as follows:                C
C                                                                       C
C                         :WHO [ ,LILU [ ,NODE ]]                       C
C                                                                       C
C                    where:                                             C
C                           LILU - List LU which defaults to 1.         C
C                           NODE - DS/1000 node # which defaults        C
C                                  to -1 (local node).                  C
C                                                                       C
C           The program has a few items in the code which each          C
C    user might want to customize for their particular system.          C
C                                                                       C
C           1).  Line 74 : ICR(1) /2/ - This is the CRN where           C
C                          the system accounts file (+@CCT!)            C
C                          resides.                                     C
C                                                                       C
C           2).  Lines 82 & 83 : These are the valid node numbers       C
C                                in the network.                        C
C                                                                       C
C           The format of the output is as follows:                     C
C                                                                       C
C    SESSION     USER                    LOG-ON TIME                    C
C    -------     --------------------    ---------------------          C
C      50        USER.GROUP              12:31:12 PM   5  OCT           C
C                                                                       C
C=======================================================================C
C

C
      IMPLICIT INTEGER (A-Z)
C
      INTEGER*4 LOGON
C
      REAL MONTH
C
      DIMENSION IDCB(144),INAM(3),IBUF(128),IDIR(128)
      DIMENSION UNAM(5),GNAM(5),XLOG(2),ICR(2),IPARM(5)
      DIMENSION IOUT1(6),IOUT2(16),IOUT(28),IERBF(24)
C
      EQUIVALENCE (XLOG(1),LOGON),(IPARM(1),ILU)
      EQUIVALENCE (IPARM(2),INODE)
C
      DATA ICR(1) /2/, ICR(2) /-1/
      DATA ISC /-31178/, INAM /6H+@CCT!/, SPACE /2H  /
      DATA AM /2HAM/, PM /2HPM/
C
      CALL RMPAR (IPARM)
C
C *** CHECK FOR VALID NODE FOR DS/1000
```

```
C
      IF (INODE .EQ. 1) ICR(2) = 1
      IF (INODE .EQ. 2) ICR(2) = 2
C
C === Open the accounts file.
C
      CALL DOPEN (IDCB,IERR,INAM,1,ISC,ICR)
      IF (IERR .LT. 0) GO TO 90
C
C === Read the account file header.
C
      CALL DREAD (IDCB,IERR,IBUF,128,LEN,1)
      IF (IERR .LT. 0) GO TO 90
C
C === Get beginning and end of active session table.
C
      BACT = IBUF(1)
      EACT = IBUF(2)
C
C === Get directory location.
C
      DREC = IBUF(5)
C
C === Write out heading.
C
      WRITE (ILU,5)
5     FORMAT (//,"SESSION",4X,"USER",20X,"LOG-ON TIME",/,7"-",4X,20"-",
     +           4X,22"-")
C
C === Read 128 words of active session table.
C === Each session uses 4 words.
C
10    CALL DREAD (IDCB,IERR,IBUF,128,LEN,BACT)
      IF (IERR .LT. 0) GO TO 90
C
C === There are 32, 4-word entries in each 128 word buffer.
C
      DO 60 I=1, 32

C
C === Step through the buffer 4 words at a time.
C
        IPTR = (I-1)*4 + 1
C
C === Word 1 of each 4-word entry is the session LU or 0 if free.
C
        IF (IBUF(IPTR) .EQ. 0) GO TO 60
C
        SESLU   = IBUF(IPTR)
C
C === XLOG is the account logon time packed into two integer words.
C     The packing is explained later in this program.
C
        XLOG(1) = IBUF(IPTR+1)
        XLOG(2) = IBUF(IPTR+2)
C
C === ILOC is the record pointer for the user account file directory.
C
        ILOC    = IBUF(IPTR+3) + 1
        XREC    = DREC
```

```
C
C === There are 8, 16-word entries per 128-word record.
C === ILOC is decremented by 8 and XREC is incremented by 1
C === until ILOC is less than or equal to 8.  XREC is then
C === pointing to the proper record & ILOC is pointing to the
C === proper entry in that record.
C
20      IF (ILOC .LE. 8) GO TO 30
C
        XREC = XREC + 1
        ILOC = ILOC - 8
        GO TO 20
C
C === Read the record which contains the user account file directory.
C
30      CALL DREAD (IDCB,IERR,IDIR,128,LEN,XREC)
        IF (IERR .LT. 0) GO TO 90
C
C === IPTR points to the specific 16-word entry in the 128-word buffer.
C
        IPTR  = (ILOC-1)*16 + 1
C
C === Get the number of characters in the user name.
C
        ULEN  = (IDIR(IPTR) .AND. 177400B) / 256
C
C === Retrieve the user name and group name.
C
        DO 40 J=1, 5
          UNAM(J) = IDIR(IPTR+J)
          GNAM(J) = IDIR(IPTR+J+5)
40      CONTINUE
C
C === Convert SESLU to ascii and store in IOUT1.
C
        ENCODE (11,50,IOUT1) SESLU
50      FORMAT (I4,"       ")


C
C === Unpack XLOG into year, julian day of the year,
C === hours, minutes, and seconds.
C === XLOG is packed as follows:
C
C
C                    (YEAR -
C                      1978)   MINUTES     SECONDS
C                   !_ _ _ _!_ _ _ _ _ _!_ _ _ _ _ _!
C      XLOG(1)      !_!_!_!_!_!_!_!_!_!_!_!_!_!_!_!_!
C                   !       !           !           !
C
C
C                     JULIAN DAY OF YEAR     HOURS
C                   !_ _ _ _ _ _ _ _ _ _ _!_ _ _ _ _!
C      XLOG(2)      !_!_!_!_!_!_!_!_!_!_!_!_!_!_!_!_!
C                   !                     !         !
C
C
C === Strip off year.
```

```
C
        JYEAR = ((XLOG(1) .AND. 170000B) / 4096) + 1978
C
C === Strip off julian day of the year.
        JDAY = (XLOG(2) .AND. 177740B) / 32
C
C === Convert year + julian day of year into year, month, date.
C
        CALL JDATE (JYEAR,JDAY,MONTH,DAY)
        AMPM = AM
C
C === Strip off the hours in 24-hour format.
C
        HOURS = (XLOG(2) .AND. 37B)
C
C === Convert to 12-hour format using A.M. and P.M.
C
        IF (HOURS .LE. 12) GO TO 53
          AMPM = PM
          HOURS = HOURS - 12
C
C === Strip off seconds and minutes.
C
53      SEC = (XLOG(1) .AND. 77B)
        MIN = (XLOG(1) .AND. 7700B) / 64
C
C === Manipulate all the information into one buffer (IOUT).
C
        ENCODE (32,54,IOUT2) GNAM,HOURS,MIN,SEC,AMPM,DAY,MONTH
54      FORMAT (".",5A2,I2,":",I2,":",I2," ",A2,"  ",I2,"  ",A4)
C
        CALL SMOVE (IOUT1,1,11,IOUT,1)
C
        CALL SMOVE (UNAM,1,ULEN,IOUT,12)
C
        START = ULEN + 12
C
        CALL SMOVE (IOUT2,1,11,IOUT,START)

C
        START = START + 11
56      IF (START .GE. 36) GO TO 58
        CALL SPUT (IOUT,START,SPACE)
        START = START + 1
        GO TO 56
C
58      CALL SMOVE (IOUT2,12,32,IOUT,36)
C
C === Write the buffer.
C
        CALL EXEC (2,ILU,IOUT,28)
C
60    CONTINUE
C
C === Read the next 128 words of active session table.
C
      BACT = BACT + 1
      IF (BACT .LT. EACT) GO TO 10
```

```
C
      WRITE (ILU,80)
80    FORMAT (" ")
      GO TO 110
C
C === DISPLAY ERROR MESSAGE
C
90    CALL DSERR (IERBF)
      WRITE (ILU,100) IERBF
100   FORMAT (24A2)
C
C === Close account file.
C
110   CALL DCLOS (IDCB,IERR)
C
      END
C
C
C
C=====================================================================C
C                                                                     C
      SUBROUTINE JDATE (JYEAR,JDAY,MONTH,IDAY),Convert Julian to year, m
     +onth, day.
C                                                                     C
C=====================================================================C
C
C
C === Change year + julian day of year to a year, month, day.
C
      INTEGER LEN(12)
C
      REAL MONTH,MNTH(12)
C
      DATA LEN /31,28,31,30,31,30,31,31,30,31,30,31/
      DATA MNTH /4HJAN ,4HFEB ,4HMAR ,4HAPR ,4HMAY ,4HJUNE,
     +           4HJULY,4HAUG ,4HSEPT,4HOCT ,4HNOV ,4HDEC /
C
C
C === If JYEAR is not divisable by 4 then it is a leap year
C === and February is changed to 29 days.

C
      ILEAP = MOD(JYEAR,4)
      IF (ILEAP .EQ. 0) LEN(2) = 29
C
      DO 100 I=1, 12
C
        JDAY = JDAY - LEN(I)
        IF (JDAY .GT. 0) GO TO 100
C
          MONTH = MNTH(I)
          IDAY = JDAY + LEN(I)
C
        GO TO 200
C
100   CONTINUE
C
200   RETURN
      END
```

WHO AM I?

by J.L. DeSchutter/DISTRIGAZ
Brussels, Belgium

The objective of this routine is to give information about the session you are running under. Application programs may wish to know capability, user and group name. This is very useful for personalized prompting or for logging the use of a key program.

With a simple Fortran call, the routine provides user name, group name and capability level. This routine uses information described in Appendix J of the RTE-IVB System Manager's Manual (92068-90006). Note that this subroutine can easily be extended to provide additional information (e.g. group and user ID).

Lines 33-35 contain the name, location and security code of the Accounts file (+@CCT!). Future revisions of software or specific configurations may require that those lines be changed.

```
ASMB,L
       NAM IDENT,7  IDENTIFY USER & GROUP (ASCII) REV. 2026
       ENT IDENT
       EXT OPEN, LOGLU,CLOSE,READF,.ENTR
       EXT .MVW,ICAPS
*****************************************************************
* CALL IDENT(IUSN,IGRN,ICAP)
*    IGRN   5 WORDS BUFFER
*    IUSN   5 WORDS BUFFER
*    ICAP   INTEGER
*
*    GRN = GROUP NAME (ASCII)
*    USN = USER NAME  (ASCII)
*    ICAP= CAPABILITY LEVEL  (INTEGER)
*
*    IF CALL FAILS ALL PARAMETERS SET TO -1
*****************************************************************
A      EQU 0
B      EQU 1
D128   DEC 128
D5     DEC 5
D4     DEC 4
D16    DEC 16
OPT    OCT 1          NON EXCLUSIVE OPEN
DCB    BSS 144
BUF    BSS 128
LU     NOP            MY TERMINAL LU
GROU   NOP
LEN    NOP
OFSET  NOP
IER    NOP
DUMMY  NOP
NAM    ASC 3,+@CCT!  *ACCOUNT FILE NAME
SEC    DEC -31178    *SECURITY CODE
CAR    DEC 2         *CARTRIGE LABEL
.USN   NOP
.GRN   NOP
.CAP   NOP
*
```

```
************  HERE WE START ******************************
*
IDENT NOP
      JSB .ENTR     GET PARAMETERS ADDRESSES
      DEF .USN
      JSB OPEN      OPEN ACCT FILE NON EXCLUSIVE
      DEF E1
      DEF DCB
      DEF IER
      DEF NAM
      DEF OPT
      DEF SEC
      DEF CAR
E1    LDA IER
      SSA           MUST BE POSITIVE
      JMP ERROR
      CLA,INA
      STA BLOC      CURRENT RECORD TO BE READ
      JSB READ      READ ONE BLOC
      JSB LOGLU
      DEF *+2
      DEF LU        SESSION LU #
*SEARCH FOR MY ACCT IN SESS TABLE
      LDA BUF+4     RECORD ADDRESS OF ACCT DIRECTORY
      STA GROU
      LDA BUF       RECORD # OF ACTIVE SESSION TABLE
      STA BLOC
      JSB READ      READ SESSION TABLE
      CLA
      STA OFSET
BOU   LDB .BUF      SEARCH FOR MY AST
      ADB OFSET
      LDA B,I       IS FIRST WORD = TO MY LU
      CPA LU
      JMP FIND      YES, WE FOUND IT
      LDA B,I       CALCULATE SUM OF FOUR ACB WORDS
      INB
      ADA B,I
      INB
      ADA B,I
      INB
      ADA B,I
      SZA,RSS       IF THE SUM IS = TO ZERO
      JMP ERROR     END OF INFO AND SESS NOT FOUND
      LDA D4
      ADA OFSET
      CPA D128      DO WE REACH END OF CURRENT RECORD
      JMP NEWBL     YES READ A NEW ONE
      STA OFSET
      JMP BOU
NEWBL ISZ BLOC      READ NEXT DISC RECORD
      CLA
      STA OFSET     RESET OFSET
      JSB READ
      JMP BOU
FIND  INB           THE LAST WORD OF OUR ASB
      INB           IS OUR ENTRY
      INB           IN ACCT DIRECTORY
      LDA B,I
```

```
       MPY D16       EACH ENTRY IS 16 WORDS LONG
       STA OFSET     AND FIRTS ENTRY HAS NUMBER ZERO
       DIV D128      CALCULATE THE CORRESOPNDING RECORD #
       STB OFSET     USE THE REMAINDER AS OFSET
       ADA GROU
       STA BLOC
       JSB READ
       LDA OFSET
       ADA .BUF
       INA
       LDB .USN      THE 2ND WORD IS USER NAME
       JSB .MVW      MOVE IT
       DEF D5
       NOP
       LDB .GRN      MOVE GROUP NAME
       JSB .MVW
       DEF D5
       NOP
****** READ CAPABILITY LEVEL WITH ICAPS CALL ******************
       JSB ICAPS
       DEF *+2
       DEF DUMMY
       STA .CAP,I
*****************************************************************
CLO    JSB CLOSE     THE END
       DEF *+2
       DEF DCB
       JMP IDENT,I
**
* READ ONE 128 WORDS RECORD
* RECORD # IS IN BLOC
**
BLOC   NOP
READ   NOP
       JSB READF
       DEF EE
       DEF DCB
       DEF IER
.BUF   DEF BUF
       DEF D128
       DEF LEN
       DEF BLOC
EE     LDA IER
       SZA
       JMP ERROR
       JMP READ,I
**
*
**
ERROR  NOP           PUT -1 IN ALL PARAMETERS
       CCA
       STA .GRN,I
       STA .USN,I
       STA .CAP,I
       JMP CLO       RETURN IMMEDIATELY
       END
```

HOW LONG HAVE I BEEN HERE?

by William J. Loye/Buckbee-Mears,
St. Paul, MN

The program ACTTI was written to facilitate user and group time reporting in the RTE file manager environment. The program opens the accounts file in a shared read-only mode and accesses the user CPU and connect time, writing to any output lu (default=6). The most common use will probably be to write to a file that can be post-processed by a separate user program or by IMAGE. (e.g.):

```
:SL,6,TIME::MN,BO
:RU,ACTTI,,6
:CS,6,RW
:RU,TRACK     (a user pgm to condense and store on monthly basis)
:CS,6,EN
:TR
```

Because this program opens and reads the accounts file, it would not take much modification to get it to print all sorts of information, including passwords. Therefore, it should be kept on a private cartridge or cassette.

The program works by first reading word 5 of the accounts header. This is the pointer to the location of the accounts file directory. The program then reads the account file directory sequentially keeping the record in an 128 word buffer (IBUF). Each 16 word logical record of the account file directory is checked to see if it is a valid user (word 12 .GT. 0). If it is, the user entry record number (as pointed to by word 15 of IBUF) is used to read the user entry record into a 128 word buffer (JBUF). The CPU and connect time will be pulled from either the first or second half of this buffer, depending on bit 15 of the record number pointer (word 15). User times will be double integer in words 25, 26, 27,and 28 of the entry.

Originally, the program used arrays to store entry record pointers thus minimizing the reads from the disk. However, because this needed arrays, maximum checks, and other fooling around, and because the accounts file is a type 1 file, that method was dropped in favor of the present one. I make no effort to see if the desired entry record is already read and stored because the overhead in doing this is more expensive than just reading (or rereading) each record.

All of the information needed to write (modify, enhance, etc.) this program will be found in Appendix J of the System Manager's manual (92068-90006).

This program can be compiled with either FTN4 or FTN4X. It will load in about 9K.

```
FTN4X,Q
      PROGRAM ACTTI(3,80), 10/26/81 BL REV 1.10
C SM
C     PGM TO READ USER/GROUP TIME FROM ACCT FILE
C     AND PUT OUTPUT TO LU6, OR OPTIONALLY, ANY LU.
C     IF LU IS SL'ed TO A FILE, FILE MAY BE READ BY
C     ANY OTHER PGMS TO KEEP RUNNING TOTALS OF
C     USER/GROUP USAGE.
C
C     THIS PROGRAM OPENS AND READS THE ACCOUNT FILE,
C     AND CAN BE MODIFIED TO OUTPUT PASSWORDS, ETC.
C     IT IS POTENTIALLY A STOOL-PIDGEON PGM.
C
C     PLEASE SEE PAGES J-3 THRU J-12 OF SYS. MNGR. MANUAL.
C     FOR ASSOCIATED DOCUMENTATION.
C
```

```
C **
C       10/26/81     BILL LOYE
C                    BUCKBEE-MEARS CO.
C                    245 EAST SIXTH ST.
C                    ST.PAUL  MN.  55101
C --
C       11/ 2/81  BL   1.10  ADD JGRP OPTION TO SUPPRESS GROUP PRINTING
C **
C     OUTPUT:  USER      CPU     CONNECT     USER    GROUP    GROUP    JULIAN
C                ID        MINUTES          NAME     NAME      ID      DATE
C              (I6)    (F13.1) (F13.1) 2X (5A2)   (5A2)     (I6)     (I6)
C
C **
C       LU   1  INPUT/OUTPUT   USER CONSOLE
C            6         OUTPUT   DEFAULT PRINTER
C       FMP      +@CCT!:-31178  ACCOUNTS FILE
C **
C        RU,ACTTI,,LUOUT
C           WHERE LUOUT IS THE LU OF THE OUTPUT FILE
C             (DEFAULT IS 6)
C       ( note luout is second parameter !)
C
C**
C     OP SYS:    RTE IVB
C     COMPILES: FTN4X,FTN4
C     LOADS:     STANDARD LOAD
C**
C
        DIMENSION IDCB(144),IBUF(128),JBUF(128),IPRM(5)
        DATA ITERM/1/,ILST/6/,IWRD1/32000/
C
C * * * USER DEFINABLE:  JGRP IS USED TO TURN GROUP ACCOUNT PRINTING
C                        ON (JGRP=1)  OR  OFF (JGRP=0)
        DATA JGRP/0/
C
        CALL RMPAR(IPRM)
        IF(IPRM(2).EQ.6.OR.IPRM(2).LT.1) GO TO 5
        ILST=IPRM(2)
        WRITE(1,3) ILST
     3  FORMAT(' OUTPUT LIST LU=',I3)
C
C OPEN ACCOUNT FILE (NAME AND SEC. CODE MAY BE DIFFERENT FOR YOUR SYSTEM)
C   NAME=+@CCT!, IOPTN=5B (BIT0=OPEN SHARED, BIT2=FORCE TYPE 1) pp. 3-29 pgmr re
     5  CALL OPEN(IDCB,IER,6H+@CCT!,5,-31178)
        IF(IER.EQ.1) GO TO 10
         WRITE(1,7) IER
     7   FORMAT('  COULDNT OPEN ACCOUNTS FILE. IER=',I3)
         STOP 7
C
C
C
C        READ HEADER RECORD AND GET IMPORTANT POINTERS
    10 CONTINUE
       CALL READF(IDCB,IER,IBUF,128,IL,1)
       IF(IER.EQ.0) GO TO 15
        WRITE(1,12) IER
    12  FORMAT('  ERROR ON HEADER READ. IER=',I5)
        JER= 12
        GO TO 900
C
```

```
C
   15 LUSER=4096-IBUF(23)
      LGRP=IBUF(24)
      LOCDIR=IBUF(5)
      WRITE(1,17) LUSER,LGRP
   17 FORMAT(I8,' USERS,',I8,' GROUPS.')
      CALL EXEC(11,IPRM)
      JDATE=IPRM(5)
C
C ***************************************************************
C
C       READ  ACCOUNT FILE DIRECTORY  (J-11)  16 WORD LOG REC
C        LOOP THROUGH  ACCOUNT FILE DIRECTORY (J-11) STORING ONE PHYSICAL
C        RECORD (128 WORDS).     NOTE THAT EACH PHYSICAL RECORD CONTAINS
C        EIGHT 16 WORD LOGICAL RECORDS.
C      (IWRD1 IS USED AS POINTER TO WORD 1 OF CURRENT LOGICAL RECORD)
   20 CONTINUE
      IWRD1=IWRD1+16
      IF(IWRD1.LE.126) GO TO 30
C        read next account file physical record
       CALL READF(IDCB,IER,IBUF,128,IL,LOCDIR)
       IF(IER.EQ.0) GO TO 25
        WRITE(1,23) IER,LOCDIR
   23   FORMAT('  ERROR',I5,'   ON READ OF ACC.DIR.  RECORD #',I5)
        JER= 23
        GO TO 900
   25  CONTINUE
       IWRD1=1
       LOCDIR=LOCDIR+1
C
C
C       read account entry   (64 words per logical block)
C        logical record may be in first or second half of 128 word phys. rec.
C        if so, record pointers will be negative
   30 CONTINUE
C       directory ends at when ibuf(iwrd1) =0, ignored when less than 0
      IF(IBUF(IWRD1)) 20,100,31
C       user ( IUSER=1-4095)  or group ( IUSER=0)
   31 IGRP=IBUF(IWRD1+12)
      IUSER=IBUF(IWRD1+11)
      IF(IUSER.LT.1) GO TO 40
C       user account
      IREC=IAND(IBUF(IWRD1+14),7777B)
      IOFFST=25
      IF(IBUF(IWRD1+14).LT.0) IOFFST=89
      GO TO 50
C
C
C        group account
   40 IF(JGRP.EQ.0) GO TO 20
      IREC=IAND(IBUF(IWRD1+13),7777B)
      IOFFST=2
      IF(IBUF(IWRD1+13).LT.0) IOFFST=66
```

```
C
C
C
C       READ DIRECTORY ENTRIES AND PULL CPU,CONNECT TIMES
C        store 128 word logical record in array jbuf
C        ioffst is used to point to first or second log. rec. of phys. rec.
  50   CONTINUE
       CALL READF(IDCB,IER,JBUF,128,IL,IREC)
       IF(IER.EQ.0) GO TO 55
        WRITE(1,53) IER,IREC
  53    FORMAT('  ERROR',I5,'  ON READ OF ENTRY. RECORD #',I5)
        JER=53
        GO TO 900
C        CALCULATE CPU AND CONNECT MINUTES.  NOTE DOUBLE INTEGER
   55 CPU=FLOAT( IAND(JBUF(IOFFST+2),77777B)) * 65536. +
      +      FLOAT( IAND(JBUF(IOFFST+3),77777B)) +
      +     (FLOAT( IAND(JBUF(IOFFST+3),100000B))*(-1.))
        CONNCT=FLOAT( IAND(JBUF(IOFFST),77777B)) * 65536. +
      +      FLOAT( IAND(JBUF(IOFFST+1),77777B)) +
      +     (FLOAT( IAND(JBUF(IOFFST+1),100000B))*(-1.))
        CONNCT=CONNCT/60.
        CPU=CPU/6000.
        M=IWRD1+1
        N=IWRD1+10
C
C  * * * * OUTPUT SECTION.  MAY BE MODIFIED TO SUIT USER * * * * *
C         iuser,igrp are user,group numbers.
C         iuser,igrp are user,group numbers.
C         cpu,connct are cpu and connect minutes
C         ibuf(i),i=m,n  prints out user and group name (5A2 each)
C         jdate is julian date.
C        note present format is image/1000 compatable
       WRITE(ILST,58) IUSER,CPU,CONNCT,(IBUF(I),I=M,N),IGRP,JDATE
   58 FORMAT(I6,2F13.1,2X,10A2,2I6)
       GO TO 20
C ****************************************
C      END OF LOOP
C
C
C
C        ALL DONE. CLOSE UP FILE AND GO HOME.
  100  CONTINUE
       CALL CLOSE(IDCB)
       STOP
C        ERRORS.
  900  CONTINUE
       WRITE(1,901) IER,JER
  901  FORMAT(' FATAL ERROR. IER,JER=',2I5)
       STOP 777
       END
```

EXECUTING A PROCEDURE AFTER LOGOFF

by Bob Desinger/Hewlett Packard
Data Systems Division

Session Monitor controls access to the system and, under normal use, prevents you from using the system after you log off. Occasionally, however, you want to do something just after logoff, such as clearing your CRT screen. The program that follows shows how this can be done; its principles can be used to perform less trivial after-logoff functions.

At logoff, if Session Monitor finds any active programs invoked by your session, it prevents normal automatic logging off. To find these active programs, Session Monitor examines the last word of each ID segment. This session word identifies the session that invoked the program, or indicates that the system has started the program up.

This word can be changed with a subroutine that overwrites it. Jack Sadubin's DEATS subroutine from Volume IV, issue 5 of The Communicator does this by finding the ID segment address and overwriting the 32nd word past it. Or, alternatively, a supported subroutine DTACH alters the session word to indicate that the system (not a session) owns a program. This allows you to log off while your program is running, since Session Monitor won't find your session identifier in the active program's session word.

To execute the procedure, a transfer file runs a program which finds out the invoking terminal LU number, detaches from session, and then delays itself. During the delay, FMGR executes the next line of the transfer file to log you off. After logoff has completed and all logoff lines have appeared on the screen, the program wakes up and sends a screen-clearing string of escape codes to your CRT.

Call the transfer file EX. To log off and clear the screen from FMGR, simply enter TR,EX or :EX.

Your FMGR clone should not run the program, since it would wait for the program to finish (leaving active programs at logoff). RTE can run a program, bypassing FMGR, with its own RU command. RTE commands are issued through FMGR by prefixing them with SY, so the command in the transfer file to run the program is :SYRU. Running the program using RTE allows your FMGR clone to execute the next line of the transfer file and log off.

ID SEGMENT CONSIDERATIONS

Every program needs an assigned ID segment to run. FMGR automatically assigns one (if necessary) to the programs that it schedules, releasing it when the program terminates. RTE neither assigns nor releases. An ID segment must be previously assigned with the FMGR RP command in order to run a program from RTE.

LOADR assigns an ID segment to a program upon successful loading, which is released by Session Monitor at logoff if the program is dormant. Remember, though, that the clearing program is detached at logoff; its ID segment has no associated session and will not be released. It remains assigned until the system is rebooted or a FMGR OF command (or the RTE command OF,progName,8) is issued.

If an ID segment remains assigned to a program, the system is saved much overhead: searching for the type 6 file, opening, reading, closing, and other disc-dependent processes. An informal, unofficial guide is that any program run more than 10 times a day should have its own assigned ID segment (using the RP command). Inserting such a command in the WELCOM file ensures that an ID segment is allocated to the program at all times.

If ID segment supply is not critically short, leave the ID segment assigned to the program: put the RP command in WELCOM, and don't OF the program after its loading.

If ID segments are in critically short supply, each invocation of the clearing process should RP the program and then release its ID segment so another program can use it. The invoking transfer file can RP, but it cannot perform the OF since this would abort the program before it cleared the screen.

The program can release its own ID segment by issuing an RTE OF command with a MESSS call. The buffer passed to MESSS should be at least 14 words long, since RTE returns messages in this buffer. Of course, if the program releases its own ID segment, put the RP in the transfer file EX instead of in WELCOM and do your own OF after a successful load and SP of the program.

The destination CRT LU for the screen-clearing escape codes can be another obstacle. A standard EXEC/REIO write has only six bits for the destination LU in the control word. Six is not enough if a terminal's system LU number is greater than 63 (77 octal). Normal I/O goes through the Session Switch Table (SST), which maps system LU numbers into session LUs of 63 or less. The SST is part of Session Monitor, from which the program has detached. So using the SST and normal EXEC/REIO are off limits for this application.

The "extended EXEC" subroutine XLUEX accepts a two-word control parameter. The first word is the destination LU; bit 16 indicates whether the word contains a system LU or a session LU. If set, the local SST is bypassed and the message is sent to the system LU.

CUSTOMIZATION AND CAVEATS

This version moves the cursor to the bottom of the screen before clearing. Thus it completely clears the screen even when invoked from the middle of a block of text. An HP 2640/driver DVR00 terminal doesn't recognize the Home Down (escape F) sequence, so invoke this procedure from the last line of text on a 2640. Logoff is normally executed from the last line anyway, so this should be no hardship.

Different versions of output are possible. The version below rolls lines off the visible portion of the screen rather than clearing them. This allows you to later read any logoff lines, such as MESSAGES WAITING. Also, you might be able to tell who just used the terminal even after the screen is clear, assuming the terminal has enough memory.

If some accounts on your system release private cartridges at logoff, create two exiting transfer files. One, called EXRP, would exit with an :EX,RP line, while EXSP would exit with an :EX,SP line. Other situations might call for other files named, for example, EXRG or some other permutation of EX; each user can invoke the one appropriate for his account. If some users log off with EX,SP and others use just EX, the logoff file can be named simply EX and contain the exit command with the largest number of parameters. FMGR ignores any extra parameters (like ",SP") if it only needs EX to exit.

Further information on DTACH and Session Monitor's logoff handler LGOFF is in Chapter 14 of the RTE-IVB Technical Specifications Reference Manual (92068-90013). A description of the XLUEX call is in Chapter 3. Specific FMGR and RTE command syntax is explained in Chapters 3 and 4 of the RTE-IVB Terminal User's Reference Manual (92068-90002).

```
FTN4X,L,C,d
      PROGRAM CLR (4,51),BD Clear the 26xx Screen After Logoff
*
*   pgrmr:  Bob Desinger, HP Data Systems Division
*    date:  82.02.16
* purpose:  Cleans the CRT screen after logoff.
*           Used under RTE-IVB and RTE-6/VM for 264x and 262x CRTs.
*
*  design:  Get the system LU number of the terminal (for I/O).
*           Detach from session, allowing logoff.
*           Wait so logoff messages can appear,
*             then clean off the CRT screen.
*           Depending on the compiling option, it either releases its
*             ID segment or keeps it allocated for the next invocation.
```

```
*
*  It's typically invoked thru a transfer file containing:
*
*           :SV,1,,IH       ** don't show anything unless errors
*           :RP,CLR         ** give it an ID segment (optional)
*           :SYRU,CLR       ** have RTE run it so FMGR is free to
*           :EX,SP          **      execute the next line to log off
*           ::
*
*  Call this transfer file EX so you can enter just ':EX' to log off.
*
*  Use the Large or Extended Background loading option (OP,LB or OP,EB).
*
*  If this program will be run many times a day, do an 'RP,CLR' from FMGR
*  or the WELCOM file after reboot (unless ID segments are critical).
*  If you RP it, be sure to:
*
*     1.  Compile it WITHOUT the 'd' option
*     2.  Remove the RP line from the transfer file
*

      IMPLICIT INTEGER (a-z)

      DIMENSION cntlWd(2), cursUp(23)              ! for CRT I/O control
      DIMENSION Hdown(2), scroll(21)
d     DIMENSION mss(14)                            ! MESSS message to off ID
d     DATA mss /'OF,CLR  ,8                  '/    ! ID segment clean-up

      DATA scroll /21 * 6412B/     ! 21 CRLFs to scroll the chars off screen
      DATA Hdown /15555B, 15506B/  ! mem lock off (esc m) & home down (esc F)
      DATA cursUp /23 * 15501B/    ! 23 cursor ups (esc A)
      !    EXEC call parameters to delay execution:
      DATA delay /12/,  self /0/,  secs /2/,  once /0/,  write /2/


      junk = LOGLU (sysLU)         ! get system LU of invoking terminal
      cntlWd(1) = 100000B + sysLU  ! set bit 16 for SYSTEM LU number
      cntlWd(2) = 0                ! no special driver instructions

      CALL DTACH (junk)            ! now detach from session so that
                                   !   LGOFF won't stop the :EX,SP
      CALL EXEC (delay, self, secs, once, -5)   ! sleep for 5 seconds

* the rest of the code executes once, 5 seconds later

      CALL XLUEX (write, cntlWd, Hdown, 2)   ! put cursor at screen bottom
      CALL XLUEX (write, cntlWd, scroll, 21) ! scroll info off screen
      CALL XLUEX (write, cntlWd, cursUp, 23) ! put cursor at top of screen

d     junk = MESSS (mss, 10)                 ! release the ID segment
      END
```

FAST FORTRAN — AN UPDATE

by John Pezzano/HP El Paso, Texas

In the Vol. 4 Issue 4 of the Communicator I wrote an article entitled "FAST FORTRAN" which contained rules and examples for writing efficient FORTRAN programs.

With the introduction of the FORTRAN 4X compiler, some of these rules and examples need to be modified. FORTRAN 4X (FTN4X) is much more efficient than FTN4 generating different code. So here are my modified rules:

1. KNOW WHAT IT TAKES TO COMPILE AN EQUATION FOR MINIMUM COMPILATION

The example shown :

```
X=Z*(Z+1.0)
```

was more efficient as well as shorter than

```
X=Z**2+Z
```

in FTN4. The FTN4X compiler generates the same efficient code for both versions, converting the equations to

```
X=Z*Z+Z
```

The rule is still valid, but the example is not.

2. REMOVE INVARIANT FROM A DO LOOP

This rule is still valid as is the example. However, the gain is not as great. By removing the part of the loop that does not change within the loop itself, considerable speed can still be gained. To illustrate the difference in the compilers,

ORIGINAL CODE

```
      DO 10 I=1,100
      Z(I)=SIN(X**2+Y**2)+Z(I)*2
10    CONTINUE
```

INVARIENT REMOVED

```
R=SIN(X**2+Y**2)
DO 10 I=1,100
Z(I)=R+Z(I)*2
10  CONTINUE
```

	FTN4	FTN4X	FTN4	FTN4X
loop size	38	24	18	17
savings of instructions in 100X loop	- - -	1400	2000	2100
SIN calls	100	100	1	1
.RTOR calls	200	0	0	0

3. MAKE SURE CONSTANTS AGREE IN TYPE WITH VARIABLES

Neither the rule nor the example are required in FTN4X although this is still a good programming technique. FTN4X would automatically convert the example

```
X=Y+1
```

to the better

```
X=Y+1.0
```

4. COMBINE CONSTANTS

The rule and example are both valid. Use

```
DATA POVER2/1.570796/
X=POVER2 + Y
```

for

```
DATA PI/3.14159/
X=PI/2.0 + Y
```

5. DON'T CALL LIBRARY ROUTINES UNNECESSARILY

No change. The rule is still valid as are the examples.

6. NOW HOW TO USE ARRAYS

The rule is valid but the examples are not. Unlike FTN4, FTN4X calculates fixed array elements at compilation, rather than at run time. Therefore,

A. The example

```
Y=X(1)**2+X(2)**2+X(3)**2
```

in FTN4X takes no more instructions than

```
EQUIVALENCE (X(1),X1),(X(2),X2),(X(3),X3)
Y=X1**2 + X2**2 + X3**2
```

whereas it took 15 more words in FTN4.

B. The example

```
      DIMENSION A(20,20)
      DO 10 I=1,20
      DO 10 J=1,20
10    A(I,J)=0.0
```

took 4400 more executing instructions and was 15 words longer than

```
      DIMENSION A(20,20),B(400)
      EQUIVALENCE (A(1,1),B(1))
      DO 10 I=1,400
10    B(I)=0.0
```

In FTN4X, the numbers are 1320 and 6 respectively. While it is still a considerable improvement, the numbers are not as dramatic.

C. The example

```
      X(I)=A(I)/B(I)
      Y(I)=A(I)/C+W
      Z(I)=R/2.0+A(I)
```

costs 6 words/loop in FTN4X vs. 19 words/loop in FTN4 over

```
      AI=A(I)
      X(I)=AI/B(I)
      Y(I)=AI/C+W
      Z(I)=R/2.+AI
```

which may not be worth the effort.

D. Knowing what the use of the array name without subscript means is still important but not in this example. In FTN4X,

```
      ARRAY=X
```

and

```
      ARRAY(1)=X
```

generate the same code.

7. AVOID FORMATTER LIKE THE PLAGUE!

This is still true since the FORMATTER is much slower and takes more space than EXEC and REIO calls. However, the flexibility in getting status, jumping on errors, and checking EOF's now exist in FTN4X formatted I/O calls. This, combined with the transportability of FORMATTED I/O must be weighed against the speed/size cost.

CONCLUSION

FTN4X can offer dramatic improvement both in code size generated as well as execution efficiency for a typical FORTRAN program. As a person who started on FORTRAN II in pre-RTE environment and who worked with the original pre-release (1974) FTN4 compiler, I can heartily recommend FTN4X. However, one can still "beat" even the best compilers by good coding.

What about FTN7X on RTE-6VM? While I have not completely tested all the above on FTN7X, preliminary testing has shown it to be similar to FTN4X but with some slight improvement in the number of loop instructions (2 fewer) when doing DO LOOPS in FORTRAN 77 mode (vs FORTRAN 66 mode).

ACCESSING PHYSICAL MEMORY

by Stephen Botzko/Hewlett Packard
Waltham Division

In a recent Communicator/1000 article ("Accessing Physical Memory in FORTRAN and PASCAL", Larry W. Smith, Vol. V, Issue 1), a technique for accessing physical memory was presented. We have also found that direct user management of physical memory is an efficient method for sharing large data structures among several user programs. The technique presented in the article has the disadvantage that the mapping mechanism restricts the maximum size of the application programs.

The following subroutines (MXGET,MXPUT) overcome this drawback. They allow the user to transfer up to 1024 consecutive words, starting from any location in physical memory.

The routines maintain maximum logical address space by mapping the physical memory through the user's driver partition map registers. (All types of user programs have two map registers reserved for driver partitions). Protection from RTE interactions is provided by using $LIBR to turn off nonpriviledged interrupts during the transfer, and by restoring the user map to its original state after the transfer completes (but before the call to $LIBX!).

The use of $LIBR and $LIBX make MXGET and MXPUT somewhat slower than the MEMGT routine provided by Larry Smith. (They execute at about 400 microseconds for a one-word transfer). However, the maintenance of the user's full logical address space will more than offset this slower running time in many applications.

```
ASMB,R,L
      NAM MXGET,7
*
*
*
*
*
*
*
*
*
*
*
*----------------------------------------------------------------------*
*
*
*     NAME: MXGET
*     FUNCTION: TRANSFER WORDS FROM/TO PHYSICAL MEMORY
*
*     CALLING SEQUENCE:
*
*           CALL MXGET(PAGE,ADDR,BUFF,NWORD)
*           CALL MXPUT(PAGE,ADDR,BUFF,NWORD)
*
*     INPUTS:
*
*           PAGE      PAGE IN PHYSICAL MEMORY
*           ADDR      PAGE OFFSET
*           BUFF      PROGRAM BUFFER(INPUT FOR MXPUT)
*           NWORD     NUMBER OF WORDS TO TRANSFER
*
```

```
*     OUTPUTS:
*
*           BUFF      USER BUFFER INTO WHICH DATA TRANSFERRED(MXGET)
*           PAGE,ADDR PHYSICAL MEMORY                        (MXPUT)
*
*     ERROR CONDITIONS:
*
*           NONE
*
*     PROCESS:
*
*           GO PRIVILEGED
*           ADJUST USER MAP (USE DRIVER PARTITION)
*
*           TRANSFER WORDS FROM TO USER BUFFER  (MXGET)
*                           (OR VICE VERSA FOR MXPUT)
*
*           RESTORE USER MAP
*           RESTORE INTERRUPTS
*
*-------------------------------------------------------------------*
*
      ENT MXGET,MXPUT
      EXT .ENTR,$LIBR,$LIBX,$DVPT
*
*-------------------------------------------------------------------*
*
PAGE  NOP
ADDR  NOP
BUFF  NOP
NWORD NOP
MXGET NOP
      JSB .ENTR
      DEF PAGE
      XLA $DVPT
      MPY =B2000        * CONVERT DRIVER PARTITION PAGE INTO LOGICAL ADDR
      STA BASE
      LDA ADDR,I
      STA TEMP
      AND =B1777
      ADA BASE
      STA ADDR
*
*
*
* COMMON CODE FOR MXGET,MXPUT STARTS HERE
*
MX.0  LDA TEMP
      CLB
      LSR 10
      STA TEMP           * TEMP HOLDS PAGE OFFSET OF ADDRESS
      XLA $DVPT
      ADA =D32
      STA DPBAS
*
* NOW FIND PAGE TO MAP INTO DRIVER PART'N (PERMITS PAGE OFFSET > 1023)
*
```

```
      LDA PAGE
      ADA TEMP
      AND =B1777
      STA PAGE
*
* SAVE OFF DRIVER PARTITION PAGES
*
      JSB $LIBR
      NOP
      LDX =D-2
      LDA DPBAS
      LDB SAVE
      XMM
*
* NOW REMAP INTO DRIVER PARTITION AREA
*
      LDX =D1
      LDA DPBAS
      LDB PAGE
      XMS
      LDX =D1
      LDA DPBAS
      INA
      LDB PAGE
      INB
      XMS
*
* NOW MOVE THE DATA
*
      LDA ADDR
      LDB BUFF
      MVW NWORD,I
*
* RESTORE USER MAP
*
      LDX =D2
      LDA DPBAS
      LDB SAVE
      XMM
*
* TURN OFF INTERRUPTS
*
      JSB $LIBX
      DEF *+1
      DEF *+1
      JMP MXGET,I
```


```
*
* ENTRY POINT FOR MXPUT
*
PAGE1 NOP
ADDR1 NOP
BUFF1 NOP
NWRD1 NOP
MXPUT NOP
      JSB .ENTR
      DEF PAGE1
      LDA MXPUT
      STA MXGET
      XLA $DVPT
      MPY =B2000
      STA BASE
*
      LDA ADDR1,I
      STA TEMP
      AND =B1777
      ADA BASE
      STA BUFF
*
      LDA BUFF1
      STA ADDR
*
      LDA NWRD1
      STA NWORD
*
      LDA PAGE1
      STA PAGE
*
      JMP MX.0
*
*
*
BASE  BSS 1
DPBAS BSS 1
TMPTR DEF TEMP
TEMP  DEC 0
TEMP2 DEC 0
SAVE  DEF SAVIT
SAVIT BSS 2          * TWO WORD SAVE AREA
      END
```

MORE NOTES ON THE USE OF UNDECLARED MEMORY

by Jeff Wynne/Naval Ordnance Station,
Indian Head, MD

Readers are referred to the article on this subject by John Pezzano which appeared in the Communicator, Volume 5, Issue 2 (1981). Suggested uses for undeclared memory and linking techniques are discussed in some detail.

EDITOR'S NOTE: see also letter from Ernst Stelzer in this issue.

REDUCED DISK STORAGE REQUIREMENTS FOR TYPE-6 FILES

Arrays in undeclared memory (UDCM) are not included in the object image of the program on disk. This is true with RTE-IVB for both permanently loaded programs and for type-6 object code program files. Just consider the following code:

```
      PROGRAM EXMPL
      COMMON IA(20000)
      DO 10 I = 1, 20000
10    IA(I) = 1
      END
```

The above program requires very little disk storage in an RTE-2 system since temporary COMMON is not included in the object program space. However, RTE-IVB does put temporary COMMON in the program space and the RTE-IVB type-6 file for this program requires 159 blocks of disk storage. At our site, where we started with an HP-2100 and a single 5 MByte disk, this waste is unacceptable (even though we now have 80 MBytes of disk storage on some systems). The same program, with the array IA in undeclared memory requires 4 blocks for its RTE-IVB type-6 file.

ALL-FORTRAN LINKING TECHNIQUES

In addition to the assembly link described in Mr. Pezzano's article, there are several ways to get around the problem of passing the address of undeclared memory as an array reference in a FORTRAN program. If desirable, it can be done with "all-FORTRAN" code.

The problem was:

```
PROGRAM MAIN
CALL LIMEN(0, IBUF, LEN)
CALL SUB(IBUF, LEN)
etc.
SUBROUTINE SUB(IBUF, LEN)
INTEGER IBUF(1)
etc.
```

Of course this won't work. As pointed out by Mr. Pezzano, the problem is that the address passed in the call to SUB is an address where the address of undeclared memory is stored rather than the address itself.

Consider the following modifications to the above code.

```
PROGRAM MAIN
DIMENSION IA(1)
CALL UDC (IA, IEL1, LEN)
CALL SUB (IA(IEL1), LEN)
etc.
SUBROUTINE SUB(IBUF, LEN)
INTEGER IBUF(LEN)
etc.
SUBROUTINE UDC(IA, IEL1, LEN)
IADD = IGETA(IA)
CALL LIMEM (0, IFWAM, LEN)
IEL1 = IFWAM - IADD + 1
CALL EXEC(22,3)
END
```

Here the subroutine UDC(IA,IEL1,LEN) is used to find the index of the first element of IA that is in UDCM and the number of words in UDCM. The CALL EXEC(22,3) is for systems like RTE-2 which do not swap the entire partition unless this call is made. It is not required for RTE-IVB (i.e. it has no apparent effect.).

The key to the solution is function subprogram IGETA which may be coded in either FORTRAN or ASSEMBLY. IGETA(IARG) returns the direct address of IARG in the A-register. In the above code, IADD = IGETA(IA) stores the address of IA in variable IADD. IADD and IFWAM are then used to calculate the index of IA which corresponds to the first word of UDCM. Of course IGETA may be used to obtain other addresses also; see subroutine SUB in the example program at the end of this article.

FORTRAN VERSION OF IGETA:

```
      FUNCTION IGETA(IA), RETURN ADDRESS OF ARGUMENT IN A-REGISTER
      ASSIGN 100 TO N
      N = N + 2
100   K = IDMY(IA)
      IGETA = IGET(N)
110   IF(IGETA.GE.0) RETURN
      IGETA = IGET( IGETA.AND.077777B )
      GO TO 110
      END
      FUNCTION IDMY(IA), DUMMY FUNCTION SUBPROGRAM
      IDMY = 0
      END
```

ASSEMBLY VERSION OF FUNCTION IGETA:

```
      NAM IGETA,7 ROUTINE RETURNS ADDRESS OF ARGUMENT 810929.0842
      ENT IGETA
IGETA NOP
      LDA IGETA,I   SAVE RETURN ADDRESS
      STA RETRN
      ISZ IGETA     ADVANCE IGETA TO POINT TO ADDRESS OF ARGUMENT.
      LDA IGETA,I   GET ADDRESS OF THE ARGUMENT.
TEST  SSA,RSS       IS IT INDIRECT ?
      JMP DONE        NO, IT'S DIRECT. ALL DONE.
      AND MASK        YES, MASK BIT 15 AND TRY AGAIN.
      LDA 0,I       GET NEW ADDRESS.
      JMP TEST      GO BACK AND TES AGAIN.
DONE  JMP RETRN,I
RETRN NOP
MASK  OCT 077777
      END
```

Note that the FORTRAN version is dependent on the HP-1000 linking convention and on the compiler implementation of the ASSIGN statement. At this point, HP is committed to supporting the linking convention. Implementation of the ASSIGN statement is fairly standard and one would not expect that to change with future compilers. This technique works with FTN4 REV 1442 part# 24177-60001, RTE-FTN4 and FTN4X. The sample program which follows has been tested with RTE-FTN4 and with FTN4X.

At this installation, we generate the assembly version of IGETA into the system disk resident library. However, there are a number of reasons for wanting to stay "all-FORTRAN", not the least of which is the rather awkward requirements for compiling, assembling and loading mixed FORTRAN and ASSEMBLY code.

The following program uses the techniques described above. It is "all-FORTRAN". To run it, just compile, load and go. In RTE-IVB, if you want more UDCM than that between the end of code and the end of the current page, use the system SZ command to increase the program space after loading the program (this is the usual thing to do). With RTE-2, you automatically get the entire background or real-time partition.

EXAMPLE FORTRAN PROGRAM

```
FTN4
      PROGRAM ICOR (3,99), DEMO ON USE OF UNDECLARED CORE 820203.1030
C
C  The main program and subroutine UDC are used to find the number of
C  elements and the starting element of array IA which are available
C  for use in undeclared memory.  The main body of the program is in
C  subroutine SUB.
C
      DIMENSION IA(1)
C
      CALL UDC(IA, IEL1, NELS)
C
C   Now call SUB and pass along the first array address in undeclared
C   memory and the number of words available.
C
      CALL SUB(IA(IEL1), NELS)
      END
      SUBROUTINE SUB (ARRAY, N)
      INTEGER ARRAY(N),  TLU
```

```
C
      TLU = LOGLU(IDMMY)
      IF(TLU.LT.1) TLU = 1
      DO 100 I = 1, N
      IF(I.GT.10.AND.I.LT.N-10) GO TO 100
      K = IGETA(ARRAY(I))
      WRITE(TLU,88) I, K, ARRAY(I)
88    FORMAT(" ELEMENT #" I6 "   ADDRESS=" K8 "    VALUE =" K8)
100   CONTINUE
      RETURN
      END
      SUBROUTINE UDC(ARRAY, ELEM1, NELMS), FIND 1ST AND NUMBER IN UDC
      IMPLICIT INTEGER (A-Z)
C
C   This subroutine calculates the element (ELEM1) of integer array
C   ARRAY that corresponds to the first word of available memory at
C   memory address FWAM.  It also calculates NELMS the number of
C   words from FWAM to the end of the partition.
C
C   Note that NELMS could be zero if the program had no undeclared
C   memory.  This technique employs subroutine  IGETA to get the
C   address of the array ARRAY.
C
C   An EXEC(22,3) call is made so that the entire partition is
C   swapped (required for RTE-2, etc.).
C
      IADD = IGETA(ARRAY)
C
120   IWW = 0
      CALL LIMEM(IWW, FWAM, NELMS)
      ELEM1 = FWAM - IADD + 1
C
C   Set swapping to get entire partition.
C
      CALL EXEC(22,3)
C
      RETURN
      END
      FUNCTION IGETA(IA), RETURN ADDRESS OF ARGUMENT IN A-REGISTER
      ASSIGN 100 TO N
      N = N + 2
100   CALL IDMY(IA)
      IGETA = IGET(N)
110   IF(IGETA.GE.0) RETURN
      IGETA = IGET( IGETA.AND.077777B )
      GO TO 110
      END
      SUBROUTINE IDMY(IA)
      CONTINUE
      END
```

SHORT FORMATTED IO FOR LUS IN PASCAL/1000

by Dave Redmond/HP Albuquerque, NM

When new users are introduced to Pascal/1000 they are usually impressed with the ability to specify either files or LUs for IO, especially at run time. This is particularly useful for determining how a developing program outputs to a file; merely direct that output to your terminal instead. The next impression new users get is typically that IO programs seem to be quite large. For example, the following program is nine pages long when loaded:

```
PROGRAM S ( OUTPUT );
  BEGIN
    WRITELN ( 'Hello!!')
  END.
```

A look at the loader listing reveals that many of the relocated modules create, open, and generally manipulate files. All these modules allow the user the file/LU versatility with formatted IO, but also add to the size of his programs, proving the old adage "You don't get something for nothing!"

Assume, though, that a particular application requires IO to/from LUs only. In this case the added modules buy nothing, yet still cost space. The use of Exec reads and writes are fine for ASCII data, but inappropriate for numbers. The need for procedures specialized for formatted IO to/from LUs is obvious.

This article presents one attempt to satisfy this need in a fashion similar to the way Pascal/1000 does formatted IO. The presented procedures are meant to be used instead of the standard Pascal/1000 files (e.g. READs, WRITEs, INPUT, OUTPUT), and, should allow the programmer most of the primitives required for formatted IO. This is a first step, one that is easily modified by the reader.

FORMATTED OUTPUT

The following is a table of calling sequences for the relevant output procedures and their Pascal/1000 equivalents:

CALL	EQUIVALENT
`WrtA (C)`	`WRITE (C)`
`WrtB (Buf, Length)`	`FOR I := 1 TO Length DO WRITE (Buf[I])`
`WrtI (I, N)`	`WRITE (I:N)`
`WrtR (R, N, M)`	`WRITE (R:N:M)`
`WrtLn`	`WRITELN`

For all of the above procedures the output is to the LU Out_Lu. This LU must be set by the programmer. With these primitives one can output data in any of the generally required formats. (Note: Neither E nor L formats are used.)

These procedures work much the same as the Pascal/1000 procedures. There is an output buffer, Out_Buf, into which the appropriate characters are appended by all of the output procedures except WrtLn. An index, Out_Index, indicates the position in Out_Buf where the next character should go: Out_Buf[Out_Index]. WrtLn does an Exec write to Out_LU and resets Out_Index to 1.

WrtA and WrtB are fairly obvious. WrtI and WrtR, however, are a bit more complex. In an effort to simplify things, an additional procedure, WrtI0, was written to determine the appropriate characters for the represention of a signed integer in a given length, with or without leading zeros. WrtI calls WrtI0 without leading zeros. WrtR calls WrtI0 without leading zeros for the digits to the left of the decimal point, and with leading zeros for those to the right. If the ASCII representation of the number will not fit in the desired length (due to the absolute value being too large or there being no room for a negative sign) the first character will be an asterisk.

FORMATTED INPUT

The following is a table of the calling sequences for the relevant input procedures and their Pascal/1000 equivalents:

```
CALL                    EQUIVALENT
RdA (C)                 READ (C)
RdB (Buf, Length)       FOR I := 1 TO Length DO READ (Buf[I])
RdI (I)                 Skip_to_Digit;  READ (I)
RdR (R)                 Skip_to_Digit;  READ (R)
RdLn
```

where Skip_to_Digit is equivalent to

```
WHILE NOT (INPUT^ IN ['0'..'9', '+', '-']) DO READ (C)
```

Again for the above procedures, the input is from the LU In_Lu, set up by the programmer. (This sure seems familiar.) With these primitives one can input data in any of the general formats (once again, neither E nor L formats are used).

There is an input buffer, In_Buf, from which the appropriate characters are read by all of the preceding procedures except RdLn. An index, In_Index, indicates the last character of In_Buf that was read. That last character read is in the global Ch, and the next character to be read is in the global Next_Ch (the equivalent to INPUT↑). Another global, the BOOLEAN Eoln, is set TRUE by RdLn and when the last character input is read. In both cases the next read will, at least temporarily, set Eoln FALSE.

RdA and RdB are quite easily implemented. RdI and RdR, however, require an additional function, R_Int, which returns the LONGREAL value converted from the next string of integer characters ('0'..'9'). (Note that R_Int returns a LONGREAL value. This is so that both RdI and RdR can use it. RdI will correctly convert integers over 7 places long only with LONGREAL.) RdI and RdR function quite differently from their Pascal/1000 logical equivalents READ (I) and READ (R), as indicated in the preceding table. RdI and RdR actually skip characters from the input stream until a number or a sign (+ or −) is encountered. Only then does the conversion process begin. This allows more freedom upon input. For instance, RdR (X) will be satisfied by the following input string:

```
X should get
      the value        -6.42
```

The advantages are obvious, so are some disadvantages (error checking, etc.).

SUGGESTED USAGE

It is recommended that these procedures (with the reader's modifications, of course) be compiled as a separate SUBPROGRAM so that the resulting relocatable can be searched during loading. This way only the procedures required will be loaded, thus saving some space. An inclusion file is also recommended (here called &IOINC) for the declarations. All of your own declarations should be included, as well as all of the following:

```
TYPE
  Char_Set = SET OF CHAR;

CONST
  User_Buffer_Length   =  nnn;   (* Typically 80                *)
    IO_Buffer_Length   =  mmm;   (* Typically the same as above *)

  Numeric_Set = Char_Set ['0'..'9', '+', '-'];
  Digit_Set   = Char_Set ['0'..'9'];
```

```
TYPE
  Int          = -32768..32767;   (* One word integers           *)

  User_Buffer = PACKED ARRAY [1..User_Buffer_Length] OF CHAR;
    IO_Buffer = PACKED ARRAY [1..  IO_Buffer_Length] OF CHAR;

VAR
  Out_Buf,                        (* The output buffer           *)
   In_Buf                         (* The  input buffer           *)
    : IO_Buffer;

  Ch,                             (* The last char read          *)
  Next_Ch                         (* The next char to be read    *)
    : CHAR;

  Eoln
    : BOOLEAN;

  Out_LU,                         (* The output LU               *)
   In_LU,                         (* The  input LU               *)

  Out_Index,                      (* The next position to write  *)
   In_Index                       (* The last position read      *)
    : Int;
```

To initialize the output, the programmer must set Out__LU to the appropriate output LU and set Out__Index to one. To initialize the input, In__LU should be set appropriately and a call should be made to RdLn.

The actual code should answer many specific questions. Immediately following the code is an example of its use.

```
$SUBPROGRAM,RECURSIVE OFF,RANGE OFF,HEAP 0,AUTOPAGE ON,VISIBLE ON$
(*
      Note : Any time size is a factor, the programmer should
               avoid the added expense of RECURSIVE ON,
               RANGE ON, and HEAP n with n <> 0.
               See the Pascal/1000 Ref Man for more info.
*)
PROGRAM IO ;
(*
      This subprogram contains the following visible procedures:

        WrtA   to write a single character
        WrtB   to write a string of characters
        WrtI   to write a signed integer
        WrtR   to write a signed real number
        Wrtln  the WRITELN equivalent

        RdA    to read a single character
        RdB    to read a string of characters
        RdI    to read a signed integer
        RdR    to read a signed real number
        RdLn   the READLN equivalent
*)
```


```
  CONST
    Asterisk      = '*';
    Blank         = ' ';
    Zero          = '0';
    Minus         = '-';
    Plus          = '+';
    Decimal_Point = '.';

$INCLUDE '&IOINC'$


  PROCEDURE Reent_IO $ALIAS 'REIO'$
              (     ICode,
                    ICnwd  : Int;
               VAR  Buf    : IO_Buffer;
                    Length : Int

              );  EXTERNAL;


  PROCEDURE Get_Xmit_Len $ALIAS 'ABREG'$
              (VAR A_Reg,
                    Length : Int

              );  EXTERNAL;
1     $PAGE$
(*
############################## WrtLn ###################################
*)
  PROCEDURE Wrtln;
(*
      Equivalent to WRITELN
*)
    BEGIN
      Reent_IO (2, Out_LU, Out_Buf, 1 - Out_Index);

      Out_Index := 1;
    END;

(*
############################## WrtA ####################################
*)
  PROCEDURE WrtA (C : Char);
(*
      Equivalent to WRITE (C);
*)
    BEGIN  (*  Proc WrtA  *)
      Out_Buf[Out_Index] := C;
(*
          Check output line length
*)
      IF Out_Index = IO_Buffer_Length THEN Wrtln
      ELSE Out_Index := Out_Index + 1;
    END;

(*
############################## WrtB ####################################
*)
  PROCEDURE WrtB (Buf : User_Buffer;  Length : Int);
```

```
(*
      Equivalent to FOR I := 1 TO Length DO WRITE (Buf[I])
*)
    VAR
      I : Int;

    BEGIN  (*  Proc WrtB  *)
      FOR I := 1 to Length DO WrtA(Buf[I]);
    END;  (*  Proc WrtB  *)
1
(*
############################## WrtI0 ###################################
*)
$VISIBLE OFF$

  PROCEDURE WrtI0 (N : INTEGER;  Length : Int;  Leading_Zeros : BOOLEAN);
(*
      This procedure is used by the other output procedures in this
        subprogram and is not designed for general user use.
      This procedure has an effect equivalent to WRITE(N:Length)
        except that if the integer requires more space than Length
        the first char will be an asterisk.
        Also, if Leading_Zeros is TRUE then leading zeros will be
        included (Note: This works only for positive N; remember that
        this is not for general use).
*)
    VAR
      Index    : Int;
      Negative : BOOLEAN;
      Buf      : PACKED ARRAY [1..20] OF CHAR;

    BEGIN  (*  Proc WrtI0  *)
      Buf := Blank;

      Negative :=   N < 0;
      N := ABS(N);
      Index := Length;
(*
          Put ASCII representation of N in Buf
*)
      WHILE (N > 0) AND (Index > 0) DO
        BEGIN
          Buf[Index] := CHR( ORD(Zero) +  N MOD 10);
          Index := Index - 1;
          N := N DIV 10;
        END;
(*
          Check for overflow, sign and leading zeros
*)
      IF (N > 0) OR ((Index = 0) AND Negative) THEN Buf[1] := Asterisk

      ELSE IF Negative THEN Buf[Index] := Minus

      ELSE IF Leading_Zeros THEN

             FOR Index := Index DOWNTO 1 DO Buf[Index] := Zero;
(*
          Output the buffer
```

```
*)
      Wrtb (Buf, Length);

    END;  (*  Proc WrtIO  *)
1
$VISIBLE ON$
(*
############################## WrtR ###################################
*)

  PROCEDURE WrtR (R : REAL;  Width, Right : Int);
(*
      Equivalent to WRITE(R:Width:Right)
*)
    VAR
      I : Int;

    BEGIN
      IF Right > 8 THEN Right := 8;     (* Not required *)
(*
          Wrt digits to left of decimal point without leading zeros
*)
      WrtIO (TRUNC(R), Width - Right - 1, FALSE);

      WrtA (Decimal_Point);
(*
          Wrt digits to right of decimal point with leading zeros
*)
      R := ABS(R - TRUNC(R));

      FOR I := 1 TO Right DO R := 10.0 * R;

      WrtIO (ROUND(R), Right, TRUE);

    END;  (*  Proc WrtR  *)

(*
############################## WrtI ###################################
*)
  PROCEDURE WrtI (N : INTEGER; Length : Int);
(*
      Equivalent to WRITE (N:Length)
*)
    BEGIN  (*  Proc WrtI  *)

      WrtIO (N, Length, FALSE);

    END;  (*  Proc WrtI  *)

1     $PAGE$
(*
############################## Get_Ch ###################################
*)
  PROCEDURE Get_Ch;
(*
      This procedure "reads" the next character from the input stream.
        That character is put into the global Ch, and the next character
        to be "read" is put into the global Next_Ch.
        Eoln is checked before and set after.
```

```
*)
    CONST
      Echo = 256;

    VAR
      A_Reg,
      Chars_Read : Int;

    BEGIN  (*  Proc Get_Ch within IO  *)

      IF Eoln THEN
        BEGIN
          Reent_IO (1, In_LU + Echo, In_Buf, -User_Buffer_Length);

          Get_Xmit_Len (A_Reg, Chars_Read);

          In_Index := 0;

          Ch := Blank;  Next_Ch := In_Buf[1];     (* As in Pascal/1000 *)

        END

      ELSE BEGIN
             In_Index := In_Index + 1;

             Ch := In_Buf[In_Index];  Next_Ch := In_Buf[In_Index + 1];
           END;

      Eoln := (In_Index >= Chars_Read);

    END;  (*  Proc Get_Ch  *)

(*
############################## RdA ######################################
*)
  PROCEDURE RdA (VAR C : CHAR);
(*
          Equivalent to READ (C);
*)
    BEGIN  (*  Proc RdA within IO  *)

      Get_Ch;

      C := Ch;

    END;  (*  Proc RdA  *)
1
(*
############################## RdB ######################################
*)
  PROCEDURE RdB (VAR Buf : User_Buffer;   Length : Int);
(*
          Equivalent to READ (Buf), but, terminate read after
            either Length chars being read, or Eoln TRUE.
```

```
*)
    VAR
      N : Int;

    BEGIN  (*  Proc RdB within IO  *)

      N := 1;
      Buf := Blank;                    (* Blank fill to start *)

      REPEAT
        Get_Ch;

        Buf[N] := Ch;

        N := N + 1;
      UNTIL (N > Length) OR Eoln;

    END;  (*  Proc RdB  *)
1
(*
############################## R_Int ###################################
*)
$VISIBLE OFF$
  FUNCTION R_Int : LONGREAL;

    VAR
      R    : LONGREAL;

      Sign : CHAR;

    BEGIN  (*  Func R_Int within IO  *)

      R := 0.0;    Get_Ch;
(*
          Skip to digit or sign
*)
      WHILE NOT (Next_Ch IN Numeric_Set) DO Get_Ch;
(*
          If it is a sign, read and save it.
*)
      IF (Next_Ch = Plus) OR (Next_Ch = Minus) THEN
        BEGIN
          Get_Ch;


          Sign := Ch;

          WHILE Next_Ch = Blank DO Get_Ch;
        END;
(*
          Convert to REAL
*)
      WHILE Next_Ch IN Digit_Set DO
        BEGIN
          Get_Ch;

          R := 10.0 * R  +  1.0 * (ORD(Ch) - ORD(Zero));
        END;
```

```
      IF Sign = Minus THEN R := -R;

      R_Int := R;

    END;  (*  Func R_Int  *)
$VISIBLE ON$

(*
############################## RdI ######################################
*)
  PROCEDURE RdI (VAR I : INTEGER);
(*
          Equivalent to  Skip_to_Digit;  READ (I)
*)
    BEGIN  (*  Proc RdI within IO  *)

      I := ROUND( R_Int );

    END;  (*  Proc RdI  *)
1
(*
############################## RdR ######################################
*)
  PROCEDURE RdR (VAR R : REAL);
(*
          Equivalent to Skip_to_Digit;  READ (R)
*)
    VAR
      Divisor : REAL;

    BEGIN  (*  Proc RdR within IO  *)

      R := R_Int;        (* Get the integer portion *)

      IF Next_Ch = Decimal_Point THEN
        BEGIN
          Get_Ch;

          IF R < 0.0 THEN Divisor := -10.0  ELSE Divisor := 10.0;
(*
              Now add the fractional portion
*)
          WHILE Next_Ch IN Digit_Set DO
            BEGIN
              Get_Ch;

              R := R + (ORD(Ch) - ORD(Zero)) / Divisor;

              Divisor := Divisor * 10.0;
            END;
        END;

    END;  (*  Proc RdR  *)
```

```
(*
############################## RdLn ###################################
*)
  PROCEDURE RdLn;
(*
          Equivalent to READLN
*)
    BEGIN  (*  Proc RdLn within IO  *)

      Eoln := TRUE;    (* Get_Ch does the work on next call *)

    END;  (*  Proc RdLn  *)


     .  (*  END SUBPROGRAM IO  *)
```

An example is worth 1024 words. The following program shows a typical use of the procedures. It loads in 3 pages.

```
$RECURSIVE OFF, HEAP 0, RANGE OFF$

PROGRAM T;
(*
      This program is an example of the use of the presented IO
        scheme.

      The user is prompted for two real numbers.  The sum and
        difference of the two numbers is output.  This process
        is repeated until the first number is zero.
*)

$INCLUDE '&IOINC'$  (* Includes all suggested and REAL A and B *)

  PROCEDURE WrtB (Buf : IO_Buffer;  Length : Int);  EXTERNAL;

  PROCEDURE WrtR (R : REAL;  Left, Right : Int);  EXTERNAL;

  PROCEDURE Wrtln;  EXTERNAL;


  PROCEDURE RdR (R : REAL);  EXTERNAL;

  PROCEDURE RdLn;  EXTERNAL;

  BEGIN
    In_LU := 1;   Out_LU := 1;       (* Initialize the IO   *)

    RdLn;         Out_Index := 1;


    REPEAT                           (* Request and display *)

      WrtB (' Please enter 2 numbers: _', 26);   Wrtln;

      RdR (A);   RdR (B);

      WrtR (A);  WrtR (B);  WrtLn;

      WrtB (' The sum is ', 12);  WrtR (A+B, 15, 6);  Wrtln;
```

```
    WrtB (' The difference is ', 19);  WrtR (A-B, 15, 6);  Wrtln;

    Wrtln;

  UNTIL A = 0.0;

END.
```

After &IO and &T have been compiled, creating %IO and %T, the loader is run with the following commands:

```
LI,%IO
REL,%T
EN
```

A sample run follows:

```
:RU,T
 Please enter 2 numbers: 123.0    456.0
       123.000000        456.000000
 The sum is       579.000000
 The difference is      -333.000000

 Please enter 2 numbers: Let's try -144.6 and 144.6, ok?
      -144.600006       -144.600006
 The sum is      -289.200012
 The difference is          .000000

 Please enter 2 numbers: How about 123456789 and
                           the number 3.14159
-123456784.000000         -3.141590
 The sum is *3456784.000000
 The difference is *3456784.000000

 Please enter 2 numbers: -  50    and  +16
       -50.000000         16.000000
 The sum is       -34.000000
 The difference is       -66.000000

 Please enter 2 numbers: 0.0     ,   456
          .000000        456.000000
 The sum is       456.000000
 The difference is      -456.000000
```

With these procedures and the reader's own modifications, programs can be written which do formatted IO to LUs only resulting in programs which are about six pages smaller than if the standard Pascal/1000 procedures had been used. This size reduction could easily make the difference between running on an L-Series or not. It is hoped that the trade-offs will be adventageous and that these procedures will be put to good use. Happy Pascalling!!

PASCAL ERROR TRAPPING AND REPORTING

by Jeffrey Hirschl/HP, Systems Technology Organization, Fort Collins, CO

A problem facing the PASCAL applications programmer is how to trap FMP (File Management Package) errors which occur when opening or otherwise accessing a file. Normally the Pascal run-time system handles errors by printing an error message and terminating the program. This is fine if the error is not an FMP error, since a programming error is usually the cause. But an FMP "error" in many cases is really a file status indication rather than an error. Where this is the case, aborting the program may not be desirable at all!

Suppose, for example, we have a program that uses a sequential access file to store its control information. If the file doesn't exist, we want the program to create and place some initial values into it. But opening the file with a RESET (to read it) will result in a program abort if the file doesn't exist (FMP error −6). Using the FMP error trapping subroutine described below, the program can trap the error and determine that it occurred. It can then issue a REWRITE to initialize the file.

In another case we may have a program which uses several files. If a file is unavailable because another program has opened it exclusively, we may want to make an orderly cleanup and terminate. We may even want to delay and then try to open it again. Here again, the FMP error trapping subroutine will inhibit Pascal normal error abort action and will inform the program that an error (FMP error -8 in this case) occurred.

The FMP error trapping subroutine maintains a storage location where the error number of the last Pascal FMP error is kept. This location contains zero if no error has occurred. The user's Pascal program obtains this error number by calling subroutine ERTST (see the program listing for calling details). Each time ERTST is called, the FMP error storage is set back to zero so that subsequent calls will send back a zero (no error) result until the next error occurs. ERTST is easy to call, requiring just one external procedure definition and one variable definition. Therefore its impact on the user Pascal program is minimal.

The subroutine allows handling of non-FMP errors by printing a Pascal error message. Then if the error is not a "warning", the program is terminated. Using the provided subroutine as a guide and the information from Appendix B of the Pascal/1000 Reference Manual, the subroutine may easily be modified to trap and report other errors.

The package is designed to work on an "F" series processor and may require some modification to work on others.

A Pascal program showing examples of usage is provided. The program deliberately terminates with an error to demonstrate how non-FMP errors are handled. Note the loading instructions given in this program as well as the subroutine package listing very carefully. The error trapping package must be relocated before the Pascal library or system library are searched. If this is not done, no error will be produced at load time, but the trapping routine will not work.

```
ASMB,R,L,T
*
      NAM PSFMP Pascal FMP Error Handler 811007
      HED Pascal FMP Error Trapping and Reporting
*
*
*     ******************************************
*     * Pascal FMP Error Trapping and Reporting *
*     ******************************************
*
*
*       Author:  Jeffrey S. Hirschl, STO, HP Fort Collins
*
*
*       Designed for "F" Type Processor
*
*
*
* This subroutine package provides a means to trap and report FMP
* errors arising out of Pascal file handling system calls.
*
* If an FMP type error occurs, no error message is printed.  A note
* is made of the error for reporting to the user Pascal program upon
* request.
*
* If a non-FMP-type error occurs, a Pascal error message is printed,
* just as would normally be done.  Then if the error is not a "warning"
* the program is terminated.  If the error is a "warning", control
* returns to the user Pascal program.
*
*
* The package consists of two subroutines:  @PREP, which replaces
* the @PREP subroutine in the Pascal run-time library.  ERTST, which
* is called by the user Pascal program to determine if any FMP errors
* have occurred.
*
*
* Further information on error handling can be found in Appendix B
* of the Pascal/1000 Reference Manual, HP 92832-90001.
*
*
* CAUTION - This routine must be loaded BEFORE the Pascal library is
* searched when running LOADR.  A loading example is:
*
*       RU LOADR
*         RE,%MYPROG          Load user Pascal program
*         RE,%PSFMP           Load error trapping routine.
*         SE                  Search libraries.
*         END
      SKP
*     *********
*     * @PREP *
*     *********
*
*
* This routine replaces the @PREP error handler in the Pascal run-time
* library.  It should never be called by the user Pascal program.
*
```


```
* The calling sequence for this routine may be found in Appendix B
* of the Pascal/1000 Reference Manual.
*
*
      ENT           @PREP
      EXT           @PRER, .ENTR, EXEC
*
*
* Storage for addresses of calling parameters (will be filled in
* by .ENTR).
*
ARTYP BSS           1         "ERR TYPE" PARA ADDR.
ARNUM BSS           1         "ERR NUMBER" PARA ADDR.
ARLIN BSS           1         "ERR LINE" PARA ADDR.
ARFIL BSS           1         "FILENAME" PARA ADDR.
ARLEN BSS           1         "FILENAME LENGTH" PARA ADDR.
*
*
*
@PREP NOP                     SUBROUTINE ENTRY POINT.
*
      JSB           .ENTR     GET PARA ADDRS AND SET
      DEF           ARTYP     ENTRY POINT FOR RETURN.
*
      LDA           ARNUM,I   STORE ERROR NUMBER.
      STA           ERNUM
*
      LDA           ARTYP,I   GET ERROR TYPE.
      CPA           FMPER     IS IT FMP?
      JMP           @PREP,I   YES, RETURN.
*
*
* Here if we don't have an FMP error.  Copy calling parameters
* so that we can call @PRER.  This will print a Pascal error message.
*
      LDA           ARTYA     SET UP ADDRS FOR
      LDB           PRNTA     "MOVE WORDS".
*
      MVW           =D5       MOVE 5 PARAS.
*
      JSB           @PRER     PRINT Pascal
      DEF           *+6       ERR MSG.
*
PRNTE BSS           5         PARAS FOR @PRER CALL.
*
*
* If the error is a warning, return.  Else terminate the program.
*
      LDA           ARTYP,I   GET ERROR TYPE.
      CPA           WARN      WARNING?
      JMP           @PREP,I   YES, RETURN.
*
      JSB           EXEC      NO, TERMINATE.
      DEF           *+2
      DEF           EXIT
*
*
* Constants and local storage.
```

```
*
EXIT  DEC           6         "EXIT" EXEC CODE.
FMPER DEC           3         FMP Pascal ERR CODE.
WARN  DEC           5         WARN Pascal ERR CODE.
ARTYA DEF           ARTYP     ADDR OF ARTYP.
PRNTA DEF           PRNTE     ADDR OF PRNTE.
*
ERNUM DEC           0         ERR NUM STORAGE.
      SKP
*     **********************
*     * Error Test Routine *
*     **********************
*
*
*
* This routine is called by the user Pascal program to determine if
* an FMP error occurred.
*
*
* Pascal calling sequence:
*
*     TYPE
*        Single_Int: -32768..32767;
*
*     PROCEDURE Error_Test $ALIAS 'ERTST'
*          (VAR Ierr: Single_Int); EXTERNAL;
*
*     Error_Test (Ierr);
*
*
* If an FMP error has not occurred, zero will be returned in IERR.
* Otherwise, the FMP error number will be returned in IERR.
*
* When this routine is called, any FMP error is reported and then
* cleared.  Subsequent calls will yield a zero IERR result until
* the next error actually occurs.
*
* If the Pascal program is to be a restartable program, a dummy call
* must be made to this routine at the beginning of the Pascal program
* (after any calls to RMPAR) to initialize the FMP error storage to
* zero.  If this is not done, any error remaining from a previous
* run of the program will yield incorrect results.
*
*
      ENT           ERTST
*
* Storage for IERR parameter address.
*
AIERR BSS           1         "IERR" PARA ADDR.
*
*
ERTST NOP                     ENTRY POINT.
*
      JSB           .ENTR     GET PARA ADDR AND
      DEF           AIERR     SET UP RETURN.
*
      LDA           ERNUM     GET ERROR NUMBER
      STA           AIERR,I   INTO IERR PARA.
```

```
*
      CLA                     CLEAR ERROR STORAGE.
      STA           ERNUM
*
      JMP           ERTST,I   RETURN.
*
      END
```

```
$Pascal 'Pascal FMP Error Example'$
$RECURSIVE OFF$

  PROGRAM TSFMP (OUTPUT);

{This routine provides an example of how the Pascal FMP Error trapping
 and reporting routine is used.}

{The routine tries to open a file ^^XYZ.  If successful, it reads
 the message in the file and prints it.  If not successful and the
 error indicates the file doesn't exist, the routine creates it and
 writes a message into it.  If not successful for any other reason,
 an error message is printed.}

{To demonstrate the action with an error message other than "file
 doesn't exist, after the file is created try holding it open with
 an editor while you run this routine.}

{NOTE: This program demonstrates what happens in the case of a non-FMP
 error by trying to access file Ffile after it is closed.  Thus it
 deliberately exits in error.}


{To load:
             RU,LOADR
              RE,%TSFMP                Relocate this program.
              RE,%PSFMP                Relocate error trapper and reporter.
              END

 To run:  RU,TSFMP,1        }
```

```
     TYPE
        Single_Int = -32768..32767;

     VAR
        Ierr: Single_Int;
        Ffile: TEXT;
        Fbuf: PACKED ARRAY [1..40] of CHAR;

     PROCEDURE Error_Test $ALIAS 'ERTST'$ (VAR Ierr: Single_Int);
                                          EXTERNAL;
```

```
BEGIN

   Error_Test (Ierr);                {This is a dummy call to
                                      Error_Test to ensure program
                                      restartability.}

   RESET (Ffile, '^^XYZ');          {Try to reset file.}
   Error_Test (Ierr);                {Check for FMP error.}

   IF Ierr < 0 THEN
      BEGIN
         WRITELN ('Error opening ^^XYZ, FMP error ', Ierr);
         IF Ierr = -6 THEN           {If FMP error is "nonexistant
                                      file", create the file.}
            BEGIN
               REWRITE (Ffile, '^^XYZ');
               Error_Test (Ierr);  {Test for error during REWRITE}
               IF (Ierr < 0) THEN
                  WRITELN ('Error creating ^^XYZ, FMP error ', Ierr)
               ELSE
                  BEGIN
                     WRITELN ('Successfully created ^^XYZ');
                     WRITELN (Ffile, 'This is a test file')
                  END
            END  {IF Ierr = -6}
      END   {IF Ierr < 0}

   ELSE
      BEGIN
         WRITELN ('No FMP error opening Ffile');
         READLN (Ffile, Fbuf);
         WRITELN ('Contents of file: ', Fbuf);
      END;
   CLOSE (Ffile);
   Error_Test (Ierr);
   IF (Ierr < 0) THEN
      WRITELN ('Error closing Ffile, FMP error ', Ierr);


   {Here we show what will happen with a non-FMP error.  Note that
    Ffile is no longer open so we will have a Pascal I/O type error.}

   WRITELN (Ffile, 'This will cause error exit')


END.   {Program}
```

MVDIR — THE CASE OF THE MOVING DIRECTORY

by John McCabe/HP Stanford Park Division

HOW WE CREATED THE PROBLEM

1. We had a 256 track disc LU in the pool
2. We allocated it as 250 tracks long.

   ```
   AC,cr,G,250
   ```

3. We loaded lots of data on the disc. Everything seemed fine. The directory track was track number 249.
4. We added the disc LU to the group's session switch table, as we planned to use it for a while.
5. We did a new system generation. We did not redefine the disc track map.
6. After we switched to the new system we did a mount cartridge to the disc LU.

   ```
   MC,lu,G
   ```

7. The system, knowing that the LU size was 256 tracks, used an old directory in track 255.
8. Since the system didn't know about the directory on track 249, we could not access any of our files. Things looked very bad.

HOW WE SOLVED OUR PROBLEM

The following solution may not be the best one, but it worked. It does not require the use of any special utilities. It takes advantage of the following:

The directory of cartridges contains the last track, and is on LU 2, not the cartridge LU. (The disc directory of files on the cartridge is at the end of the cartridge LU).

LSAVE and RESTR dumps disc LUs to tape track by track, regardless of the location of the directory.

WRITT and READT backs up LUs treating the directory as something special.

1. We did an LSAVE to store a copy of the disc to tape.

   ```
   LSAVE,,lu,,VE,title
   ```

2. We dismounted the disc.

   ```
   DC,cr,RR
   ```

3. We reallocated the cartridge with the old number of tracks, 250.

   ```
   AC,cr,G,250,if
   ```

   This wiped out all the data on the disc. This was ok.

4. We restored the disc from the tape saved in step 1.

   ```
   RESTR,,lu
   ```

   This restored everything as it was before the switch to the new system. Our old directory on track 249 accessed all our old files.

   We wanted to move the directory from track 249 to track 255 so we could utilize the last 6 tracks on the disc and so this same thing would not happen when we switched to another new system.

5. We used WRITT to back up the disc to tape.

   ```
   WRITT,cr
   ```

6. We released the disc back to the pool.

   ```
   DC,cr,RR
   ```

7. We used READT to restore the disc and move the directory from track 249 to track 255. We used the tape from step 5.

   ```
   READT,cr,,G,256
   ```

   This moved the directory properly and everything is as it should be.

HOW TO BUILD SYSTEM UTILITIES USING A DISC DIRECTORY AND EDIT/1000 SUBSYSTEM

by Bob Gordon/Boeing Computer Services

Manipulating large numbers of disc files can be tedious and subject to errors. The traditional method is to construct a transfer file or execute FMGR commands, one at a time, from the keyboard.

The disadvantages of this method are:

- Building transfer files is time consuming and tedious.
- Old transfer files are kept laying around consuming valuable disc space because the user feels they can be re-used and/or re-editted. After saving a half-dozen or so, the user forgets why the files were saved.
- Files are purged because the user needs to save disc space.

Our objective is to design and develop a utility from information provided by the system. It has to be easy to use, save disc space, be able to be reused, and be available to all users.

Using the information contained in a disc directory, one could develop a whole family of utilities to manipulate disc files. The question is, how can we do this automatically with minimal user intervention? By examining the characteristics of the Edit/1000 sub-system, one will note that Edit/1000 will perform automatic editing when used in the batch mode. This means the editor receives its commands from a "command file" or from the edit "run string". Hence, the editor does not have to stop between each command. The command files are user developed, oriented for a specific task, written only once, and are reusable. The result of this technique is a "back door" compiler. Every time one of the utilities is executed it compiles a set of FMGR commands in the form of a transfer file. When the transfer file completes its execution it is purged from the system.

The advantages to this method are:

- It's available to all users (placed on LU 2/3 or any global cartridge).
- New utilities can be developed from existing utilities.
- Disc space is not consumed by unwanted transfer files.
- Has to be developed only once.
- Easy to use.
- Re-usable
- No special FORTRAN or ASSEMBLY LANGUAGE programs are required. (They could be used for more flexibility or special applications)
- All software is designed around the disc directory.

The limitations are:

- More effective on large number of files.
- Limited to disc files.

- Requires a reasonable knowledge of Edit/1000.
- All files must come from the same LU.
- File names cannot be re-named during the transfer process.
- Files with security codes may require more than one pass.

In outline form, the utilities are structured as follows:

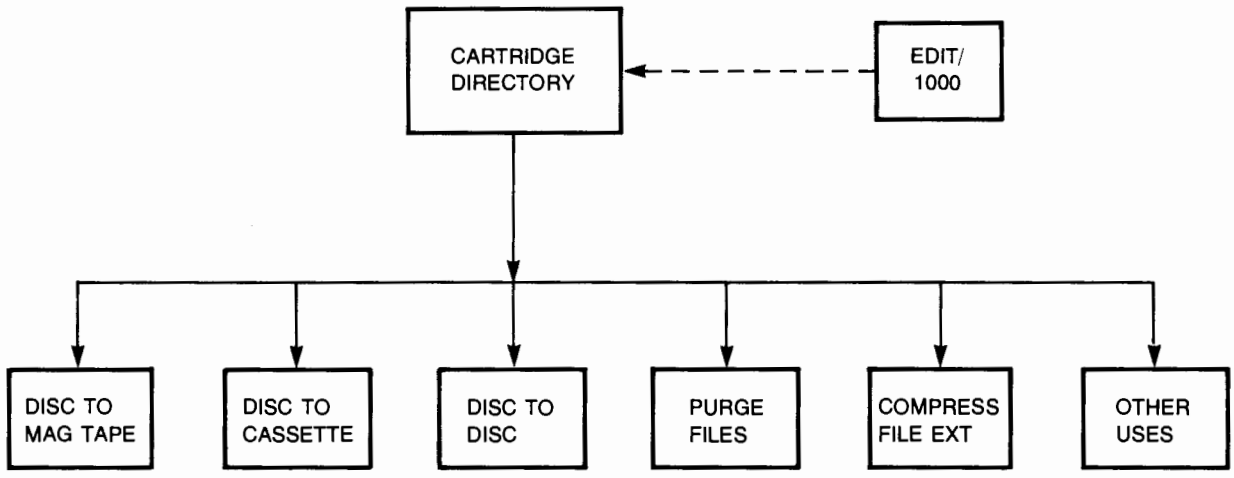
- Request information from the user, e.g. LU.
- Create a temporary scratch file.
- Assign the scratch file as the list device.
- List the directory into the scratch file.
- User manually edits directory entries in scratch file.
- Edit/1000 converts contents of scratch file into FMGR commands.
- Executes "edited" transfer file.
- At termination the transfer file is purged from the system.
- List result/trail on terminal.

In reviewing the utilities below, one will see that various embellishments can be made. This is the whole idea of the concept, building off existing software to fit your needs.

REFERENCES:

HP Edit/1000 User's Guide part # 92074-90001 pages 2-35 through 2-44. EXTENT Utility written by Dave Markwald, Hewlett Packard, Bellevue, WA.

FLOW CHART:



The STORE FILES ON TAPE utility has been augmented with comments to explain the various steps. The other utilities are left uncommented but are structured in a similar manner.

```
*******************************************************************************
************************** U T I L I T I E S ******************************
*******************************************************************************


****************   S T O R E   F I L E S   O N   T A P E ***************

:SV,4,,IH
-------------------------------------------------------operator instructions-
:DP,>  THE NAME OF THIS UTILITY IS "*STT".                                    ¦
:DP,>  WRITE DISC FILES ON MAGNETIC TAPE.                                     ¦
:DP,>                                                                         ¦
:DP,>  TWO TRANSFER FILES ARE GENERATED FROM THIS UTILITY.                    ¦
:DP,>  THESE TRANSFER FILES ARE DERIVED FROM A USER SELECTED CARTRIDGE        ¦
:DP,>  DIRECTORY. ONE TRANSFER FILE IS USED TO WRITE THE OTHER TRANSFER       ¦
:DP,>  FILE ON TAPE AS FILE ONE, FOLLOWED BY THE USER FILES.                  ¦
:DP,>  THE FIRST FILE ON TAPE IS USED AS A DIRECTORY WHICH WHEN READ IN       ¦
:DP,>  AND EXECUTED AS TRANSFER FILE WILL READ IN THE REMAINING FILES         ¦
:DP,>  FROM TAPE. EACH FILE TRANSFERRED HAS THE SAME NAME AS ITS SOURCE       ¦
:DP,>  FILE. THE CONTENTS OF FILE ONE ARE DISPLAYED ON LU 1.                  ¦
:DP,>  THIS UTILITY CAN TRANSFER DATA TO ANY CONVENTIONAL NON-DISC LU.        ¦
:DP,>  NOTE THAT THIS UTILITY WAS DESIGNED FOR MAGNETIC TAPE.                 ¦
:DP,>                                                                         ¦
:DP,>  THE TAPE FORMAT IS WRITTEN USING FMGR 'ST' COMMAND AND                 ¦
:DP,>  USES THE DEFAULT FORMAT OF THE 'ST' COMMAND.                           ¦
:DP,>                                                                         ¦
:DP,>  CALLING SEQUENCE:                                                      ¦
:DP,>                                                                         ¦
:DP,>  STEP 1 (TYPE) ::*STT (RETURN)                                         ¦
:DP,>  STEP 2        :SE,lu1,lu2 (RETURN)                                     ¦
:DP,>  STEP 3        ::(RETURN)                                               ¦
:DP,>                                                                         ¦
:DP,>  Where LU1 is the SOURCE LU, the cartridge containing the directory     ¦
:DP,>  list. LU2 is the DESTINATION lu, usually set to a non-disc lu,         ¦
:DP,>  such as LU 4,5,8,....                                                  ¦
:DP,>                                                                         ¦
:DP,>  EXAMPLE:                                                               ¦
:DP,>           ::*STT (RETURN)                                               ¦
:DP,>           :SE,:12,8 (RETURN)                                            ¦
:DP,>           :: (RETURN)                                                   ¦
:DP,>                                                                         ¦
:DP,>  ASSUMPTION:  USER IS FAMILIAR WITH THE PAGE EDITOR.                    ¦
:DP,>                                                                         ¦
:DP,>  THIS UTILITY WILL STAY IN THE EDITOR TO ALLOW THE USER TO              ¦
:DP,>  CUSTOM EDIT THE "TRANSFER FILE", USUALLY DELETING THOSE                ¦
:DP,>  DIRECTORY ENTRIES THAT YOU DO NOT WISH TO TRANSFER TO MAG              ¦
:DP,>  TAPE.                                                                  ¦
:DP,>                                                                         ¦
:DP,>  YOU MUST HAVE CAPABILITY OF 40 OR MORE TO EXECUTE THIS UTILITY.        ¦
:DP,>                                                                         ¦
:DP,>  MOUNT TAPE WITH WRITE RING.                                            ¦
-------------------------------------------------------------------------------¦
```

```
:PAUSE
-----------------¦
:CN,1G           ¦
:PU,X::12        ¦---> housekeeping to make sure temporary file doesn't exist.
:PU,Y::12        ¦
-----------------¦
:CR,Y::12:4:256  ----> create a temporary scratch file.
:LL,Y::12        ----> declare Y as the list device.
:DL,2G           ----> copy directory into scratch file.
:LL,1
:DP,>    CUSTOM EDITING CAN NOW BE DONE,IF DESIRED,
:DP,>    TYPE A COLON AND (RETURN) TO PROCEED.
:PAUSE
:RU,EDIT,Y::12,TR,STT0/   ----> edit command file STT0 deletes BLKS &EXTENTS
:RU,EDIT,Y::12,1/¦LN9999¦1/ --> list directory, let user edit file.
:DP,>
:DP,>  *********  END OF CUSTOM EDITING  **************
:DP,>
:ST,Y::12,X::12 --> create second transfer file (used to copy files to tape).
:RU,EDIT,Y::12,TR,STT1/ --> creates tape xfer file 1. STT1 is the command file
:RU,EDIT,X::12,TR,STT2/ --> creates xfer file to write files on tape with STT2
-----------------------------¦
:RU,EDIT,Y::12,$/¦C¦I:TR¦ER  ¦---> housekeeping to close transfer files.
:RU,EDIT,X::12,$/¦C¦I:TR¦ER  ¦
-----------------------------¦
:ST,Y::12,2G ----> write tape directory as file one on tape.
:SE,2G,1G,3G ----> set up the globals for X transfer file.
::X::12  ----> write files on tape.
:LI,Y::12 ---> list the contents of file one on terminal.
-----------¦
:PU,X::12  ¦----> purge temporary files from tape.
:PU,Y::12  ¦
-----------¦
:CN,2G ----> rewind tape.
:SE
:DP,>  *****************  END OF TRANSFER  ***************
:SV,0,,IH
::
```


```
*******************************************************************************
****************** E D I T  C O M M A N D  F I L E S ***********************
*******************************************************************************


IT IS ASSUMED THAT THE READER IS FAMILIAR WITH THE DIRECTORY FORMAT.


*******  STT0   DELETES ALL OCCURANCES OF 'BLKS' AND EXTENTS  ***************


SEREON   ----> set regular expression mode.
1$ D/BLKS/AQ/  ----> deletes all occurances of the string "BLKS".
1$ D/\+[0-9]* *$/AVQ/ ---> deletes all occurances of any extents lines.
ER


***** STT1 COMPILES TRANSFER FILE OF 'ST' COMMANDS, FOR TAPE DIRECTORY. ****


SEREON   ----> set regular expression mode
1 ----> go to line 1
I:SV,4,,IH ----> insert this command at line 1 in transfer file.
1$ X/ *{[^ ]*} *0<4>{[1-7]*}@/:ST,1G,&1::2G:&2:-1/Q/ ---> create FMGR commands
        from directory. Name, length, and type are substituted into ST string.
 :SV,4,,IH
1
MSTT3 ----> merge operator instruction into transfer file
$  ----> go to end of edit file.
-------------¦
 :CN,1G      ¦
 :SE         ¦---> insert these files at end of file.
 :SV,0,,IH   ¦
-------------¦
ER


**** STT2  COMPILES TRANSFER FILE OF 'ST' COMMANDS,WRITES FILES ON TAPE. ****


SEREON  ----> set regular expression mode.
1$ X/ *{[^ ]*} *0<4>{[1-7]*} *0*{[^ ]*}@/:ST,&1::2G:&2:&3,1G/Q/ ----> create
the FMGR ST commands from directory using the name, length, and type.

ER


**** STT3  USERS INSTRUCTIONS FOR FILE ONE ON TAPE.  **********************
       THIS IS THE OPERATOR INSTRUCTIONS ON HOW TO READ FILES FROM
           TAPE. IT CAN BE MODIFIED TO USERS NEEDS.
```

```
:DP,>  THIS UTILITY WILL TRANSFER FILES FROM SOURCE LU (I.E. MAG TAPE)
:DP,>  TO DESTINATION LU (I.E. DISC),THEREFORE THE USER MUST ENTER
:DP,>  THE SOURCE AND DESTINATION LU.
:DP,>  EXAMPLE:    ASSUME THE MAG TAPE  LU IS 8 AND THE DISK LU TO RECEIVE
:DP,>              THE TAPE FILES IS 12.  THEN ENTER AS FOLLOWS:
:DP,>  :SE,8,12(RETURN)
:DP,>  ::(RETURN)
:PAUSE
:CN,1G
:CN,1G,FF
:SV,0,,IH


****** PRS1  CREATES A TRANSFER FILE TO COMPRESS FILES WITH EXTENTS ***

SEREON   ----> set regular expression mode..
1$,D/BLKS/AQ/!ER  ----> delete all occurances of the string "BLKS".
1$,D/\+001* *$/AQ/!ER  ----> deletes all extent lines.
1$,X/ *{[^ ]*} *{[^ ]*} *{[^ ]*} *{[^ ]*} *{[^ ]*}@/::*EXTS,&1,&5,3G/Q/-->
create a transfer file to compress extents using a nested utility.

ER

*******************************************************************************
**************** E N D  O F  C O M M A N D  F I L E S  *************************
*******************************************************************************
```

```
********** T R A N S F E R  F I L E S  F R O M  crn T O  crn ***************

:SV,4,,IH
:DP,>  THE UTILITY NAME IS "*STORE".
:DP,>
:DP,>  THIS UTILITY WILL GENERATE A TRANSFER FILE WHICH WILL COPY
:DP,>  SELECTED FILES FROM ONE DISC CARTRIDGE TO ANOTHER DISC
:DP,>  CARTRIDGE. THE TRANSFER FILE IS COMPILED FROM THE DIRECTORY
:DP,>  ENTRIES AND CONVERTED TO FMGR STORE COMMANDS (ST).
:DP,>
:DP,>  CALLING SEQUENCE
:DP,>
:DP,>  :*STORE    Followed by :SE,lu1,lu2 followed by :(RETURN).
:DP,>
:DP,>  EXAMPLE:  :*STORE
:DP,>            :SE,12,13,sc
:DP,>            ::(RETURN)
:DP,>       Where 12 is source LU, and 13 is the destination LU.
:DP,>             sc is the security code to be written on lu2.
:DP,>  ASSUMPTION: USER IS FAMILIAR WITH PAGE EDITOR.
:DP,>
:DP,>  YOU MUST HAVE A CAPABILITY LEVEL OF 40 OR MORE TO EXECUTE THIS
:DP,>  UTILITY.
:DP,>
:DP,>  THIS UTILITY WILL STAY IN THE EDITOR TO ALLOW THE USER
:DP,>  TO CUSTOM EDIT THE TRANSFER FILE. DELETING THOSE
:DP,>  DIRECTORY ENTRIES THAT YOU DO NOT WISH TO TRANSFER.
:PAUSE
:PK,2G
:PU,S....G::12
:CR,S....G::12:4:256
:LL,S....G::12
:DL,1G
:LL,1
:DP,>     CUSTOM EDITING CAN NOW BE DONE,IF DESIRED.
:DP,>     TYPE A COLON AND (RETURN) TO CONTINUE.
:PAUSE
:RU,EDIT,S....G::12,TR,STT0/
:RU,EDIT,S....G::12,1/¦LN9999¦1/
:DP,>       ********* END OF CUSTOM EDITING *********
:RU,EDIT,S....G::12,SEREON¦1$,X/ *{[^ ]*}@/:ST,&1::1G,&1:3G:2G,/Q/¦ER
:RU,EDIT,S....G::12,$/¦C¦I:TR¦ER
::S....G::12
:PU,S....G::12
:DL,2G
:SE
:DP, ************** END OF FILE TRANSFER ****************
:SV,0,,IH
::
```

```
************** P U R G E   F R O M  C A R T R I D G E **********************


:SV,4,,IH
:DP,> THE NAME OF THIS UTILITY IS "*PURGE"
:DP,>
:DP,> THIS UTILITY WILL PURGE A SET OF USER SELECTED FILES, FROM DISC,
:DP,> BY EDITING THE DIRECTORY, CONVERTING THE DIRECTORY ENTRIES TO
:DP,> A TRANSFER FILE CONTAINING PURGE COMMANDS. AFTER THE TRANSFER FILE
:DP,> IS EXECUTED THE DISC CARTRIDGE IS PACKED.
:DP,>
:DP,> CALLING SEQUENCE
:DP,>
:DP,> :*PURGE     Followed by :SE,lu,SC  followed by :(RETURN).
:DP,>
:DP,> EXAMPLE:   ::*PURGE
:DP,>            :SE,23,sc  Where 23 is cartridge containing the files to
:DP,>                       be purged, SC IS THE SECURITY CODE IF REQUIRED.
:DP,>            ::(RETURN)
:DP,>
:DP,> ASSUMPTION: USER IS FAMILIAR WITH PAGE EDITOR.
:DP,>
:DP,> THIS UTILITY WILL STAY IN THE EDITOR TO ALLOW THE USER
:DP,> TO CUSTOM EDIT THE "TRANSFER FILE". DELETING THE DIRECTORY ENTRIES
:DP,> OF THE FILE YOU DO NOT WISH TO PURGE.
:DP,>
:DP,> YOU MUST HAVE CAPABILITY OF 40 OR MORE TO BE ABLE TO EXECUTE
:DP,> THIS UTILITY.
:PAUSE
:PU,S....P::12
:CR,S....P::12:4:256
:LL,S....P::12
:DL,1G
:LL,1
:DP,>     ANY CUSTOM EDITING CAN BE DONE NOW,IF DESIRED,
:DP,>     IF NOT,OR TO RESUME, ENTER "ER".
:RU,EDIT,S....P::12,TR,STT0/
:RU,EDIT,S....P::12,1/!LN9999!1/
:DP,         ********* END OF CUSTOM EDITING *********
:RU,EDIT,S....P::12,SEREON!1$,X/ *{[^ ]*}@/:PU,&1:2G:1G//!ER
:RU,EDIT,S....P::12,$/!C!I:PK,1G!I:TR!ER
::S....P::12
:PU,S....P::12
:DL,1G,XX ----> WHERE XX IS YOUR MASTER SECURITY CODE.
:DP, *************** END OF PURGE ********************
:SE
:SV,0,,IH
::
```

********** C O M P R E S S E X T E N T S **********

```
:SV,4,,IH
:DP,>  THE NAME OF THIS UTILITY IS  "*PRESS".
:DP,>
:DP,>  THIS UTILITY WILL COMPRESS EXTENTS IN A SPECIFIED CARTRIDGE. USES
:DP,>  CARTRIDGE DIRECTORY LISTING TO BUILD A SET OF COMMANDS TO COMPRESS
:DP,>  ANY OR ALL EXTENTS THAT MAY APPEAR IN A GIVEN CARTRIDGE. THIS PROCESS
:DP,>  ALLOWS THE SYSTEM TO RECLAIM UNUSED SPACE ON THE SPECIFIED
:DP,>  CARTRIDGE.
:DP,>
:DP,>  CALLING SEQUENCE:
:DP,>
:DP,>  STEP 1 (TYPE) :*PRESS (RETURN)
:DP,>  STEP 2        :SE,lu1 (RETURN)       (CARTRIDGE TO COMPRESS EXTENTS)
:DP,>  STEP 3        : (RETURN)
:DP,>
:DP,>  Where lu1 is the LU to compress EXTENTS.
:DP,>
:DP,>  EXAMPLE:
:DP,>           ::*PRESS (RETURN)
:DP,>           ::SE,12 (RETURN)
:DP,>           :: (RETURN)
:DP,>
:DP,>  ASSUMPTION:  USER IS FAMILIAR WITH THE PAGE EDITOR.
:DP,>
:DP,>  YOU MUST HAVE CAPABILITY OF 40 OR MORE TO EXECUTE THIS UTILITY.
:DP,>
:DP,>  IF THERE ARE NO EXTENTS IN THE CARTRIDGE, THEN THE UTILITY
:DP,>  WILL STOP IN THE EDITOR; ENTER ' ER AND return' TO CONTINUE.
:DP,>  THE CONTENTS OF THE CARTRIDGE ARE NOT CHANGED IF THIS CONDITION
:DP,>  EXISTS. THIS IS AN EDIT CONSTRAINT.
:DP,>
:PAUSE
:PU,Y::12
:CR,Y::12:4:256
:LL,Y::12
:DL,1G,XX
:LL,1
:RU,EDIT,Y::12,TR,PRS1/
:RU,EDIT,Y::12,$/!C!I:TR!ER
:CA,3,1G
::Y::12
:SV,4,,IH
:PK,3G
:PU,Y::12
:DL,3G
:SE
:DP,>  *****************  END OF COMPRESSION *************
:SV,0,,IH
::
```

```
****************** D E L E T E  E X T E N T S ******************************

(This is nested into utility *PRESS)


:SV,4,,IH,** FILE EXTENTS DELETE  FEB. 1982  D. MARKWALD NSR-BELLEVUE
:**
:**      TR,*EXTS,FILE NAME,SC,CRN[,NEW SC]
:**   OR   :*EXTS,.......................
:**
:** GLOBAL USAGE
:**    1G - FILE NAME
:**    2G - ORIGINAL FILE SECURITY CODE
:**    3G - CRN(+) OR LU(-)
:**    4G - OPTIONAL NEW FILE SECURITY CODE
:**    6P - CURRENT FMGR ERROR
:** -24P - 4G'S TYPE
:**
:IF,-24P,GT,0,1
:CA,4,2G
:** DP,1G,2G,3G,4G,-24P
:CA,6:P,0
:PU,)TMPEX:2G:3G
:IF,6P,EQ,0,1
:IF,6P,NE,-6,9
:CA,6:P,0
:PU,)TMPEX:4G:3G
:IF,6P,EQ,0,1
:IF,6P,NE,-6,4
:CA,6:P,0
:ST,1G:2G:3G,)TMPEX:4G:3G::-1
:IF,6P,EQ,0,4
:PU,)TMPEX:4G:3G
:PU,)TMPEX:2G:3G
:DP,*EXTS FMGR ERROR
::
:CA,6:P,0
:PU,1G:2G:3G
:IF,6P,NE,0,-7
:RN,)TMPEX:4G:3G,1G
:SE
:SV,0,,IH
```

```
*******************************************************************************
************************ EXAMPLE TRAIL OF "*PRESS" ****************************
*******************************************************************************
```

The following text is what one would see on the CRT during execution of this utility.

```
:*PRESS
 >  THE NAME OF THIS UTILITY IS  "*PRESS".
 >
 >  THIS UTILITY WILL COMPRESS EXTENTS IN A SPECIFIED CARTRIDGE. USES
 >  CARTRIDGE DIRECTORY LISTING TO BUILD A SET OF COMMANDS TO COMPRESS
 >  ANY OR ALL EXTENTS THAT MAY APPEAR IN A GIVEN CARTRIDGE. THIS UTILITY
 >  ALLOWS THE SYSTEM TO RECLAIM UNUSED SPACE ON THE SPECIFIED
 >  CARTRIDGE.
 >
 >  CALLING SEQUENCE:
 >
 >  STEP 1 (TYPE):*PRESS (RETURN)
 >  STEP 2:SE,lu1 (RETURN)       (CARTRIDGE TO COMPRESS EXTENTS)
 >  STEP 3:(RETURN)
 >
 >  Where lu1 is the LU to compress EXTENTS.
 >
 >  EXAMPLE:
 >::*PRESS (RETURN)
 >::SE,12 (RETURN)
 >::(RETURN)
 >
 >  ASSUMPTION:USER IS FAMILIAR WITH THE PAGE EDITOR.
 >
 >  YOU MUST HAVE CAPABILITY OF 40 OR MORE TO EXECUTE THIS UTILITY.
 >
 >  IF THERE ARE NO EXTENTS IN THE CARTRIDGE, THEN THE UTILITY
 >  WILL STOP IN THE EDITOR; ENTER ' ER AND return' TO CONTINUE.
 >  THE CONTENTS OF THE CARTRIDGE ARE NOT CHANGED IF THIS CONDITION
 >  EXISTS.
 >
 :PAUSE
:SE,32
::
```

SET UP YOUR 2608 LINE PRINTER

by Linnea L. Fort/Central Iowa Power Co-op, Cedar Rapids, ID

This program deals with the HP 2608 line printer using driver DVB12 and is run on an HP 1000 computer.

SETPR PROGRAM

This program sets up the 2608A line printer to the operator's specifications. It will:

1. set or clear "auto page eject"
2. set lines per inch (6 or 8)
3. set paper length (in inches)
4. set vertical margin
5. set horizontal margin
6. set normal or double size print

The program lists defaults at beginning of program. To default any parameter, hit return key.

To inhibit top-of-page on your 2608A printer, clear "auto page eject".

The printer will remain set to the specified parameters, until:

1. power on switch on 2608A printer is turned off
2. boot-up
3. reset button on 2608A printer is pushed
4. re-run SETPR program

This program comes in handy for legal sized forms or any form that is not 11 inches in length. Inhibiting T-O-P comes in handy for printing labels. Resetting horizontal margins comes in handy for vertically perfed paper where your 1st column starts to the left of that perforation. The applications are merous.

This program can be put in your WELCOM file, if it is to be used a lot.

```
FTN4,L
C
      PROGRAM SETPR
C
C
C               05/07/81              RTE-IVB           LINN FORT - CIPCO
C      REVISED    /  /
C
C               SOURCE - SETPRS::CR
C               RELO   - SETPRR::CR
C
C               COMPILE: RU,FTN4,SETPRS::CR,6,SETPRR::CR
C               LOAD:    RU,LOADR,,SETPRR::CR,,NL
```

```
C
C         THIS PROGRAM SETS UP THE PRINTER TO OPERATOR'S SPECIFICATIONS.
C
C                     (1) SETS OR CLEARS AUTO PAGE EJECT
C                     (2) SETS LINES PER INCH
C                     (3) SETS PAPER LENGTH
C                     (4) SETS VERTICAL MARGIN
C                     (5) SETS HORIZONTAL MARGIN
C                     (6) SETS DOUBLE SIZE PRINT
C
C         TO RESET PRINTER TURN POWER SWITCH OFF, THEN ON    OR RE-BOOT
C
      DIMENSION IBUFA(127),MARG2(16)
C
      DATA MARG2 /00B,
     1           01B,
     1           02B,
     1           03B,
     1           04B,
     1           05B,
     1           06B,
     1           07B,
     1           10B,
     1           11B,
     1           12B,
     1           13B,
     1           14B,
     1           15B,
     1           16B,
     1           17B/
C
      ILU=6
      ILPI=6
      ALNGTH=11.0
      MARGIN=6
      IMARG=1
C
C=============================================================
      WRITE(1,1)
    1 FORMAT(//"THIS PROGRAM SETS PARAMETERS ON THE 2608A LINE",
     1" PRINTER",//,5X,"DEFAULTS ARE:",//,
     1               7X,  "PRINTER LU          =  6",/,
     1               7X,  "AUTO PAGE EJECT     =  SET",/,
     1               7X,  "LINES PER INCH      =  6",/,
     1               7X,  "PAPER LENGTH        = 11 INCHES",/,
     1               7X,  "VERTICAL MARGIN     =  6",/,
     1               7X,  "HORIZONTAL MARGIN   =  1",/,
     1               7X,  "PRINT SIZE          =  NORMAL")
C ...
    5 WRITE(1,10)
   10 FORMAT(//,"ENTER PRINTER LU ? _")
C
      READ(1,*) ILU
C ...
   15 WRITE(1,20)
   20 FORMAT("AUTO PAGE EJECT (SET OR CLEAR) ? _")
C
      READ(1,25) IAUTO
   25 FORMAT(A2)
```

```
C
      IF(IAUTO.EQ.2H  ) IAUTO=2HSE
C
      IF(IAUTO.NE.2HSE .AND. IAUTO.NE.2HCL) GO TO 15
C ...
   30 WRITE(1,35)
   35 FORMAT("ENTER LINES PER INCH (6 OR 8) ? _")
C
      READ(1,*) ILPI
C
      IF(ILPI.EQ.6) ILINE=000B
      IF(ILPI.EQ.8) ILINE=200B
C
      IF(ILPI.NE.6 .AND. ILPI.NE.8) GO TO 30
C ...
      IF(IAUTO.EQ.2HSE) WRITE(1,40)
   40 FORMAT("ENTER PAPER LENGTH (IN INCHES) ? _")
C
      IF(IAUTO.EQ.2HSE) READ(1,*) ALNGTH
C ...
      IF(IAUTO.EQ.2HSE) WRITE(1,50)
   50 FORMAT("ENTER VERTICAL MARGIN LINES ? _")
C
      IF(IAUTO.EQ.2HSE) READ(1,*) MARGIN
C ...
   75 WRITE(1,80)
   80 FORMAT("ENTER HORIZONTAL MARGIN (1-16) ? _")
C
      READ(1,*) IMARG
C
      IF(IMARG.LT.1 .OR. IMARG.GT.16) GO TO 75
C ...
   85 WRITE(1,90)
   90 FORMAT("NORMAL OR DOUBLE SIZE PRINT ? _")
C
      READ(1,25) ISIZE
      IF(ISIZE.EQ.2H  ) ISIZE=2HNO
      IF(ISIZE.NE.2HNO .AND. ISIZE.NE.2HDO) GO TO 85
C
C*************************************************************
C                                                            *
C           COMPUTE ALL FACTORS                              *
C                                                            *
C*************************************************************
C
C
C            COMPUTE LINES PER PAGE
C
      LPPAG1 = ALNGTH * ILPI
C
C          LINES PER PAGE CANNOT EXCEED 127
C
      IF(LPPAG1.GT.127) GO TO 5
C
C           COMPUTE PRINTABLE LINES
C
      LPPAG2 = ALNGTH * ILPI - MARGIN
C
C           MAKE SURE PRINTABLE LINES IS = OR < 0
```


```
C
      IF(LPPAG2.LE.1) GO TO 5
C
C           FIGURE IBUFL
C
      IBUFL = ILINE + LPPAG1
C
C             SET UP IBUFA BUFFER FOR PRINTER
C
C                    "TOP-OF-PAGE" SWITCH
C
      IBUFA(1) = 5B
C
C                    "PRINT ON THIS LINE" SWITCH
C
      DO 100 I=2,LPPAG2-1
      IBUFA(I) = 4B
  100 CONTINUE
C
C                    "BOTTOM LINE" SWITCH
C
      IBUFA(LPPAG2) = 6B
C
C                    "DON'T PRINT ON THIS LINE" SWITCH
C
      DO 200 I=LPPAG2+1,LPPAG1
      IBUFA(I) = 0B
 200  CONTINUE
C
C                 SET UP 2608A PRINTER
C
      CALL EXEC(2,1000B+ILU,IBUFA,IBUFL)
      CALL EXEC(3,2100B+ILU,MARG2(IMARG))
      IF(ISIZE.EQ.2HDO) CALL EXEC(3,3000B+ILU,1)
      IF(ISIZE.EQ.2HNO) CALL EXEC(3,3000B+ILU,0)
      IF(IAUTO.EQ.2HCL) CALL EXEC(3,1100B+ILU,65)
      IF(IAUTO.EQ.2HSE) CALL EXEC(3,1100B+ILU,64)
C
      END
      END$
```

1351A GRAPHICS GENERATOR WITH A 21MXM COMPUTER IN RTE-IVB

K.H. Kitching, J. Robinson/Canadian Forces Maritime
Nanoose, British Columbia

The first introduction to any graphics system is to draw something on the screen to establish communication with the device. In our case it consisted of a Mexican hat containing 740 vectors. Our first experience was extremely disappointing since although the control language is relatively simple and easy to learn, and the commands do exactly what the book says they should do, it took 15.8 seconds to draw the picture. The following article outlines procedures which have been adopted to improve this performance to the point where even an M series computer can draw the picture in .6 seconds.

The most important thing to learn is that groups of instructions should be pre-packaged into an array in computer memory and then transmitted as a block to the graphics generator. This reduced the time consumed from 15.8 to 6.3 seconds. The time saved was entirely consumed in handshakes between the graphics generator and the computer. The next major step forward is to use the sub-routine "CNUMD" to convert numbers to ASCII in lieu of the formatter. This further reduced the time required to draw the picture from 6.3 seconds to 2.0 seconds. The final maneuver which can only be used for static information is to pre-package the ASCII command string in a disc file. The total time required to open the file and transmit the picture to the screen was .6 seconds.

One final point to remember is that the graphics generator receives its information on the IEEE-488 interface bus and the configuration of this bus can affect the presentation and the time consumed. When transmitting the picture from an ASCII disc file, the visual presentation is instantaneous if the bus is configured for direct memory access. The picture takes a visible fraction of a second to appear if the bus is configured without direct memory access. Configuring the bus for DMA can be accomplished from the file manager as follows:

```
:CN,LU,25B,37000B
```

Further experimentation showed that the bus should not be configured for DMA when the vectors are being transmitted individually. In this case it raised the total time from 15.8 to 18.4 seconds, obviously considerably complicating the transmission protocols.

DESIGN OBJECTIVES OF A FILE SYSTEM

Our computer system is used to control a real time test environment using 3 electronic counters, a signal synthesizer, an arbitrary waveform generator and a 96-channel multiplex A/D converter. It also uses a 5451B Fourier Analyzer as a satellite. Displays on the graphics generator are scheduled by event, by time, by operator control, and by request from the satellite. The variety of displays is evolving rapidly. It soon became apparent that some system was required to make the file structure of the graphics translator transparent to the operator and the programmer. The best approach would be to allow the system manager to experiment with different file configurations without requiring any of the display generating programs to be recompiled. All of these objectives were achieved by use of a program which is scheduled by each program desiring to display on the graphics generator. The calling program defines by parameters what it requires and the "son" returns a parameter string defining the file number, the number of files in the group assigned and the start location of each file assigned. In our current system the file structure is related to the functions by the system manager. A fairly simple extension of this system would allow the computer to make dynamic allocations similar to the current RTE partition allocations.

A control program has been written for the operator which allows him to define which pattern will be displayed on any or all graphics screens. This control program determines the file structure by scheduling the "son" and displays, inhibits, or erases the appropriate file or group of files by an operator function definition without the operator being aware of the file structure or even necessarily that there is a file structure.

A FAST SUBROUTINE

To illustrate these principles first is a subroutine for plotting a prescaled array. This subroutine transmits a prepackaged array without using the formatter. The integer array to be plotted consists of NUM points each defined by a prescaled x and y pair.

```
FTN,L
      SUBROUTINE VLIN(IFL,IAR,NUM), PLOT LINE VECTOR GRAPHICS  811209.1308
C*KK* PLOT AN ARRAY ON THE 1351A GRAPHICS GENERATOR
C     IFL DEFINES THE START LOCATION IN GRAPHICS TRANSLATOR MEMORY
C     IAR IS A PRESCALED ARRAY OF XY POINTS
C     NUM  IS THE NUMBER OF VECTORS IN THE ARRAY LIMITED BY IOUT TO 100
      DIMENSION IAR(2,NUM),IBUF(6),IBF(3),IOUT(613),IFMT(19)
      DATA IFMT/':FL     ,:PE0,:PA     ,     ;:PE1,:PA'/
C
C     ENCODE THE START LOCATION
C
      CALL CNUMD(IFL,IBF)
      IFMT(3)=IBF(2)
      IFMT(4)=IBF(3)
C
C     ENCODE THE FIRST POINT TO BE PLOTTED WITH PEN UP
C
      CALL CNUMD(IAR(1,1),IBF)
      IFMT(10)=IBF(2)
      IFMT(11)=IBF(3)
      CALL CNUMD(IAR(2,1),IBF)
      IFMT(13)=IBF(2)
      IFMT(14)=IBF(3)
C
C     STORE THE ENCODED STRING IN THE ARRAY FOR LATER OUTPUT
C
      DO 5 N=1,19
    5 IOUT(N)=IFMT(N)
      KK=19
C
C     ENCODE THE REST OF THE ARRAY TO BE PLOTTED WITH PEN DOWN
C
      IBUF(3)=2H,
      IBUF(6)=2H;
      DO 20 N=2,NUM
      CALL CNUMD(IAR(1,N),IBF)
      IBUF(1)=IBF(2)
      IBUF(2)=IBF(3)
      CALL CNUMD(IAR(2,N),IBF)
      IBUF(4)=IBF(2)
      IBUF(5)=IBF(3)
      DO 10 M=1,6
   10 IOUT(M+KK)=IBUF(M)
      KK=KK+6
   20 CONTINUE
C
C     PASS THE ENCODED ARRAY TO THE GRAPHICS TRANSLATOR FOR PLOTTING
C
      CALL EXEC(2,23,IOUT,KK)
      END
```

A TRANPARENT FILE SYSTEM

Next we present the program VFL which on a menu basis allows initialization and control of the graphics translator.

```
FTN4,L
      PROGRAM VFL(3,99),   811209.1308
C**JR* CONTAINS VECTOR GRAPHICS FILE STRUCTURE AND ALLOCATION
C
C  PERIPHERALS.  VECTOR GRAPHICS SYSTEM 1351A WITH TWO DISPLAYS.
C                                        1317A AND 1311B.
C
C
C  PARAMETERS.  1.  -1 FOR NO OUTPUT     IEX
C               2.  MENU ITEM            IVM
C               3.  DISPLAY MENU         IDM
C               4
C               5.  SEARCH VALUE
C
C  FILE CONFIGURATION IS LOADED INTO COMMON 201 AT INITIALIZATION
C
C  DATA IVF PARAMETERS. FILE NUMBER, STARTING LOCATION, MENU ITEM, MNEMONIC
C  IVM IS POSITION IN THE MENU LIST
C  IDM IF THE FUNCTION MENU ITEM
C
C
       DIMENSION IVF(68),IP(5),IAR(1000)
       DATA IVF/1,   1, 1,2HSS,
      2         2,1001, 2,2HHD,
      3         3,1601, 3,2HI1,
      4         4,2601, 3,2HI2,
      5         5,3201, 6,2HWF,
      6         6,7201, 4,2HD1,
      7         7,7401, 4,2HD2,
      8         8,7501, 4,2HD3,
      9         9,7601, 5,2HF1,
      A        10,7801, 5,2HF2,
      B        11,7901, 5,2HF3,
      C        12,8001, 9,2HT1,
      D        13,8046, 9,2HT2,
      E        14,8091, 9,2HT3,
      F        15,8136,10,2HC1,
      G        16,8181,10,2HC2,
      G         0,8191,0,-99 /
       CALL RMPAR(IP)
       LU=LOGLU(IDUMMY)
       IETX=1400B
       IEX=IP(1)
       IF(IP(2).GT.0) GO TO  50
C
C  INTERACTIVE
```

```
C
   20 WRITE(LU,25)
   25 FORMAT(     4X,"VECTOR GRAPHICS MENU",/,
     1            10X,"1.  SYSTEM STATUS       ",/,
     2            10X,"2.  DATA HISTORY        ",/,
     3            10X,"3.  3D IMAGES           ",/,
     4            10X,"4.  DIRECTION FINDERS   ",/,
     5            10X,"5.  SPECTRA             ",/,
     6            10X,"6.  WATERFALL DISPLAY   ",/,
     9            10X,"9.  TIMERS              ",/,
     A            9X,"10.  CLOCKS              ",/,
     B            9X,"11.  ALL FUNCTIONS       ",/,
     C            9X,"12.  INITIALIZATION DATA",/,
     D            9X,"13.  INITIALIZATION      ",/,
     E            9X,"14.  DEMONSTRATION FILES",/,
     F            9X,"15.  SEARCH FOR A MNEMONIC",/,
     G            9X,"99.  EXIT PROGRAM")
   30 WRITE(LU,'(2X,"ENTER THE DISPLAY MENU ITEM REQUIRED - _")')
      CALL INPUT(0,IP(2),1,LU,*30)
   50 IVM=IP(2)
      IF(IVM.EQ.99)   GO TO 8000
      IF((IVM.LT.1).OR.(IVM.GT.15)) GO TO 7000
      IF(IVM.EQ.12) GO TO 1200
      IF(IVM.EQ.13) GO TO 1300
      IF(IVM.EQ.14) GO TO 1400
      IF(IVM.EQ.15) GO TO 1500
      IF(IP(3).GT.0) GO TO 100
   60 WRITE(LU,65)
   65 FORMAT(     15X,"1.  LARGE DISPLAY ONLY",/,
     1            15X,"2.  SMALL DISPLAY ONLY",/,
     2            15X,"3.  BOTH DISPLAYS",/,
     3            15X,"4.  NONE",/,
     4            15X,"5.  ERASE THE FILE",/,
     5            14X,"99.  RETURN TO MAIN FILE")
   70 WRITE(LU,'(10X,"ENTER THE ONE REQUIRED - _")')
      CALL INPUT(0,IP(3),1,LU,*70)
  100 IDM=IP(3)
      IF(IDM.EQ.99)  GO TO 7000
      IF((IDM.LT.1).OR.(IDM.GT.5)) GO TO 60
      IF(IDM.EQ.1) JDM=2
      IF(IDM.EQ.2) JDM=1
      IF(IDM.EQ.3) JDM=0
      IF(IDM.EQ.4) JDM=3
      IF(IVM.EQ.11) GO TO 1100
      IF(IDM.EQ.5) GO TO 500
C
C  CHECK FILE FOR MNEMONIC
C
      DO 150 I=0,15
      IF(IVF(I*4+3).NE.IVM) GO TO 150
      WRITE(LU,'(":FF",I2,",:BF,:WX",I1,",:UF",I2,",:SX,:")')
     1IVF(I*4+1),JDM,IVF(I*4+1)
      WRITE(23,'(":FF",I2,",:BF,:WX",I1,",:UF",I2,",:SX,:")')
     1IVF(I*4+1),JDM,IVF(I*4+1)
  150 CONTINUE
      GO TO 7000
C
C  ERASE A FILE
```

```
C
  500 DO 510 I=0,15
      IF(IVF(I*4+3).NE.IVM) GO TO 510
      WRITE(23,'(":EF",I2,",:")') IVF(I*4+1)
  510 CONTINUE
      GO TO 7000
C
C  HANDLE 'ALL FILES'
C
 1100 IF(IDM.EQ.5) GO TO 1150
 1120 DO 1130 I=0,15
      WRITE(23,'(":FF",I2,",:BF,:WX",I1,",:UF",I2,",:SX,:")')
     1IVF(I*4+1),JDM,IVF(I*4+1)
 1130 CONTINUE
      GO TO 7000
 1150 DO 1160 I=0,15
      WRITE(23,'(":EF",I2,",:")') IVF(I*4+1)
 1160 CONTINUE
      GO TO 7000
C
C  INITIALIZATION DATA
C
 1200 DO 1210 I=0,15
      WRITE(LU,'(4X,"FILE ",I2," INITIALISED FROM ",I4,"  TO ",I4)')
     1IVF(I*4+1),IVF(I*4+2),IVF(I*4+6)-1
 1210 CONTINUE
      GO TO 7000
C
C  INITIALIZATION
C
 1300 CONTINUE
      CALL LURQ(1,23,1)
      CALL CNFG(23,1,37000B)
      WRITE(23,'(A2,":EM,:EN,:EX,:SN,:SX,:UM,:")') 1424B
      DO 1310 N=1,1000,4
      IAR(N)=2H:P
      IAR(N+1)=2HA0
      IAR(N+2)=2H,0
 1310 IAR(N+3)=2H;
      DO 1340 I=0,15
      WRITE(23,'(":FL",I4,",:NF",I2,",:PE0,")') IVF(I*4+2),IVF(I*4+1)
      N=(IVF(I*4+6)-IVF(I*4+2))*4
 1320 IF(N.GT.1000)THEN
       CALL EXEC(2,23,IAR,1000)
      ELSE
       CALL EXEC(2,23,IAR,N)
      ENDIF
      N=N-1000
      IF(N.GT.0)GO TO 1320
      WRITE(23,'(":SN,")')
 1340 CONTINUE
      CALL LURQ(0,23,1)
      GO TO 7000
```

```
C
C  DEMONSTRATION FILES
C
 1400 CALL LURQ(1,23,1)
      WRITE(23,'(":CS1,:")')
      DO 1450 IVM=1,10
      DO 1405 I=0,15
      IF(IVF(I*4+3).EQ.IVM) GO TO 1406
 1405 CONTINUE
      WRITE(LU,'("THERE IS NO FILE FOR ITEM ",I2)') IVM
 1406 IFN=IVF(I*4+1)
      IVP=800-IVM*75
      GO TO (1411,1412,1413,1414,1415,1416,1450,1450,1419,1420) IVM
 1411 WRITE(23,'("FF",I2,",:PE0,:PA200,",I3,";:PE1,:TX 1.  ",
     1"  SYSTEM STATUS",A1,",:")') IFN,IVP,IETX
      GO TO 1450
 1412 WRITE(23,'("FF",I2,",:PE0,:PA200,",I3,";:PE1,:TX 2.  ",
     1"  DATA HISTORY ",A1,",:")') IFN,IVP,IETX
      GO TO 1450
 1413 WRITE(23,'("FF",I2,",:PE0,:PA200,",I3,";:PE1,:TX 3.  ",
     1"  3D IMAGES",A1,",:")') IFN,IVP,IETX
      GO TO 1450
 1414 WRITE(23,'("FF",I2,",:PE0,:PA200,",I3,";:PE1,:TX 4.  ",
     1"  DIRECTION FINDERS",A1,",:")') IFN,IVP,IETX
      GO TO 1450
 1415 WRITE(23,'("FF",I2,",:PE0,:PA200,",I3,";:PE1,:TX 5.  ",
     1"  SPECTRA",A1,",:")') IFN,IVP,IETX
      GO TO 1450
 1416 WRITE(23,'("FF",I2,",:PE0,:PA200,",I3,";:PE1,:TX 6.  ",
     1"  WATERFALL DISPLAY",A1,",:")') IFN,IVP,IETX
      GO TO 1450
 1419 WRITE(23,'("FF",I2,",:PE0,:PA200,",I3,";:PE1,:TX 9.  ",
     1"  TIMERS",A1,",:")') IFN,IVP,IETX
      GO TO 1450
 1420 WRITE(23,'("FF",I2,",:PE0,:PA200,",I3,";:PE1,:TX10.  ",
     1"  CLOCKS",A1,",::PE0,:")') IFN,IVP,IETX
 1450 CONTINUE
      CALL LURQ(0,23,1)
      GO TO 7000
C
C  RETURN THE BUFFER
C
 1500 NUM=0
      DO 1510 I=15,0,-1
      IF(IAND(IP(5),177400B).NE.IAND(IVF(I*4+4),177400B))GO TO 1510
      NUM=NUM+1
      INUM=I
 1510 CONTINUE
      IVF(67)=NUM
      IF(NUM.NE.0) IVF(68)=INUM
      CALL EXEC(14,2,IVF,68)
      GO TO 7000
 7000 IF(IEX.EQ.-1) GO TO 8000
      IP(3)=0
      GO TO 20
 8000 CALL EXEC(6,0)
      END
```

Next we present a simple program using this system.

```
FTN,L
      PROGRAM TVLIN
C*KK* ILLUSTRATE USE OF VLIN AND VFL
      DIMENSION IBUF(2,100),NVFL(3),IVF(68)
      DATA NVFL/'VFL   '/
      LU=  LOGLU(ID)
C
C     RUNP SCHEDULES VFL WITH WAIT AND ASKS FOR AREA KK
C
      CALL RUNP(LU,9,NVFL,-1,15,0,0,2HKK,N,0)
      CALL EXEC(14,1,IVF,68)
      I=IVF(68)            !I IS THE LINE IN IVF CONTAINING KK
      IF(I.EQ.-99)THEN     !THE AREA ASKED FOR IS NOT DEFINED
       WRITE(LU,'("VECTOR GRAPICS AREA ",A2," IS NOT DEFINED")')IVF(67)
       GO TO 100
      ENDIF
      IFL=IVF(I*4+2)       !IFL IS THE START LOCATION
C
C     SET UP SOME DEMO DATA
C
      DO 10 N=1,100
      IBUF(1,N)=N*10
   10 IBUF(2,N)=600+N
C
C     PLOT THE ARRAY IBUF
C
      CALL VLIN(IFL,IBUF,100)
  100 END
```

And finally a handy little subroutine which schedules the program even if it is not memory resident.

```
FTN4,L
      SUBROUTINE RUNP(LU,ICODE,NAME,IP1,IP2,IP3,IP4,IP5,IST,NST), PROGRA
     1M SCHEDULER  811209.1308
C**JR*ROUTINE TO RUN A PROGRAM WHICH MAY NOT BE AVAILABLE
C  IT SHOULD REPLACE A PROGRAM NOT IN THE ID LIST
C  ERROR MESSAGES PRINTED ON THE GIVEN LU
C
C  LU    - LOGICAL UNIT NUMBER OF THE CALLING TERMINAL
C  ICODE - THE EXEC SCHEDULING CODE.   9,10,23,24 ALLOWED
C  NAME  - PROGRAM NAME. A THREE WORD ARRAY
C  IP1 TO IP5 ARE 5 INPUT PARAMETERS
C  IST   - THE ASCII STRING TO BE PASSED.  DO NOT DEFAULT TO 0 USE IST
C  NST   - NUMBER OF WORDS IN THE STRING IST
C
C WARNING. DO NOT DEFAULT ANY PARAMETERS. USE ZERO EXCEPT FOR IST.
C
      DIMENSION NAME(3),IST(1),IDCB(144),IDT1(17),IDT2(15),IDT3(12),
     1LER(3)
      DATA IDT1/'RUNP ERROR MESSAGE    NAME    IAIB'/
      DATA IDT2/'RUNP - OPEN ERROR IERR    NAME'/
      DATA IDT3/'RUNP - IDRPL ERROR   IER'/
   10 CALL EXEC(ICODE+100000B,NAME,IP1,IP2,IP3,IP4,IP5,IST,NST)
      GO TO 100
   20 CONTINUE
      RETURN
  100 CALL ABREG(IA,IB)
      IF((IA.EQ.2HSC).AND.(IB.EQ.2H05)) GO TO 200
      IDT1(11)=NAME(1)
      IDT1(12)=NAME(2)
      IDT1(13)=NAME(3)
      IDT1(16)=IA
      IDT1(17)=IB
      CALL EXEC(1,LU+400B,IDT1,17)
      RETURN
  200 CALL OPEN(IDCB,IERR,NAME)
      IF(IERR.GE.0) GO TO 220
      CALL CNUMD(IERR,LER)
      IDT2(10)=LER(2)
      IDT2(11)=LER(3)
      IDT2(13)=NAME(1)
      IDT2(14)=NAME(2)
      IDT2(15)=NAME(3)
      CALL EXEC(1,LU+400B,IDT2,15)
      RETURN
  220 CALL IDRPL(IDCB,IER,NAME)
      IF(IER.NE.0) GO TO 220
      CALL CLOSE(IDCB)
      GO TO 10
  240 CALL CNUMD(IER,LER)
      IDT3(11)=LER(2)
      IDT3(12)=LER(3)
      CALL EXEC(1,LU+400B,IDT3,12)
      RETURN
      END
```

JOIN AN HP 1000 USER GROUP!

Here are the groups that we know of as of December 1980. (If your group is missing, send the Communicator/1000 editor all of the appropriate information, and we'll update our list.)

NORTH AMERICAN HP 1000 USER GROUPS

Area	User Group Contact
Arizona	Jim Drehs 7120 E. Cholla Scottsdale, Arizona 85254
Boston	LEXUS P.O. Box 1000 Norwood, Mass. 02062
Chicago	David Olson Computer Systems Consultant 1846 W. Eddy St. Chicago, Illinois 60657 (312) 525-0519
Greenville/S. C.	Henry Lucius III American Hoechst Corp. P.O. Box 1400 Greer, South Carolina 29651 (803) 877-8471
Huntsville/Ala.	John Heamen ED35 George C. Marshall Space Flight Ctr. Nasa Marshall Space Flight Ctr., AL. 35812
Montreal	Erich M. Sisa Siemens Electric Ltd. 7300 Trans Canada Highway Pointe Claire, Quebec H9R 1C7
New Mexico/El Paso	Guy Gallaway Dynalectron Corporation Radar Backscatter Division P.O. Drawer O Holloman AFB, NM 88330
New York/New Jersey	Paul Miller Corp. Computer Systems 675 Line Road Aberdeen, N.J. 07746 (201) 583-4422

NORTH AMERICAN HP 1000 USER GROUPS (CONTINUED)

Area	User Group Contact
Philadelphia	Dr. Barry Perlman RCA Laboratories P.O. Box 432 Princeton, N.J. 08540
Pittsburgh	Eric Belmont Alliance Research Ctr. 1562 Beeson St. Alliance, Ohio 44601 (216) 821-9110 X417
San Diego	Jim Metts Hewlett-Packard Co. P.O. Box 23333 San Diego, CA 92123
Toronto	Nancy Swartz Grant Hallman Associates 43 Eglinton Av. East Suite 902 Toronto M4P1A2
Washington/Baltimore	Mal Wiseman Hewlett-Packard Co. 2 Choke Cherry Rd. Rockville, MD. 20850
General Electric Co. (GE employees only)	Stu Troop Special Purpose Computer Ctr. General Electric Co. 1285 Boston Ave. Bridgeport, Conn. 06602

OVERSEAS HP 1000 USER GROUPS

Belgium	J. Tiberghien Vrije Universiteit Brussel Afdeling Informatie Pleinlaan 2 1050 Brussel Belgium Tel. (02) 6485540

OVERSEAS HP 1000 USERS GROUPS (CONTINUED)

Area	User Group Contact
France	Jean-Louis Rigot Technocatome TA/DE/SET Cadarache BP.1 13115 Saint Paul les Durance France Tel. (042) 253952
Germany	Hermann Keil Vorwerk + Co Elektrowerke Abt. TQPS Rauental 38-40 D-5600 Wuppertal 2 W. Germany Tel. (0202) 603044
Netherlands	Albert R. Th. van Putten National Institute of Public Health Antonie van Leeuwenhoeklaan 9 Postbox 1 3720 BA Bilthoven The Netherlands Tel. (030) 742344
Singapore	W. S. Wong Varta Private Ltd. P.O. Box 55 Chai Chee Post Office Singapore Tel. 412633
Switzerland	Graham Lang Laboratories RCA Ltd. Badenerstrasse 569 8048 Zurich Switzerland Tel. (01) 526350
United Kingdom	Mike Bennett Riva Turnkey Computer Systems Caroline House 125 Bradshawgate Bolton Lancashire United Kingdom Tel. (0204) 384112

Although every effort is made to ensure the accuracy of the data presented in the **Communicator,** Hewlett-Packard cannot assume liability for the information contained herein.

Prices quoted apply only in U.S.A. If outside the U.S., contact your local sales and service office for prices in your country.

Printed in U.S.A. 6/82

Part No. 5951-6111
V5#5