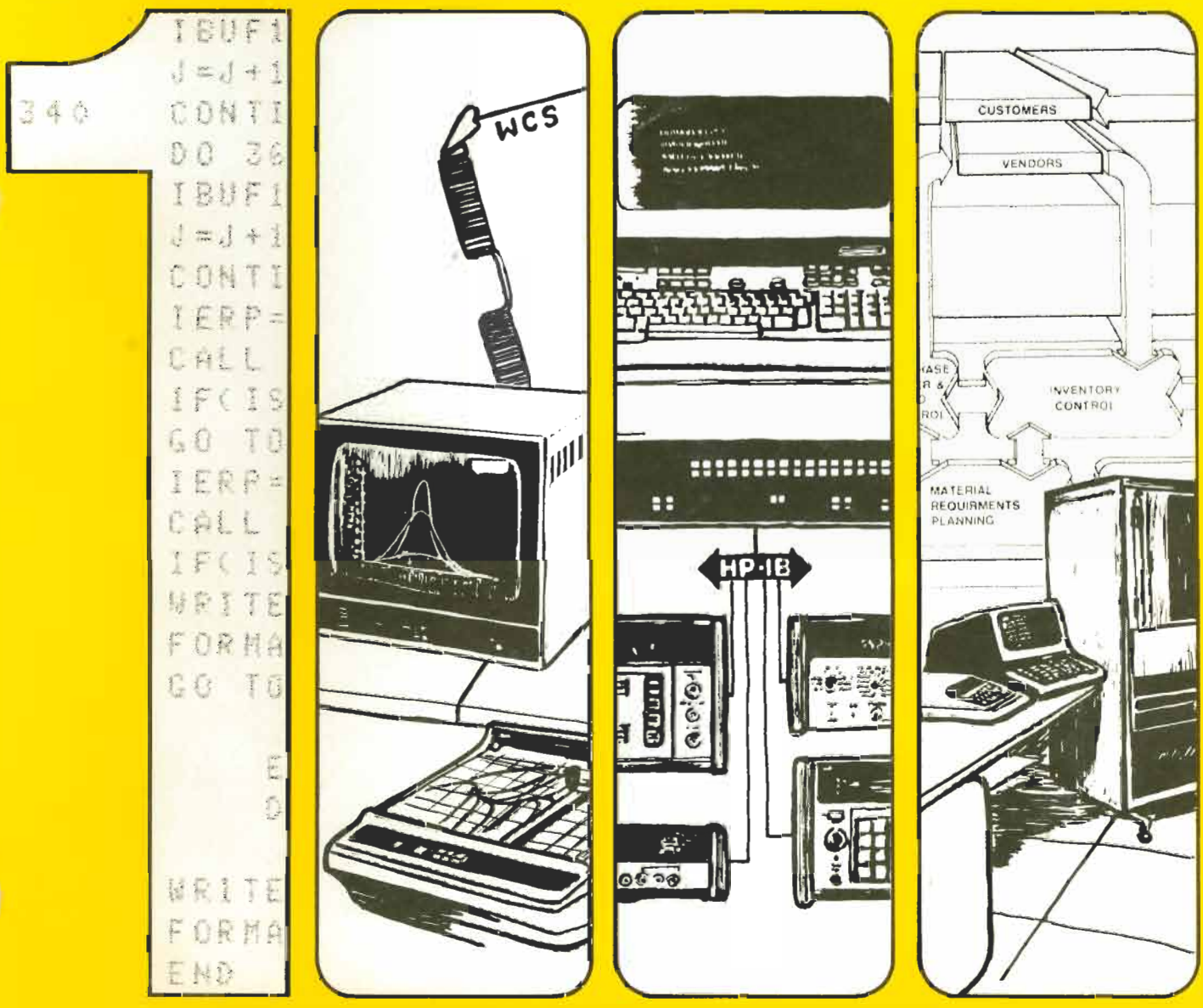
HEWLETT PACKARD

Hewlett-Packard Computer Systems

COMMUNICATOR



1981

Volume V Issue 3

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

HEWLETT-PACKARD
COMPUTER SYSTEMS

Volume V
Issue 3

COMMUNICATOR/1000

Feature Articles

Departments

ABOUT THIS ISSUE

I am pleased to let you know that this third issue of 1981 has a good turnout of customer contributions. My thanks to all of you out there who are supporting the Communicator/1000. We still have a need for articles for future issues; perhaps this issue will stimulate more of our readers to share some of their ideas.

Before telling you specifics of Volume 5, Issue 3 I'd like to refer you to a change of policy regarding calculator awards. The decision has been made to no longer exclude HP's Data System Division Technical Marketing Department employees from consideration. Whereas this will have no impact on the competition between either the customer or field employee judgments, I wanted to call your attention to this change. We feel that as long as impartiality can be achieved, this new policy will be in effect. In addition, we have added the category Languages to our list of topics.

Another subject I'd like to make a note of is regarding requests for back issues of the Communicator/1000. Our method of publication is to handle techincal editing, typesetting and graphics within Hewlett Packard. Printing however is done by an outside vendor and is limited to a specific number of issues. Distribution is then done by an HP distribution center. This precludes therefore individual requests for reprints of earlier issues. For copies of specific articles, I suggest you contact your local sales office — they should have a library of Communicators.

Now about our articles — our first article is in the area of Operating Systems. This feature is a team effort from the Institute of Clinical Physiology, University of Pisa. The subject is data acquisition in a real-time environment without operating system overhead.

Our second article is from one our more prolific writers, John Pezzano from the HP El Paso, Texas office. John's latest contribution is in the category Languages and is about FORTRAN Common. You will also find a second contribution from John in the Bit Bucket section. Thanks for such strong support, John!

We have two articles in the area of Operations Management. The first is by Dan Laskowski, a Systems Engineer in the HP Indianapolis, Indiana office on "MOVER", a file moving program. The second article is another useful concept, "Customized Hello Files". Thanks to Don McLaren from Martin Marietta in Orlando, Florida.

Our calculator award this time in the category HP field employees goes to Dan Laskowski. John Pezzano was not eligible for the competition as he has already received an award in 1981. Our award for best feature article by an HP customer goes to the group effort from University of Pisa. Congratulations!

Best feature article by an HP Customer:	**"User Written I/O Routines for HP 1000 Computer"** M. Varanini University of Pisa
Best feature article by an HP Field Employee:	**"MOVER: A File Moving Program"** Dan Laskowski HP Indianapolis, Indiana

Until next time . . .

Best Regards,

The 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

CORRECTION TO PREVIOUS ISSUE

Volume V, Issue One of the Communicator/1000 included a letter to the Editor by Charles L. Elliot of Elliot Geophysical Company. The included program, SCAN, contained errors when published. Following is a corrected version of the program. The Editor apologizes for any inconvenience that may have been caused by these mistakes.

```
FTN4,L
C***********
C#0176 SCAN
C***********
      PROGRAM SCAN(3),791121 C.L. ELLIOT
C     ADAPTED FROM LOCUS PROGRAM 22683-90017
C     THIS PROGRAM EMULATES A HARD COPY DEVICE BY
C     READING THE CONTENTS OF THE 26XX MEMORY
C     AND WRITING THAT DAT TO LUPR
C     DEFAULT IF LUPR=6
C     RU,SCAN,LUPR WILL COY FROM CURSOR POSITION
C     AND ENDS AT ^^^^ OR :RU,SCAN
      DIMENSION IBUF(50),IPAR(5),IREG(2)
      EQUIVALENCE (REG,IREG(1),IA),(IREG)(2),IB)
C     GET TERMINAL LOGICAL UNIT NUMBER AND PARAMETERS
      CALL RMPAR(IPAR)
      LU=LOGLU(LU)
      LUN=LU
      IOUT=IPAR(1)
      IF(IPAR(1).EQ.LU)IOUT=6
C     DEFINE CONTROL FUNCTIONS
      IDC1=10537B
      NLESC=33B
      IDUN=62137B
      IUP=40537B
      IEND=57136B
    5 FORMAT(2A2)
C     PUT TERMINAL IN BLOCK MODE
      WRITE(LU,5)IDC1
      WRITE(LU,5)NLESC,IUP
C     REQUST 1 LINE FROM TERMINAL
   20 WRITE(LU,5)NLESC,IDUN
C     READ THE LINE
      IBUF(1)=2H
      REG= EXEC(1,LUN,IBUF,50)
      LEN=IB
C     CHECK FOR END -- ^^^^ OR :RU,SCAN
      IF(LEN.GT.6) GO TO 50
      IF((IBUF(1).EQ.IEND).AND.(IBUF(2).EQ.IEND))GOTO99
      IF((IBUF(1).EQ.2H:R).AND.(IBUF(2).EQ.2HU,)
     $.AND.(IBUF(3).EQ.2HSC).AND.(IBUF(4).EQ.2HAN))GO TO 99
   12 FORMAT(1H ,50A2)
C     WRITE THE LINE TO THE LINE PRINTER
   50 WRITE(IOUT,12)(IBUF(JJ),JJ=1,LEN)
      GO TO 20
C     EJECT PAGE AND END
   99 ICNWD=IOR(1100B,IOUT)
      CALL EXEC(3,ICNWD,-1)
  999 END
      END$
```

HOW TO FIND SYSTEM STATUS WITHOUT REALLY LOGGING ON

John Pezzano/HP El Paso

Recently, a customer asked me why we don't provide a capability for the user to find out how active a Session Monitor System is without going through the effort of Logging On. In other words, how can a person run WHZAT without having to type in his/her user name, group name, and password? Wouldn't it also be nice if the system manager could quickly examine the programs and/or partitions?

There is an easy way to do these things. When you log on to session, you provide a USER.GROUP/PASSWORD. The user name must be provided but the others are not necessary. The group name is defaulted to GENERAL and the password to NONE. The system manager can create a user with user name WHZAT with defaulted group and password. If the capability level is set to 1 (one) and a "HELLO" file

```
:WH,AL
:EX
```

is set up, anyone can examine the system by typing WHZAT in response to a LOGON message. The result is that the user is logged on as WHZAT.GENERAL, the "HELLO" file is executed (running WHZAT with the "ALL" option), and the user is immediately logged off. Similarly, "WHZATPA" can, with

```
:WH,PA
:EX
```

find partition status. Voila! One can find system status without "really" logging on.

QUICK DATA BASE CHANGES

Camilla Foppes/Fairchild Test Systems

Listed below is an interactive FORTRAN utility used to standardize ASCII entries in an Image data base and to eliminate missed data due to abbreviations, misspellings, etc., during either a Query search or applications program string comparison. 'UPDAT' prompts the user for the name of the data base, level, data set item, old data string (to be replaced), and new data string. The user has the option to update more than one data item within a data base, more than one data base or to exit the program before any changes are made.

```
Compilation:   (usual HP FTN4 manner)

    :RU,FTN4,&UPDAT,6,%UPDAT

Loader sequence:

    :RU,LOADR
    /LOADR: SZ,14
    /LOADR: RE,%UPDAT
    /LOADR: END
```

Note

'UPDAT' must be sized 1 to 2 pages larger than required — therefore the 'SZ,14' in the LOADR sequence.

Programmer: Camilla Foppes
Fairchild Test Systems Group
MS 32/756
1601 Technology Dr.
San Jose, California 95115
(408) 998-0123 ext. 2873

```
**************************************************************************
FTN4,L
      PROGRAM UPDAT(),IMAGE UPDATE USER'S PROGRAM** 6-5-81 CF REV 3
C
C     IMAGE UPDATE USER PROGRAM
C     PROGRAMMER:  CAMILLA FOPPES
C                  FAIRCHILD TEST SYSTEMS
C     DATE:        6 JUNE 1981
C
C     THIS PROGRAM CAN BE USED TO UPDATE DATA ITEMS IN A GIVEN DATA SET.
C     OLD DATA WILL BE REPLACED WITH THE NEW DATA THAT IS INPUT THROUGH
C     THIS PROGRAM.  USER MUST INPUT INFORMATION SUCH AS LEVEL,DATA BASE
C     NAME, DATA ITEM TO BE CHANGED PLUS OLD AND NEW DATA.
C
C     IBASE = DATA BASE NAME
C     LEVEL = ACCESS LEVEL
C     MODES = MODE 1, 2, 3, 4, 8
C     LISTO = OLD DATA STRING
C     LISTN = NEW DATA STRING
C
C
C
      INTEGER IBASE(8),LEVEL(4),STAT(10),LIST(3),ID(3)
      INTEGER IANS(1),LALL(1),IARG(2)
      INTEGER LISTO(36),LISTN(36),OBUF(36),OBUF1(36)
C
      EQUIVALENCE (STAT(2),ILEN)
C
      DATA MODE1/1/
      DATA MODE2/2/
      DATA MODE3/3/
      DATA MODE4/4/
      DATA MODE8/8/
      DATA IBLANK/2H  /
      DATA LALL/2H@ /
      DATA IBASE(1)/2H  /
C
C
10    WRITE (1,20)
20    FORMAT ("TO EXIT THIS PROGRAM AT ANY TIME, TYPE 'EX'."/
     1"PLEASE ENTER YOUR LEVEL:  ")
      READ (1,30) (LEVEL(I),I=1,3)
30    FORMAT (3A2)
      IF (LEVEL(1).EQ.2HEX) GOTO 999
C
C     GET DATA BASE NAME (IBASE)
C
      WRITE (1,40)
40    FORMAT ("ENTER NAME OF DATA BASE IN THE FOLLOWING FORMAT: "/
     1"DATA BASE:SECURITY CODE:CRN;  ")
      READ (1,50) (IBASE(I),I=2,8)
50    FORMAT (8A2)
      IF (IBASE(1).EQ.2HEX) GOTO 950
C
C     OPEN DATA BASE
C
      CALL DBOPN(IBASE,LEVEL,MODE1,STAT)
      IF (STAT(1).NE.0) GOTO 800
C
C     LOCK DATA BASE
```

```
C
      CALL DBLCK(IBASE,ID,MODE1,STAT)
      IF (STAT(1).NE.0) GOTO 860
C
C
C     GET DATA SET NAME (ID)
C
      WRITE (1,60)
60    FORMAT ("PLEASE ENTER DATA SET ID:  ")
      READ (1,70) (ID(I),I=1,3)
70    FORMAT (3A2)
      IF (ID(1).EQ.2HEX) GOTO 950
C
C     GET DATA ITEM (LIST)
C
80    WRITE (1,90)
90    FORMAT ("PLEASE ENTER DATA ITEM TO BE CHANGED:  ")
      READ (1,100) (LIST(I),I=1,3)
100   FORMAT (3A2)
      IF (LIST(1).EQ.2HEX) GOTO 950
C
C     GET OLD DATA STRING (LISTO)
C
      WRITE (1,110)
110   FORMAT ("PLEASE ENTER OLD DATA STRING TO BE REPLACED:  ")
      READ (1,120) (LISTO(I),I=1,36)
120   FORMAT (36A2)
      IF (LISTO(1).EQ.2HEX) GOTO 950
C
C     GET NEW DATA STRING (LISTN)
C
      WRITE (1,130)
130   FORMAT ("PLEASE ENTER NEW DATA STRING:  ")
      READ (1,140) (LISTN(I), I=1,36)
140   FORMAT (36A2)
      IF (LISTN(1).EQ.2HEX) GOTO 950
C
C
C     POINT TO FIRST RECORD
C
150   CONTINUE
      IARG(1)=0
      IARG(2)=0
160   CALL DBGET(IBASE,ID,MODE4,STAT,LIST,OBUF,IARG)
      IF (STAT(1).NE.0) GOTO 820
C
C     FETCH NEXT DATA ENTRY
C
170   CALL DBGET(IBASE,ID,MODE2,STAT,LIST,OBUF,0)
      IF (STAT(1).NE.0) GOTO 820
C
180   WRITE(1,190) (OBUF(I),I=1,36)
190   FORMAT("READING  - ",36A2)
C
C     COMPARE DATA FOR REPLACEMENT
C
      IF (JSCOM(OBUF,1,STAT(2),LISTO,1,IERR)) 170,200,170
C
C     UPDATE DATA BASE
```

```
C
200   CONTINUE
      CALL DBUPD(IBASE,ID,MODE1,STAT,LIST,LISTN)
      IF (STAT(1).NE.0) GOTO 840
      GOTO 170
C
210   CONTINUE
      CALL DBUNL(IBASE,ID,MODE1,STAT)
      IF (STAT(1).NE.0) GOTO 880
      GOTO 950
C
C     CONTINUATION ROUTINE
C
700   CONTINUE
      WRITE (1,701)
701   FORMAT ("DO YOU WISH TO UPDATE MORE ITEMS?  Y/N ")
      READ (1,702) IANS
702   FORMAT (1A2)
      IF (IANS.EQ.2HY ) GOTO 760
      WRITE (1,703)
703   FORMAT ("DO YOU WISH TO UPDATE ANOTHER DATA SET?  Y/N ")
      READ (1,704) IANS
704   FORMAT (1A2)
      IF (IANS.EQ.2HY ) GOTO 750
      GOTO 900
C
750   CONTINUE
      WRITE (1,751)
751   FORMAT ("UNLOCKING OLD DATA BASE ")
C
C     UNLOCK OLD DATA BASE
C
      CALL DBUNL(IBASE,ID,MODE1,STAT)
      IF (STAT(1).NE.0) GOTO 880
C
C     CLOSE OLD DATA BASE
C
      WRITE (1,752)
752   FORMAT ("CLOSING OLD DATA BASE")
C
      CALL DBCLS(IBASE,ID,MODE1,STAT)
      IF (STAT(1).NE.0) GOTO 999
      GOTO 10
C
C     BLANK OUT OLD BUFFERS
C
760   CONTINUE
      DO 777 I=1,36
      LISTO(I)=IBLANK
      LISTN(I)=IBLANK
 777  CONTINUE
      GOTO 80
C
C     ERROR MESSAGES
C
800   WRITE (1,810) STAT(1)
810   FORMAT ("SYSTEM ERROR = ",I5," ON OPEN")
      GOTO 999
```

```
C
820   IF (STAT(1).EQ.12) GOTO 700
      WRITE (1,830) STAT(1)
830   FORMAT ("SYSTEM ERROR = ",I5," ON GET")
      GOTO 900
840   CONTINUE
      IF (STAT(1).EQ.112) 841,850
841   WRITE (1,842) STAT(1)
842   FORMAT ("SYSTEM ERROR = ",I5,"ON UPDATE."/"ATTEMPT WAS MADE TO
     1 ALTER THE VALUE OF A KEY ITEM."/"PLEASE TRY ANOTHER DATA ITEM."/)
      GOTO 700
850   WRITE (1,851) STAT(1)
851   FORMAT ("SYSTEM ERROR = ",I5," ON UPDATE")
      GOTO 900
860   WRITE (1,870) STAT(1)
870   FORMAT ("SYSTEM ERROR = ",I5," ON LOCK")
      GOTO 950
880   WRITE (1,890) STAT(1)
890   FORMAT ("SYSTEM ERROR = ",I5," ON UNLOCK")
      GOTO 950
C
C     CLOSE DATA BASE AND END PROGRAM
C
900   CALL DBUNL (IBASE,ID,MODE1,STAT)
      IF(STAT(1).NE.0) GOTO 880
C
950   CALL DBCLS(IBASE,ID,MODE1,STAT)
      IF (STAT(1).EQ.0) GOTO 999
      WRITE (1,960)
960   FORMAT ("SYSTEM ERROR = ",I5," ON CLOSE")
999   CONTINUE
      WRITE (1,1000)
1000  FORMAT ("END OF UPDATE PROGRAM.  BYE.")
      END
*************************************************************************
```

DATE RANGING

R. Arthur Gentry/American Tel & Tel

A problem facing most Data-Base programmers, is how to easily range dates. An EXEC (11) call will give you the Julian date and year, but this requires several statements to test if the date you are testing is, for example Dec. 20 and your range is Dec. 10 to Jan 5. GDATE eliminates this problem by setting each date to its chronological number since the begining of time. Now Dec. 20, 1980 becomes 723169 and the range dates become 723159 and 723185 respectively, making ranging a snap!

This subroutine is based on the Gregorian Calendar. The Gregorian Calendar defines a leap year of 366 days as occuring whenever the year is divisible by 4 except for centesimal years (years ending in 00) which are leap years only if divisible by 400. All other years are common years consisting of 365 days. The extra day in a leap year is added on by giving Feb. 29 days. Sept., April, June & Nov. have 30 days & all the other months have 31 days.

Dates are expressed in the normal manner — Month,Day & Year — or in terms of the "Equivalent Gregorian Day" (EGD). The EGD is defined as the number of elapsed days from the beginning of the calendar (1/1/1) to a given Month, Day & Year. For example, 12/31/1973 equals 720623 EGD.

In my particular use, I store the Gregorian Date in my Image/1000 Data-Base records for fast and easy date testing. It takes up a lot less room than placing an ASCII 12/20/1980 and is easier to handle than 354,1980.

Call: `CALL GDATE (OPT,MON,DAY,YEAR,DWK,DYR,DMON,EGD,*)`

All arguments are Integer*2 except EGD which is Integer*4 and have the following meanings:

OPT = 1	The caller supplies the Month, Day & Year and the other arguments are computed and returned.
= 2	The caller supllies the EGD and the other arguments are computed and returned.
MON =	The month (1-12) DAY
DAY =	The day of the month (1-31)
YEAR=	The year (1-up)
DWK =	The day in the week (where 1-7 = Sunday-Saturday)
DYR =	The day in the year (1-366 — Julian Date)
DMON=	The total days in the month (MON)
EGD =	The Equivalent Gregorian Day (1-up)
* =	*nnn Where nnn Equals a statement No. to GO TO on an error

If valid date information is passed by the caller, return is made to the next statement following the CALL statement. If invalid date information is passed by the caller, return is made to *nnn.

The subroutine's use is illustrated by the following:

1. Given a date, the day of the week is determined.
2. A date may be tested to be in a report by seeing if its GREG. day falls within the range of the two Gregorian Days that define the report period.
3. Cyclical date functions may be performed by utilizing the Gregorian day function (e.g., — defining dates of recurring 45 day periods, Biorythm Analysis, etc.)
4. If Feb. 29, (year) is offered to the subroutine and the return arguments indicate that the date is valid, then the year is a leap year.
5. Perpetual calendars may be generated.

This subroutine is based on the adoption of the Gregorian Calendar on 9/14/1752, which is still in effect. All dates from 9/14/1752 forward will be valid until the calendar is changed again. All date info prior to 9/14/1752 has been extrapolated to 1/1/1 based on the various calendar systems in effect prior to 9/14/1752.

Note

This subroutine will work in systems that do not have Fortran 4X by changing the INTEGER*4 to REAL and eliminating the error return.

Also, the LYEAR Function can be used by any program to test for Leap Years, and will be accurate until they change the calendar again.

```
FTN4X,L
C
$TITLE TRUE GEGORIAN DATE ROUTINE ISS. 2  801126  R.A.G.(MWR)
   SUBROUTINE GDATE (OPT,MON,DAY,YEAR,DWK,DYR,DMON,EGD,*)
  >, GREGORIAN DATE ROUTINE  801126
C
   IMPLICIT INTEGER*2 (A-Z)
C
C
C
C*********************************************************************
C
C REVISION LIST
C
C --DATE-- --BY-- -- D E S C R I P T I O N --
C
C 03/19/80 R.A.G. -ORIGINAL ISSUE
C 11/26/80 R.A.G. -CONVERTED REAL NUMBERS TO DOUBLE INTEGER AND ADDED
C                  DAY OF THE WEEK, DAY OF THE YEAR(JULIAN DATE),
C                  AND NUMBER OF DAYS IN THE MONTH.  ALSO ADDED AN
C                  ERROR RETURN ADDRESS.
C
C*********************************************************************
C
   INTEGER*2 M(12)
C
   INTEGER*4 EGD,EEGD,GDATS
C
   LOGICAL LYEAR
C
   DATA M/31,28,31,30,31,30,31,31,30,31,30,31/
C
C INITIALIZE SOME VARIABLES
C
   M(2)=28
   DYR=0
C
C CHECK FOR OPTION
C                     1 = CALLER SUPPLIES MONTH DAY YEAR
C                    >1 = CALLER SUPPLIES GREGORIAN DATE
C
   IF (OPT.GT.1) GO TO 120
C
C THIS SECTION CONVERTS TO A GREGORIAN DATE
C
C TEST ARGUMENTS FOR VALIDITY
C
   IF (MON.LT.1.OR.MON.GT.12.OR.DAY.LT.1.OR.DAY.GT.31.
  >    OR.YEAR.LT.1) GO TO 180
C
   IF (LYEAR(YEAR)) M(2)=29             ! TEST FOR LEAP YEAR
C
   IF (DAY.GT.M(MON)) GO TO 180         ! DAY > NO. DAYS IN MON ??
C
C CALCULATE GREG. DATE TO 1st OF REQUESTED YEAR
C
   Y=YEAR-1
   EGD=GDATS(Y)
C
C CALCULATE TO CURRENT GREGORIAN DATE
```

```
C
   J=MON-1
   IF (J.EQ.0) GO TO 110
   DO 100 I=1,J
   DYR=DYR+M(I)
100   CONTINUE
   EGD=EGD+DYR
110   EGD=EGD+DAY
   DYR=DYR+DAY
   GO TO 170
C
C THIS SECTION CONVERTS FROM A GREGORIAN DATE
C
120   IF (EGD.LT.1) GO TO 180              ! ARGUMENT OK ??
C
C CALCULATE CURRENT DATE (MM/DD/YYYY)
C
   YEAR=(EGD/366)-1
130   YEAR=YEAR+1
   EEGD=GDATS(YEAR)
   IF (EGD-EEGD-368) 140,140,130
140   YEAR=YEAR+1
   DYR=EGD-EEGD
C
   IF (LYEAR(YEAR)) M(2)=29             ! LEAP YEAR ??
C
   DO 150 MON=1,12
   EEGD=EEGD+M(MON)
   IF (EGD.LE.EEGD) GO TO 160
150   CONTINUE
   M(2)=28
   GO TO 140
C
C CALCULATE THE REMAINING ARGUMENTS
C
160   DAY=EGD+M(MON)-EEGD
170   DMON=M(MON)
   DWK=MOD(EGD,7)+1
   RETURN
C
180   RETURN 1                             ! ERROR RETURN
   END
C
C
$TITLE FUNCTION LYEAR (LEAP YEAR) ISS. 1  801126  R.A.G.(MWR)
   FUNCTION LYEAR(YEAR), LEAP YEAR TESTER 801126
C
   IMPLICIT INTEGER*2 (A-Z)
C
C THIS FUNCTION WILL TEST A GIVEN YEAR AND RETURN A TRUE/FALSE
C   INDICATION.
```


```
C
C=====================================================================
C
C REVISION LIST
C
C --DATE-- --BY-- -- D E S C R I P T I O N --
C
C 11/26/80 R.A.G. -ORIGINAL ISSUE
C
C=====================================================================
C
   LYEAR=0
   IF (MOD(YEAR,4).EQ.0.AND.MOD(YEAR,100).NE.0.OR.
  >    MOD(YEAR,400).EQ.0) LYEAR= -1
   RETURN
   END
C
C
$TITLE FUNCTION GDATS (GDATE)     ISS. 1  801126  R.A.G.(MWR)
   FUNCTION GDATS(YEAR), GDATE FUNCTION 801126
C
   IMPLICIT INTEGER*2 (A-Z)
C
   INTEGER*4 GDATS
C
C
C THIS FUNCTION IS PART OF THE GDATE SUBROUTINE, AND RETURNS A
C   GREGORIAN DATE TO THE 1ST OF THE YEAR BASED ON THAT YEAR.
C
C=====================================================================
C
C REVISION LIST
C
C --DATE-- --BY-- -- D E S C R I P T I O N --
C
C 11/26/80 R.A.G. -ORIGINAL ISSUE
C
C=====================================================================
C
C ** N O T E **
C
C GDATS MUST BE DECLARED AS AN INTEGER*4 FUNCTION !!!!
C
   GDATS=365*YEAR
   GDATS=GDATS+(24*(YEAR/100))
   GDATS=GDATS+(YEAR/400)
   GDATS=GDATS+(MOD(YEAR,100)/4)
   RETURN
   END
$
```

TEST PROGRAM FOR GDATE

```
FTN4X
   PROGRAM DATE
   IMPLICIT INTEGER*2 (A-Z)
C
C THIS IS A PROGRAM TO TEST THE SUBROUTINE "GDATE".
C
   INTEGER*4 EGD
C
100   M=0
   D=0
   Y=0
   DWK=0
   DYR=0
   DMON=0
   EGD=0
   WRITE (1,'(" ENTER 1 FOR CALENDER DATE, 2 FOR EGD, 3 TO END _")')
   READ (1,*) ANS
   IF (ANS.GT.2) GO TO 910
   IF (ANS.EQ.1) GO TO 110
   WRITE (1,'(" ENTER EGD _")')
   READ (1,*) EGD
   OPT=2
105   CALL GDATE (OPT,M,D,Y,DWK,DYR,DMON,EGD,*900)
   GO TO 120
110   WRITE (1,'(" ENTER M,D,Y _")')
   READ (1,*) M,D,Y
   OPT=1
   GO TO 105
120   WRITE (1,1000) OPT,M,D,Y,DWK,DYR,DMON,EGD
1000  FORMAT (' OPT=',I2,/,' DATE= ',I2.2,'/',I2.2,'/',I4.4,/,
  >         ' DAY OF WEEK= ',I2,/,' DAY OF YEAR= ',I4,/,
  >         ' DAYS IN MONTH= ',I3,/,
  >         ' GREGORIAN DATE= ',I10,/)
   GO TO 100
900   WRITE (1,'(" ERROR IN CALL TO GDATE")')
910   STOP
   END
```

PACKING LU 'S 2 AND 3 DURING THE OFF HOURS.

Wayne P. Reidinger/Bell Laboratories

There's nothing more aggravating on RTE-IV than to attempt to save a file or program on LU 2 or 3 only to receive a FMGR-033 error (not enough room on cartridge). Packing the appropriate disc LU will usually gain back the needed space, but if it's LU 2, the packing can't take place until all active ID-segments have been OFF'ed. If you're the only user, no problem, but if there are other users, they probably would not appreciate having their programs OFF'ed just to allow you to save your file or program. One answer to this problem is to have a time-scheduled program that would do the offing and packing during the off-hours thereby avoiding any inconvenience to system users. Presented here is a program (actually 2 programs and one transfer file) that does just this. Three modules are required for this job:

1. Program PACK2 — This is the main program that is put in the time list. PACK2 kills the FMGR and then reschedules FMGR to execute the transfer file #PACK2.

2. #PACK2 — The transfer file that OFF's the ID-segment names returned by program IDSEG. This transfer file also issues the pack command and may be edited to include RP's to restore vital ID-segments after the disc packing has been performed.

3. Program IDSEG — Gets the name of a temporary ID-segment from the system table and returns it to #PACK2. If all temporary segments have been OFF'ed then IDSEG will return $$$$$$ for the name. IDSEG will NOT return it's own ID-segment name. Transfer file #PACK2 will off IDSEG just before packing begins.

Program PACK2

```
FTN4
C.....THIS PROGRAM TERMINATES THE FILE MANAGER (FMGR), THEN
C.....RESCHEDULES FMGR AND PASSES TO IT THE NAME OF THE XFER
C.....FILE #PACK2 WHICH WILL PACK LU 2.
      PROGRAM PACK2
      DIMENSION IBUFA(5)
      INTEGER BUFR(7),FMGR(3)
      DATA BUFR/2H:T,2HR,,2H#P,2HAC,2HK2,2H::,2H3 /
      DATA IBUFA/2HOF,2H,F,2HMG,2HR,,2H1 /
      DATA FMGR/2HFM,2HGR,2H  /
      DATA IBUFL/-13/
C.....FIRST OFF THE CURRENT FMGR
      ICOUN=9
      I=MESSS(IBUFA,ICOUN,1)
C
C.....NOW SET UP ARGUMENTS TO RESCHEDULE FMGR AND PASS
C.....THE NAME OF THE XFER FILE TO BE EXECUTED.
      ICODE=23
      IASC=2HXX
      LIST=1
      ISV=0
      LOG=1
      IDUM=0
C.....SCHEDULE FMGR AND XFER TO #PACK2
      CALL EXEC(ICODE,FMGR,0,LIST,ISV,LOG,IDUM,BUFR,IBUFL)
      END
```

Transfer File #PACK2

```
:SV,0
:* THIS TRANSFER FILE ATTEMPTS TO PACK LU-2.
:* BEFORE DOING SO, HOWEVER, PROGRAM IDSEG IS CALLED
:* TO DETERMINE IF THERE ARE ANY PGMS THAT HAVE TO
:* BE OFF'D. IF PGM IDSEG FINDS A TEMP ID SEGMENT,
:* IT RETURNS THE NAME IN 10G (1P,2P,3P) WHERE IT CAN
:* OFF'D BY THE TRANSFER FILE. IF ALL PGMS HAVE BEEN
:* OFF'D , THEN IDSEG RETURNS WITH $$$$$$ IN 10G -
:* INDICATING THAT PACKING MAY BEGIN.
:* IT IS RECOMMENDED THAT THIS XFER FILE BE TIME-SCHEDULED
:* TO BEGIN EXECUTION OUTSIDE OF WORKING HOURS TO AVOID
:* INCONVENIENCE TO SYSTEM USERS.
:RU,IDSEG
:IF,1P,NE,9252,6
:IF,2P,NE,9252,5
:IF,3P,NE,9252,3
:OF,IDSEG
:PK,2
:PK,3
:IF,1,EQ,1,2
:OF,10G
:IF,1,EQ,1,-10
:* WHEN DONE REINITIALIZE TIME FOR DISC PACKING
:RP,PACK2
:SYIT,PACK2,4,24,3,00
:SYON,PACK2::3
:TR
```

Program IDSEG

```
FTN4
      PROGRAM IDSEG(3,70),   GET ID SEGMENT
      DIMENSION IGLOBL(5)
      DATA IGLOBL/2H$$,2H$$,2H$$/
C
C.....GET ADDR OF KEYWORD BLOCK
C
      KEYWD=IGET(1657B)
C.....SEARCH ID SEGMENTS AND CHECK FOR TEMP ONES.
C
20    IDSEG=IGET(KEYWD)
C
C.....CHECK FOR END OF KEYWD BLOCK
C
      IF(IDSEG.NE.0)GO TO 21
C.....RETURN WITH 10G =$$$$$$
      IGLOBL(1)=2H$$
      IGLOBL(2)=2H$$
      IGLOBL(3)=2H$$
      CALL PRTN(IGLOBL)
      CALL EXEC(6)
21    CONTINUE
```

```
C
C      RECORD NAME
C
       IGLOBL(1)=IGET(IDSEG+12)
       IGLOBL(2)=IGET(IDSEG+13)
       IGLOBL(3)=IGET(IDSEG+14)
C.....SKIP BLANK OR PERM. ID SEG'S
       IDBLNK=IAND(IGLOBL(1),177400B)
       IF(IDBLNK.NE.0)GO TO 400
C.....COUNT BLANKS AND CONTINUE
22     KEYWD=KEYWD+1
       GO TO 20
C.....IF TEMP MAKE THIRD WORD PRINTABLE.
C.....SKIP OVER PERM. ID SEG'S
400    ITEM=IAND(IGLOBL(3),200B)
       IF(ITEM.EQ.0)GO TO 22
       IGLOBL(3)=IOR(IAND(IGLOBL(3),177400B),40B)
C.....CHECK FOR ID SEG OF THIS PGM.
       IF(IGLOBL(1).NE.2HID)GO TO 404
       IF(IGLOBL(2).NE.2HSE)GO TO 404
       IF(IGLOBL(3).NE.2HG )GO TO 404
       GO TO 22
404    CALL PRTN(IGLOBL)
       CALL EXEC(6)
       END
```

Program Execution

To put PACK2 into the system time list to execute at 3 A.M., enter the following commands (assuming we're running under FMGR):

```
:RP,PACK2
:SYIT,PACK2,4,24,3,00
:SYON,PACK2
```

Don't forget it will be necessary to reenter these commands whenever the system is rebooted since the time list will be lost.

USER WRITTEN I/O ROUTINES FOR HP 1000 COMPUTER

M. Varanini, A. Macerata, P. Pisani,
and C. Marchesi/University of Pisa

BACKGROUND

There are some fields of computer applications where it is useful to work without the operating system overhead. In research laboratories where the computer is extensively used to process data and to develop new programs in a very broad range of problems, the enhanced capabilities of operating systems are appreciated; on the other hand, other applications are required which are not well supported by standard procedures. We refer to high speed acquisition or, more in general, to the control of all devices by completely user written programs. Solutions suggested by manuals are not always adequate. For example, to program continuous real time acquisition by the Class I/O Exec approach (Ref.1), the operating system requires an unavoidable time interval to control the process. Also the privileged interrupt method needs an extra copy of the I/O cards plugged in privileged slots. The advantages of an "outside operating system" technique are: no duplications of I/O cards and, simple, well known techniques, grown using previous HP operating systems (DOS,RTEII) (Ref. 2). The main results of this technique are: a very fast response to interrupts, friendly implementation, easy updating, compatibility with DOS, RTEII program structure. The disadvantages are: the loss of multiprogramming capability during program run time (relevant only when some system resources are left) and the impossibility to access standard I/O drivers through EXEC calls.

In the next pages we describe three ways to operate outside the operating system. The first one uses the Skip on Flag technique. The second one uses the two DCPC channels. In both examples the interrupt system is left disabled. The third one is the most general and flexible approach and it allows multiinterrupt processing. The example given refers to a case of two interrupts.

Let us summarize, for reader convenience, the main features of HP1000 computers that are involved in these techniques to process flags and interrupts. These computers can host up to 2 Megabytes memory, but they cannot address more than 32 pages (1k words per page) of memory at a time (that is the maximum fifteen bit address). To address all available pages, the computer uses a group of four maps, made of 32 registers each. The maps are associated to the system, to the user, to the first DCPC channel and to the second DCPC channel. Each map points to its particular 32 pages of memory. In every instant the computer sees only 32 pages. It is the duty of the operating system to call the system map or the user map and to change dynamically the map contents to satisfy the requests of many users working in a multiprogramming configuration. Also the two DCPC maps are changed dynamically by the operating system, according to the system or user requests. When the system is turned off by the $LIBR subroutine, the configuration of the four maps is frozen. The system map contains the 32 pages associated to the system, the user map contains the pages of that user who killed the system, while the DCPC maps contents depend on the last request. Furthermore the system and the user base page are different because of a fence dividing it in two parts. Only one is common to the system and to the user. Normally the part containing the trap cells belongs to the system and it cannot be addressed directly from the user pages. To overcome this limitation, it is necessary to employ the DMS instructions, which link the two maps and control the contents of all four maps. If it is considered preferable to avoid the use of most of the DMS instructions, the same procedure can be applied over the base page shared between system and user by changing the base page fence.

The following examples have to be considered as samples of the techniques used; they are working, although shortened for space saving.

SAMPLE PROGRAMS

The programs run on a HP21MX-F computer with a RTEIVB Operating System. The I/O configuration is:

- HP Time Base Generator
- HP 91000 A/D Converter
- HP Magnetic Tape Interface
- HP 12966 Terminal Interface

All programs use the system subroutines $LIBR and $LIBX to disable and enable the interrupt system and the memory protect option.

Skip on Flag Technique

This example is given through a funny program that displays running seconds on the switch register.

```
ASMB
*
* Subroutine to display seconds on switch register.
* Counting last 30 seconds.
*
       NAM TIME,7
       ENT TIME
       EXT $LIBR,$LIBX
TIME   NOP
       JSB $LIBR      Disable interrupt system
       NOP              and memory protect.
       CLC CTIME      Turn off Time Base Generator (TBG).
       LDA =B4        TBG control word = 1 second
       OTA CTIME      Output to TBG
       STC CTIME,C    Turn on TBG
       CLA            Clear A register
       LDB M30        B register = complemented time counter
CT1    SFS CTIME      Wait flag
       JMP *-1          of TBG
       OTA 1B         Flag arrived. Output the time counter on switch reg.
       INA            Increment time counter
       CLF CTIME      Clear TBG flag
       INB,SZB        Increment complemented time counter; test if zero
       JMP CT1        No, jump to CT1
       LDA =B2        Yes, it is zero. Restore
       OTA CTIME        time period to 10 msec.
       STC CTIME,C    Turn on TBG
       JSB $LIBX      Turn on
       DEF *+1          interrupt system
       DEF *+1            and memory protect.
       JMP TIME,I     Return
CTIME  EQU 11B        Select code of TBG
M30    DEC -30        Time set value
       END TIME
```

Direct Memory access technique (DCPC)

This subroutine is designed to sample an analog signal and store it on mag tape. It uses the double buffer technique to obtain a continuous acquisition. The converter is paced. The first DCPC channel is used to transfer data from converter to memory. The second DCPC channel transfers data from memory to mag tape. The flag of the first channel signals the filling up of an acquisition buffer. We suppose that the time to transfer a buffer on mag tape by the DCPC is less than the time necessary for an acquisition. An error condition takes care of speed incompatibility. The program through the DMS instructions assigns the user map to both the DCPC maps (two DCPC channels are used).

```
ASMB
*
*  Subroutine to sample, store and transfer on mag tape a continuous
*  analog signal.
*
       NAM CANVI,7
       ENT CANVI
       EXT .ENTR,$LIBR,$LIBX
C      NOP             Address of converter channel number
V1     NOP                "    "  first buffer
V2     NOP                "    "  second buffer
NSAMP  NOP                "    "  number of sample per record
NREC   NOP                "    "  number of mag tape record
ERR    NOP                "    "  error cell
*
CANVI  NOP
       JSB .ENTR       Load first external parameter address
       DEF C           Address of first word of parameter buffer
       LDA NSAMP,I
       CMA             Complemented
       STA L             buffer length
       LDA NREC,I      Prepare
       CMA               negative counter
       STA NRES            for records
       CLA             Clear
       STA ERR,I         error cell
       JSB $LIBR       Disable interrupt system
       NOP               and memory protect
       CLC MV          Turn off converter
       LDA =B140001    Control word to normalize converter
       OTA MV
       STC MV,C        Normalize start
       NOP
       STC MV
       SFS MV          Wait
       JMP *-1           flag
       CLC MV,C        Yes,flag arrived. Turn off converter
       CLC 00,C        Disable all devices
       LDA MAPA        Control word to save
       IOR =B100000      Port A map
       PAA             Save Port A map
       LDA MAPB        Control word to save
       IOR =B100000      Port B map
       PBA             Save Port B map
       LDA =B100000    Transfer user map to
       XMA               Port A map
```

```
LDA =B100001  Transfer user map to
XMA             Port B map
LDA C,I       Prepare control word for converter:
ALS             differential input
IOR CONT        digitize mode, paced.
OTA MV
STF MV
LDA V1        A register = address of buffer for DMA acquisition.
JSB INDMA     Subroutine call to acquire with DMA (first time).
WAIT  SFS DMAH2     Wait the end of
      JMP *-1         the first buffer acquisition.
      CLC DMAL2,C   When the DCPC flag is on,
      CLC DMAH2,C     clear the control.
      SFS DMAH1     Test if transfer on mag tape has finished.
      JMP ERROR     No, go to ERROR routine.
      ISZ NRES      Yes, increment record counter.
      JMP ANCO1     Acquisition is not terminated. Continue.
      JMP DECOF     Acquisition is terminated. Go to deconfiguration routine.
ANCO1 LDA V2        A register = address of buffer for DMA acquisition.
      JSB INDMA     Subroutine call to acquire with DMA.
      JSB MTWRI     Subroutine call to store on mag tape.
      DEF *+3
      DEF V1,I      Address of buffer to store
      DEF L         Buffer length.
WAIT2 SFS DMAH2     Wait until the end of
      JMP *-1         the second buffer acquisition.
      CLC DMAL2,C   When the DCPC flag is on,
      CLC DMAH2,C     clear the control.
      SFS DMAH1     Test if transfer on mag tape has finished.
      JMP ERROR     No. Go to ERROR routine.
      ISZ NRES      Yes. Increment record counter.
      JMP ANCO2     Acquisition is not terminated. Continue.
      JMP DECOF     Yes. Go to deconfiguration routine.
ANCO2 LDA V1        A register = address of buffer for DMA acqisition.
      JSB INDMA     Subroutine call to acquire with DMA.
      JSB MTWRI     Subroutine call to store on mag tape.
      DEF *+3
      DEF V2,I      Address of buffer to store
      DEF L         Buffer length.
      JMP WAIT      End of the acquisition of the second buffer.
*
DMACW OCT 120000    DMA control word
DMAL1 EQU 2B        DMA channel 1
DMAH1 EQU 6B         "     "    1
DMAL2 EQU 3B         "     "    2
DMAH2 EQU 7B         "     "    2
M15   OCT 100000
VSS   NOP
*
* Subroutine for acquisition with channel 2 DMA.
* Input: A register = address of the acquisition buffer
```

```
*
INDMA NOP
      STA VSS        Save buffer address
      CLC DMAL2,C    Disable DMA channel
      CLC DMAH2,C      "     "     "
      LDA DMACW      DMA control word
      IOR INMV         for input data.
      OTA DMAH2
      CLC DMAL2
      LDA VSS        Prepare
      IOR M15          DMA
      OTA DMAL2          channel
      STC DMAL2            for
      LDA L                  acquisition
      OTA DMAL2                of N samples
      STC MV,C       Converter start
      STC DMAH2,C    DMA start
      JMP INDMA,I    Return
*
* Error routine
*
ERROR CLA,INA        If DMA mag tape transfer time > DMA acquisition time
      STA ERR        then ERR cell = 1
*
DECOF CLC MV,C       Routine to deconfigure used devices & to restore status
      CLC DMAL1,C    Disable DMA channel 1
      CLC DMAH1,C      "     "     "     1
      CLC DMAL2,C      "     "     "     2
      CLC DMAH2,C      "     "     "     2
      CLC MTC,C      Disable mag tape control channel
      CLC MTD,C        "     "    "  data channel
      LDA MAPA       Restore
      PAA              DCPC map A
      LDA MAPB       Restore
      PBA              DCPC map B
      JSB $LIBX      Restore
      DEF *+1          interrupt system
      DEF *+1            and memory protect.
      JMP CANVI,I    Exit from CANVI subroutine.
*
NFCB  NOP
NRES  NOP
MSKK  OCT 3400
MASK  OCT 10
L     NOP
CONT  OCT 130000
CONTC NOP
SW    NOP
MAPA  DEF *+1
      BSS 32
MAPB  DEF *+1
      BSS 32
MTD   EQU 14B
MTC   EQU 15B
MV    EQU 25B
INCA  DEF MTD
INMV  DEF MV
```

```
*
* Subroutine to write on mag tape. It uses the DMA channel 2.
* Input: Address of data buffer
*        Buffer length
*
PARA   BSS 2
MTWRI  NOP
       JSB .ENTR      Input parameter
       DEF PARA         address.
       CLC MTC,C      Disable mag tape data channel.
       CLC MTD,C        "     "    "  control channel.
       LDA PWC        Prepare
       ADA INCA         DMA control word.
       OTA DMAH1
       CLC DMAL1
       LDA PARA       Prepare
       OTA DMAL1        DMA
       STC DMAL1          channel
       LDA PARA+1,I         to store a buffer
       OTA DMAL1              on mag tape.
       LDB WCC        Test
RIP    OTB MTC          on
       LIA MTC            magnetic
       AND MASK             tape.
       SZA
       JMP RIP        Tape busy. Try again.
       STC MTC,C      Tape ready. Record
       STC MTD,C        on mag tape.
       STC DMAH1,C    DMA start.
       JMP MTWRI,I
*
WCC    OCT 31
PWC    OCT 20000
       END
```

Interrupt process technique

This subroutine acquires a N-sized sample buffer and writes it on a terminal. The sampling rate is controlled by the Time Base Generator. User written routines are used to process the interrupts from Time Base Generator and Converter. The interrupts can be processed by altering the trap cells and using short routines which have to be in the physical base page. Of course cells not used by the system must be chosen. They link the system pages to user pages. The interrupt processing routine has a special structure. It is divided in two parts. The first one, stored in the base page, is executed when the corresponding interrupt occurs (every interrupt automatically enables the system pages). Its duty is only to link the trap cells to the interrupt service routines loaded in the user page. We point out that the next example uses the DMS instructions to communicate with the system pages. The same result can be obtained by changing the fence of the system base page so the base page is available to the user and can be addressed with no-DMS instructions.

```
ASMB
*
* Subroutine to acquire and store N samples using a technique which
* switches off the operating system and processes interrupts in a
* user written way.
* The used devices are:
* HP Time Base Generator
* HP 91000 A/D Converter
*
      NAM SBAC,7
      ENT SBAC
      EXT .ENTR,$LIBR,$LIBX
ITB   NOP
IA    NOP
N     NOP
SBAC  NOP
      JSB .ENTR
      DEF ITB
      LDA N,I
      CMA,INA
      STA CONT
      JSB $LIBR      Turn off interrupt system
      NOP              and memory protect.
      CLC 13B        Turn off terminal 1
      CLC 17B          "   "      "     2
      CLC 20B          "   "      "     3
      XLA 11B        Save
      STA SAV11        trap cells
      XLA 25B            using
      STA SAV25            the system map.
      LDA =B114040   Modify
      XSA 11B          trap cell
      LDA =B114050       using
      XSA 25B              the system map.
      LDA INDTM      Move
      LDB =B40         a string
      LDX =B4            of words
      MWI                  from
      LDA INDCV              user's map
      LDB =B50                 to
      LDX =B4                    system
      MWI                          map.
```

```
      LDA ITB,I     Set Time Base Generator
      OTA 11B         to selected time.
      STC 11B,C     Turn on Time Base Generator.
      STF 00        Turn on interrupt system.
      LDA CONT      Loop
      SZA             to wait
      JMP *-2           the end of acquisition.
      CLF 00        End of acquisition. Turn off interrupt system.
      CLC 11B       Clear control of Time Base Generator.
      CLC 25B,C     Clear control of A/D converter.
      LDA SAV11     Restore
      XSA 11B         trap
      LDA SAV25         cells.
      XSA 25B
      LDA =B2       Restore Time Base Generator
      OTA 11B         to 10 msec.
      STC 11B,C     Turn on Time Base Generator.
      STC 13B       Turn on terminal 1
      STC 17B         "   "    "     2
      STC 20B         "   "    "     3
      JSB $LIBX     Turn on interrupt system and memory protect.
      DEF SBAC      End of subr. Return to the main program.
*
* Timer interrupt process routine
*
TIME  CLF 11B       Clear flag
      STA SV1A      Save A register
      LDA =B100000  Output A/D
      OTA 25B         control word.
      XLA 41B       Take the return address
      STA RTTM        from own link routine.
      LDA SV1A      Restore A register.
      STC 25B,C     Turn on A/D converter.
      JMP RTTM,I    Return.
*
* A/D converter interrupt process routine
*
CONV  CLF 25B       Clear flag.
      STA SV2A      Save A register.
      LIA 25B       Load data from converter.
      ARS,ARS       Normalize
      ARS,ARS         data.
      STA IA,I      Store data in the buffer.
      ISZ IA        Increment the address of buffer.
      ISZ CONT      Increment the sample counter.
      NOP
      XLA 51B       Take the return address
      STA RTCV        from own link routine.
      LDA SV2A      Restore A register.
      JMP RTCV,I    Return
CONT  NOP
SAV11 NOP
SAV25 NOP
SV1A  NOP
SV2A  NOP
RTTM  NOP
RTCV  NOP
INDTM DEF SYTM
INDCV DEF SYCV
```

```
*
* Link routine of timer; it is moved in the base page of the
* system. When an interrupt occurs, the system map is enabled,
* this routine is executed and it links the timer interrupt
* process routine in the user's map.
*
SYTM  OCT 41
      NOP
      UJP TIME      Enable user's map and jump.
*
* Link routine of A/D converter.  It is moved as in timer routine.
*
SYCV  OCT 51
      NOP
      UJP CONV      Enable user's map and jump.
      END
```

References

1. MULTI-TERMINAL ACCESS TO A REAL-TIME DATA ACQUISITION SYSTEM
   Bradley Ward/Consultant, Atlanta, Ga.
   Communicator/1000, Vol.IV, Issue 3

2. HP 1000 F-Series Computer
   Operating and Reference Manual

ABOUT FORTRAN COMMON

John Pezzano/HP El Paso

INTRODUCTION

The HP 1000 computer supports many different options for common variables in FORTRAN. Each of these has its own uses and advantages. To get the most out of your system, it is important that you, the user, understand the features, benefits and limitations of each one.

TWO BASIC KINDS

As far as the FORTRAN program itself is concerned, there are two different kinds of COMMON. UNLABELED COMMON is declared by naming each COMMON block such as:

```
COMMON X, Y, Z(100)
```

All routines accessing any one of these variables must declare them in the same manner or provide dummy placeholders that agree in size with the original declaration. For example, if one subroutine needed to access the array Z and variable I, it could declare: COMMON DUMMY (2), Z(100), I. DUMMY would then replace both "X" and "Y". Variables after "I" are not needed and could be ignored.

Unlike unlabelled Common, BLOCK COMMON (or LABELLED COMMON) is accessable to other routines by the label name. For example, instead of the previous COMMON statement, we could use in the main program:

```
COMMON/DATA1/X,Y
COMMON/DATA2/Z(100), I
COMMON/DATA3/J,K(100)
```

Now our subroutine need only declare:

```
COMMON/DATA2/Z(100), I
```

without having to declare dummy placeholders for the other COMMON variables. This, of course, is far easier. But then, why use unlabelled COMMON?

WHY USE COMMON?

Since most FORTRAN programmers know and understand about COMMON there is no need to go into details. But to summarize, COMMON permits programs and subprograms to share "COMMON" areas of memory by declaring these areas in the affected routines. COMMON eliminates the need to pass variables to subroutines and is very useful in the HP 1000 for segment-to-segment and program-to-program communication.

THE HP 1000 AND COMMON

Each type of COMMON is handled differently in an HP 1000. Unlabelled COMMON is treated as true COMMON in the HP 1000. On the other hand, BLOCK COMMON is treated as external variables (and/or arrays). Unlabelled COMMON cannot be initialized with DATA statements but needs no special declarations. BLOCK COMMON can be initialized with DATA statements but requires a separate BLOCK DATA declaration. Since BLOCK COMMON is actually external variables, loading programs without the BLOCK DATA routines results in an undefined external error by the loader. The external will be the BLOCK name (not the variable name).

In unlabelled COMMON, it is only important that variable positions line up.

For example:

```
COMMON X, Y, Z(100)
```

and

```
COMMON I, J, K(2), X(100)
```

in two different routines are acceptable if X, Y, are real and I, J, & K are single integer. I & J (one word each) together make up X (2 words long). K and Y both are the same size and array Z and X are equivalent. Note, however, that putting something in I, J, or K, will destroy the integrity of X or Y. In BLOCK COMMON, the BLOCK name is critical. It is this name, not the variables in it that will be the external name. However, the preceding information about unlabelled COMMON is also true within any BLOCK.

WHERE IS COMMON LOCATED?

One of the nicest features of the HP 1000 is its ability to load COMMON in different places. Again, each has its own advantages. Unlabelled COMMON can be loaded in three areas. The usual (and default) location is to load the COMMON with the program itself (local COMMON). The COMMON is then put in front of the program. If the program is swapped, so is the COMMON. This means that COMMON is shareable only by the program, its subroutines, and its segments. Since COMMON is located at the front of the program (that is just the way the loader was designed) the routine declaring the largest COMMON must be loaded first. Otherwise, a loader error results. A program can also be loaded with SYSTEM COMMON. This means that the COMMON variables will be put in the system area and will not be swapped with the program. This also permits multiple programs to share COMMON variables and not just within a single progam. There are two problems with SYSTEM COMMON: (1) At generation time, sufficient SYSTEM COMMON must be allocated to hold the entire COMMON area. COMMON variables cannot be split between local and system areas. The system area cannot be changed in size except by a new system generation. (2) All programs declaring SYSTEM COMMON share the same area. This means that your cooperating programs may share SYSTEM COMMON while someone else's programs may also want to use it. Remember, SYSTEM COMMON is a shared resource with no user protection.

There is a second area of SYSTEM COMMON called Reverse Common. To a background program (default load), SYSTEM COMMON is Background Common and Reverse Common is REAL TIME COMMON. To a Real Time Program SYSTEM COMMON is Real Time COMMON and Reverse COMMON is Background COMMON. Thus, two background programs can communicate with each other if they are both loaded with System (Background) COMMON or Reverse (Real Time) COMMON.

A Real Time and Background program can communicate if the Background program is loaded with System (Background) COMMON, and the Real Time program is loaded with Reverse COMMON or vice versa. Note that rules (1) and (2) above apply except programs sharing a Real Time COMMON will not interfere with programs sharing Background COMMON and vice versa, except as described later. (Confusing, isn't it?) HP programs DO NOT USE either types of COMMON. They are reserved strictly for the user. There are also choices for BLOCK COMMON. Again the default is to load COMMON with the program.

Unlike unlabelled COMMON, BLOCK COMMON can be loaded anywhere in the program area and there are no restrictions as to which routine is loaded first. For segmented programs, the BLOCK COMMON must be loaded with the main if it is to be shared by multiple segments or if its integrity must be maintained when a new segment is loaded. If the COMMON IS ONLY TO BE USED BY A SEGMENT and is only to be used by a segment and its subroutines, it is not needed when the segment is overlaid; it can be loaded with that segment. BLOCK COMMON can also be loaded into the SYSTEM. If it is, it is put into the Subsystem Global Area (SSGA) which is RTE's Labelled COMMON area. Like System and Reverse COMMON, SSGA area must be allocated at generation and cannot be increased in size. However, unlike them, the allocation is not done by declaring a size but by putting the COMMON modules themselves into SSGA at generation. This consequently considerably reduces the usefulness of SSGA to the user. Many of HP subsystems make use of SSGA. The System Manager must declare the Real Time and Background COMMON sizes of generation. However, RTE-IVB likes to see all areas end on page boundaries. Therefore, because SSGA exists in all systems there will almost always be Background COMMON. The default size being the end of SSGA to the end of the next page boundary.

WHAT KIND OF COMMON SHOULD I USE?

Now that you realize all the COMMON options available, what should you pick? Your choice depends upon your intentions. If you intend to use the COMMON only with the program, either type of COMMON will do. BLOCK COMMON will probably be preferable because of the convenience of only declaring what is necessary in each routine and the fewer loading restrictions.

System and Reverse Common are useful for inter-program communication (it is faster than CLASS I/O since no data need be moved between programs), for programs that need to maintain their data even if swapped, and for use when the data must be in the system map (which is required for some user written privileged drivers).

SSGA, because of its generation requirements, should be restricted to use only when absolutely necessary. Since the variables in the SSGA are accessed by name, they are especially useful for keeping special variables that must be maintained for use between programs or if a program is reloaded or restarted. For example, a class or resource number or disc file name could be kept in SSGA.

WHAT ABOUT PROTECTION?

Unlabelled COMMON loaded locally is located at the beginning of a program in memory. It is protected from other programs by the system. The system and Reverse Common have two limitations. Multiple sets of programs can use the areas and RTE will not care if they are overwriting each other's variables. In addition, the protection is according to a priority. Programs loaded with Background Common cannot have their area overwritten by programs with local or no COMMON. They cannot overwrite the Real-Time area or SSGA. Programs with Real-Time Common are protected from Background or Common programs but because of the way that the system protects areas, blowing a COMMON array can cause Background Common to be overwritten. Similarly, SSGA programs can overwrite other SSGA areas or Real Time or Background Common.

HOW ABOUT ASSEMBLY ROUTINES?

Assembly routines can declare unlabelled COMMON with the "COM" pseudo instruction. This is equivalent to the FORTRAN unlabelled COMMON statement. BLOCK COMMON is accessed by the pseudo instruction "EXT name" where name is the BLOCK name. Variables in the BLOCK are equivalent to the BLOCK name with offset. For example, in FORTRAN, COMMON/DATA1/I, J, K would be written in assembly:

```
        EXT DATA1
I       EQU DATA1
J       EQU DATA1+1
K       EQU DATA1+2
```

SUMMARY

The HP 1000 is designed as a Real Time System primarily for scientific and engineering applications. Therefore, there is a lot of versatility even in such mundane things as COMMON declaration and loading to provide a broad application area for users. It also means, therefore, that users should understand and use COMMON to their advantage.

Summary Table of COMMON

NAME	TREATED BY RTE	LOADED TYPE	LOCATION	COMMAND REQUIRED	ADVANTAGE	ACCESS BY	PROTECTION	RECOMMENDED
Unlabelled COMMON	As COMMON	Local	Beginning of program space	OP, NC (Default)		Loaded program, Segments and Subroutine only	Protected from all other programs	Normal Use
		Back-ground	System area	OP, SC	Background Programs	Any program using same area	Protected from programs not using RT, BG and SSGA.	Program-to-Program Communication or special drivers
				OP, RC	Real Time programs			
		Real Time	System area	OP, RC OP, SC	BG Programs RT Programs	Any program using same area	Protected from programs not using RT or SSGA	Same as above
BLOCK COMMON	As External Variable	Local	Anywhere in program space	(Default) (REL, % COMMON file)		Loaded program Segments and Subroutines only	Protected from all other programs	Normal use
		System	SSGA	OP, SS		Programs declaring variables only	Protected from programs not using SSGA	Only when absolutely necessary

MOVER: A FILE MOVING PROGRAM

Dan Laskowski/HP Indianapolis

Have you ever needed to move a few files from one system to another? Did you just store the files to the cartridge and supply a list of the files on the cartridge? Or did you store a procedure file as the first file that could be used to restore the files? If any of this sounds familiar, read on.

I found myself doing file transfers often as an SE and got the technique down pretty well. I created both store and restore files and used the store file to build the tape with the restore file as the first file on the tape. When you got the tape you simply loaded in the first file, modified it as necessary, and executed it to load in the remaining files. But that took time and errors were easily made.

MOVER is a program I wrote that reads in a list of files to store on tape, and creates the tape complete with procedure files for restoring the files. The restore procedure files are unique in themselves because they do not need to be restored from the cassette/mag tape before use. They can be executed right from the tape! Let's see how that can work.

Using the cassette as an example, consider three files stored on the cassette. The file contents are as follows:

```
File 1

0001 :ST,4,/RSTR
0002 :ST,4,/TEMP
0003 :ST,4,DUMMY
0004 :CN,4
0005 :PU,/TEMP
0006 :PU,/RSTR
0007 ::-2

File 2

0001 ::/RSTR
0002 : Dummy line

File 3

0001 Dummy data file!
```

With the cassette at the beginning, typing 'TR,4' in response to the FMGR prompt would execute the first command 'ST,4,/RSTR'. This would store the remainder of the first file in a file called /RSTR. When that file store is finished, the cassette is positioned at file two. The next command is ':/RSTR' which is a transfer control to the file /RSTR. Commands will now be executed from the first file, and the remainder of the restore will be controlled by the first file.

The first command in /RSTR is 'ST,4,/TEMP' which stores the remainder of file 2 into a dummy file called /TEMP. With the cassette now positioned at file three, stores of the data can be made for as many files as there are to move. The procedure file /RSTR then ends by purging itself and /TEMP. The ::−2 erases the transfer stack so that the cassette will not attempt to execute itself again.

The process is simple, straightforward, and can be modified to store a file to disc, compile the file, and then load and run the program if necessary. The original intent was to move files, and when moving files, three rules need to be obeyed.

1. File type and record format do not need to be specified in a store command if the source is a disc file. Security is necessary only if the file has negative security, and supplying disc CRN will speed up finding the file to be stored.

2. File type and record format do need to be specified in a store command if the source file is non-disc. The security will default to zero if not supplied, and the file will be placed on the first cartridge if a disc CRN is not specified.

3. If the file is type two, you must specify the record length.

File type, record format, and record length is not stored to tape. This makes it necessary to specify file type and record format when restoring files. In most cases, the files are ASCII source or binary relocatable, and specifying 'AS' or 'BR' as record formats defaults to file types 4 and 5. This is okay for most file moves, but if you are not careful, ignoring the file type can be disasterous.

MOVER is a program that writes procedure files using this process to move files. Additional features have been added for flexibility. MOVER can be run interactively, or from an answer file. The output can be to cassette or mag tape, and can be restored from any LU. The files can be restored with any security code, and to any disc cartridge. There is a purge and pack option for updating purposes.

MOVER first asks for the LU# to store the files to. Then, line by line, MOVER asks for files to be moved. MOVER takes the name, (security and cartridge reference can be specified), and opens the file. If the open is successful, MOVER gets the file type and writes three procedure files. The first procedure file will create the final tape and contains the store commands to move the files from disc to tape.

The second procedure file is the main file used to restore the files on the receiving system. It contains the store commands to move the files from tape to disc. Record formats in the store commands are assigned according to file type as follows:

Type	Record format
1,2	BN
3,4	AS
5	BR
6	BN
7	BA
8 or greater	BN

The last procedure file is an optional file that can purge the named files from disc and then pack the disc. This is useful in an update situation.

There are cautions when using MOVER. The restoring procedure files are standard FMGR procedure files and all rules concerning procedure files apply. MOVER uses FMGR when building the tape, so if you are running the 'real' FMGR, you will have to 'OF' it sometime in the process to finish creating the tape. Note: MOVER is not needed in the receiving system to restore the tapes. MOVER is only used to build the tapes! Image data bases need to be restored with the original security code and cartridge reference number to be useable. The last caution to using MOVER is that it is not as efficient in tape useage as routines like READT/WRITT, JSAVE/JRSTR, and/or READR/SAVER. These routines all use large buffers for reading and writing to tape. Large buffers cut down the quantity of inter-record gaps on the tape, with each inter-record gap taking roughly 1/2 or more inches of tape. Fewer inter-record gaps means more data per the same amount of tape, hence the larger buffers in READT and WRITT. MOVER uses FMGR store commands which means many inter-record gaps on the tape. MOVER was not designed to replace READT/WRITT, but rather to make easy the process of moving a few files from system to system.

The source code is written using the FTN4X compiler. To keep the partition requirements at their minimum, EXEC and FMP calls replace READ and WRITE calls. Extensive use of the IF-THEN-ELSE-ENDIF construct is used. The directions for using MOVER is in the source.

If you have any questions or suggestions, I can be found in the HP Indianapolis office.

```
FTN4,L
      PROGRAM MOVER(3,90),Tape transport program          *810506.0939*
C
C     This program will create a mag tape with built in
C     procedure files for restoring files saved on the tape.
C     Run the program and answer the questions.  Type /E as
C     the last file.  /A will abort MOVER.  MOVER will then
C     proceed to build the tape as necessary.  When the tape
C     is moved to the new system, to restore the files simply
C     type ':X,X,Y,Z,[PU or NOPU]' in response to FMGR where
C     'X' is the LU of the tape device and 'Y' is the security
C     you want, and 'Z' is the cartridge reference number of
C     where you want the files to go.  PU is an option to purge
C     the files before restoring them.  Purge will also pack
C     cartridge 'Z'.  You must specify purge to get a purge.
C     The only requirements on the receiving system is that
C     it have FMGR, and should restore on all versions of RTE
C     and RTE-L/XL.
C
C     To run mover simply type 'RU,MOVER' and answer the
C     questions.  If the answers are put into an answer file
C     in the form of:
C
C                 4        <== Tape device LU number
C                 NAMR1    <== File Namr number 1
C                 NAMR2    <== File Namr number 2
C                 .        .
C                 .        .
C                 .        .
C                 .        .
C                 NAMRn    <== File Namr number n
C                 /E       <== /E ends the list
C
C     then type 'RU,MOVER,FILE' to build the tape.
C
C     Dan Laskowski-HP-Indianapolis
```

```
C
      INTEGER DCB1(144),DCB2(144),DCB3(144),DCB4(144),DCB5(144)
      DIMENSION IBUF(128),NAME(10),LINE1(14),LINE2(14),LINE3(8)
      DIMENSION LINE4(20),LINE5(3),LINE6(7),LINE7(7),LINE8(5)
      INTEGER LINE9(33),COUNT(3)
      INTEGER ERROR,OPT,SEC(3),CRN(3),SIZE(2),A,B,TYPE(3),TEMP(3)
      INTEGER NMRBF(10),PARM(5),STRT,OU,PRSBF(33),LUBUF(10),LEN(3)
      DATA SIZE/12,0/,COUNT/0,0,0/
      DATA LINE1/':ST,1G,XXXXXX:2G:3G:XX:-1,XX'/
      DATA LINE2/':ST,XXXXXX:XXXXXX:XXXXXX, XX'/
      DATA LINE3/':PU,XXXXXX:2G:3G'/
      DATA LINE4/':ST,1G,XXXXXX:2G:3G:XX:-1:XXXXXX,BN'/
      DATA LINE5/':CN,XX'/
      DATA LINE6/':ST,/RSTR\, XX'/
      DATA LINE7/':ST,/PURG\, XX'/
      DATA LINE8/':CN,XX,EO '/
      DATA LINE9/' XXXXXX is a type 2 file with record length greater th
     1an 128 words'/
C
C     Get input info
C
      CALL RMPAR(PARM)
      CALL GETST(IBUF,20,LEN)
      STRT=1
      CALL NAMR(NMRBF,IBUF,2*LEN,STRT)
C
C     Set up input/output lu for questions and errors
C
      LU=1
      OU=1
      TYPE=IAND(NMRBF(4),3B)
      IF(TYPE.EQ.0.OR.TYPE.EQ.1) LU=PARM
      IF(IFTTY(LU).EQ.-1.AND.TYPE.NE.3) OU=PARM
C
C     Output rev code
C
      CALL REIO(2,OU,14H Mover 06May81,7)
C
C     Check if we need to open a namr for input to mover
C
      IF(TYPE.EQ.3) THEN
C
C     Need to open input file
C
          LU=0
          CALL OPEN(DCB5,ERROR,NMRBF(1),0,NMRBF(5),NMRBF(6))
          IF(ERROR.EQ.-6) THEN
              CALL REIO(2,OU,22H Input file not found!,11)
              GO TO 999
          ENDIF
          IF(ERROR.LT.0) GO TO 999
      ENDIF
C
C     Get the lu of the tape device
```

```
C
99    CALL REIO(2,OU+2000B,18H Enter tape Lu#:   ,-17)
      IF(LU.NE.0) THEN
          CALL SFILL(LUBUF,1,20,2H  )
          CALL REIO(1,LU+400B,LUBUF,10)
          CALL ABREG(A,B)
C
C         Read tape lu # from file
C
      ELSE
          CALL READF(DCB5,ERROR,LUBUF,20,B)
          IF(B.EQ.-1) GO TO 1000
          IF(ERROR.NE.0) GO TO 999
          CALL REIO(2,OU,LUBUF,B)
      ENDIF
C
C     Check if input is numeric
C
      CALL PARSE(LUBUF,2*B,PRSBF)
      IF(PRSBF.NE.1) THEN
C
C         We need a number here return and prompt again

C
          IF(LU.EQ.0) CALL REIO(2,OU,20H First line in file ,10)
          CALL REIO(2,OU,22H Needs to be a number!,11)
          IF(LU.EQ.0) GO TO 999
          GO TO 99
      ENDIF
C
C     Now convert to ascii and place in line buffers
C
      CALL CNUMD(PRSBF(2),LUBUF)
      LINE2(14)=LUBUF(3)
      LINE5(3)=LUBUF(3)
      LINE6(7)=LUBUF(3)
      LINE7(7)=LUBUF(3)
      LINE8(3)=LUBUF(3)
C
C     First create the files as necessary
C
10    CALL CREAT(DCB1,ERROR,6H/BLD\ ,SIZE,4)
      IERR=ERROR
      IF(IERR.EQ.-2) CALL PURGE(DCB1,ERROR,6H/BLD\ )
      IF(IERR.EQ.-2.AND.ERROR.NE.0) GO TO 999
      IF(IERR.EQ.-2) GO TO 10
      IF(ERROR.LT.0) GO TO 999
11    CALL CREAT(DCB2,ERROR,6H/RSTR\,SIZE,4)
      IERR=ERROR
      IF(IERR.EQ.-2) CALL PURGE(DCB2,ERROR,6H/RSTR\)
      IF(IERR.EQ.-2.AND.ERROR.NE.0) GO TO 999
      IF(IERR.EQ.-2) GO TO 11
      IF(ERROR.LT.0) GO TO 999
12    CALL CREAT(DCB3,ERROR,6H/PURG\,SIZE,4)
      IERR=ERROR
      IF(IERR.EQ.-2) CALL PURGE(DCB3,ERROR,6H/PURG\)
      IF(IERR.EQ.-2.AND.ERROR.NE.0) GO TO 999
      IF(IERR.EQ.-2) GO TO 12
      IF(ERROR.LT.0) GO TO 999
```

```
C
C       Now start /PURG\ file
C
        CALL WRITF(DCB3,ERROR,8H::/RSTR\,4)
        IF(ERROR.NE.0) GO TO 999
        CALL WRITF(DCB3,ERROR,14H:IF,4G,EQ,PU,1,7)
        IF(ERROR.NE.0) GO TO 999
        CALL WRITF(DCB3,ERROR,2H::,1)
        IF(ERROR.NE.0) GO TO 999
        CALL WRITF(DCB3,ERROR,26H:DP, Purging these files! ,13)
        IF(ERROR.NE.0) GO TO 999
        CALL WRITF(DCB3,ERROR,10H:SV,0,,IH ,5)
        IF(ERROR.NE.0) GO TO 999
C
C       Now start /RSTR\ file
C
        CALL WRITF(DCB2,ERROR,10H:SV,2,9,IH,5)
        IF(ERROR.NE.0) GO TO 999
        CALL WRITF(DCB2,ERROR,4H:** ,2)
        IF(ERROR.NE.0) GO TO 999
        CALL WRITF(DCB2,ERROR,58H:**  To restore this tape first rewind th
       1e tape and type: ,29)
        IF(ERROR.NE.0) GO TO 999
        CALL WRITF(DCB2,ERROR,64H:**  TR,X,X,Y,Z,[PU or NOPU] where X is t
       1he LU of the tape unit,,32)
        IF(ERROR.NE.0) GO TO 999
        CALL WRITF(DCB2,ERROR,64H:**  Y is the Security for the files, Z i
       1s the CRN or -LU of the,32)
        IF(ERROR.NE.0) GO TO 999
        CALL WRITF(DCB2,ERROR,62H:**  disc and PU or NOPU is an option to
       1first purge the files,31)
        IF(ERROR.NE.0) GO TO 999
        CALL WRITF(DCB2,ERROR,24H:**  and pack the disc. ,12)
        IF(ERROR.NE.0) GO TO 999
        CALL WRITF(DCB2,ERROR,4H:** ,2)
        IF(ERROR.NE.0) GO TO 999
        CALL WRITF(DCB2,ERROR,18H:DP, Mover 06May81,9)
        IF(ERROR.NE.0) GO TO 999
        CALL WRITF(DCB2,ERROR,14H:ST,1G,/RSTR\ ,7)
        IF(ERROR.NE.0) GO TO 999
        CALL WRITF(DCB2,ERROR,14H:ST,1G,/PURG\ ,7)
        IF(ERROR.NE.0) GO TO 999
        CALL WRITF(DCB2,ERROR,8H::/PURG\,4)
        IF(ERROR.NE.0) GO TO 999
        CALL WRITF(DCB2,ERROR,28H:DP, Restoring these files! ,14)
        IF(ERROR.NE.0) GO TO 999
        CALL WRITF(DCB2,ERROR,6H:SV,0 ,3)
        IF(ERROR.NE.0) GO TO 999
```

```
C
C       Now start the /BLD\ file
C
        CALL WRITF(DCB1,ERROR,10H:SV,2,,IH ,5)
        IF(ERROR.NE.0) GO TO 999
        CALL WRITF(DCB1,ERROR,LINE5,3)
        IF(ERROR.NE.0) GO TO 999
        CALL WRITF(DCB1,ERROR,LINE6,7)
        IF(ERROR.NE.0) GO TO 999
        CALL WRITF(DCB1,ERROR,LINE7,7)
        IF(ERROR.NE.0) GO TO 999
        CALL WRITF(DCB1,ERROR,26H:DP, Storing these files! ,13)
        IF(ERROR.NE.0) GO TO 999
        CALL WRITF(DCB1,ERROR,10H:SV,0,,IH ,5)
        IF(ERROR.NE.0) GO TO 999
C
C       Check if break bit is set
C
100     IF(IFBRK(I).LT.0) GO TO 994
C
C       If from lu get filename
C
        CALL REIO(2,2000B+0U,18H Enter filename:  ,-17)
        IF(LU.NE.0) THEN
            CALL REIO(1,400B+LU,IBUF,20)
            CALL ABREG(A,B)
        ELSE
C
C           If from disc file read filename
C
            CALL READF(DCB5,ERROR,IBUF,20,B)
            IF(B.EQ.-1) GO TO 1000
            IF(ERROR.NE.0) GO TO 999
            CALL REIO(2,0U,IBUF,B)
        ENDIF
C
C       If zero length record, then end
C
        IF(B.EQ.0) GO TO 1000
C
C       Parse into a namr buffer
C
        STRT=1
        CALL NAMR(NAME,IBUF,2*B,STRT)
C
C       Check if /E to end of /A to abort
C
        IF(NAME.EQ.2H/E.AND.NAME(2).EQ.2H  .AND.NAME(3).EQ.2H  ) GOTO 1000
        IF(NAME.EQ.2H/A.AND.NAME(2).EQ.2H  .AND.NAME(3).EQ.2H  ) GOTO 1003
C
C       Open the file to get file type
C
        CALL OPEN(DCB4,ERROR,NAME,0,NAME(5),NAME(6))
        IF(ERROR.EQ.-6) THEN
C
C           File not found.  Try again!
```

```
C
          CALL REIO(2,OU,22H File does not exist! ,11)
          GO TO 100
       ELSE IF(ERROR.EQ.-7) THEN
C
C         File has read security.  Try again!
C
          CALL REIO(2,OU,22H Check file security! ,11)
          GO TO 100
       ELSE IF(ERROR.EQ.-8) THEN
C
C         File locked.  Try again!
C
          CALL REIO(2,OU,22H File is locked open! ,11)
          GO TO 100
       ELSE IF(ERROR.EQ.-32) THEN
C
C         CRN not found. Try again!
C
          CALL REIO(2,OU,26H That disc is not mounted!,13)
          GO TO 100
       ELSE IF(ERROR.LT.0) THEN
C
C         Some other Fmgr error.  Abort!
C
          GO TO 999
       ENDIF
       TYPE=ERROR
C
C      If type 2, get record length
C
       IF(TYPE.EQ.2) CALL READF(DCB4,ERROR,IBUF,128,LEN)
       IF(TYPE.EQ.2.AND.ERROR.NE.0) GO TO 999
C
C      Check if CRN given was null.  If null, get CRN from file
C
       IF(IAND(NAME(4),60B).EQ.0) THEN
          CALL LOCF(DCB4,ERROR,IREC,IRB,IOFF,JSEC,JLU)
       ENDIF
C
C      Close the file and continue
C
       CALL CLOSE(DCB4,ERROR)
       IF(ERROR.NE.0) GO TO 999
       IF(TYPE.EQ.2.AND.LEN.GT.128) THEN
C
C         Type 2 files with record lengths cannot be moved with FMGR
C
          CALL SMOVE(NAME,1,6,LINF9,2)
          CALL REIO(2,OU,LINE9,33)
          CALL REIO(2,OU,28H FMGR cannot move this file!,14)
          GO TO 100
       ENDIF
```


```
C
C     Increment count for count of files moved
C
      COUNT=COUNT+1
C
C     Build sec in ascii
C
      SEC=NAME(5)
      TEMP=(IAND(NAME(4),14B))/4
      IF(TEMP.EQ.3) THEN
C
C         Sec is already ascii--fill last four characters with blanks
C
          CALL SFILL(SEC,3,6,2H  )
      ELSE IF(TEMP.EQ.0) THEN
C
C         Sec was null--fill all six characters with blanks
C
          CALL SFILL(SEC,1,6,2H  )
C
C         Sec was numeric.  Break out into ascii
C
      ELSE IF(TEMP.EQ.1) THEN
          IF(NAME(5).LT.0) SEC=IXOR(NAME(5),177777B)+1
          CALL CNUMD(SEC,SEC)
          IF(NAME(5).LT.0) THEN
              DO 14 I=6,1,-1
              CALL SGET(SEC,I,J)
              IF(J.EQ.40B) THEN
                  CALL SPUT(SEC,I,2H -)
                  I=0
              ENDIF
14            CONTINUE
          ENDIF
      ENDIF
C
C     Build CRN in ascii
C
      CRN=NAME(6)
      TEMP=(IAND(NAME(4),60B))/16
      IF(TEMP.EQ.0) THEN
C
C         CRN was null--use negative JLU from the LOCF call
C
          NAME(6)=-JLU
          CRN=-JLU
          TEMP=1
      ENDIF
      IF(TEMP.EQ.3) THEN
C
C         CRN is already ascii--fill last four characters with blanks
C
          CALL SFILL(CRN,3,6,2H  )
      ELSE IF(TEMP.EQ.1) THEN
C
C     The CRN was numeric,  break out into ascii
```

```
C
          IF(NAME(6).LT.0) CRN=IXOR(NAME(6),177777B)+1
          CALL CNUMD(CRN,CRN)
          IF(NAME(6).LT.0) THEN
              DO 13 I=6,1,-1
              CALL SGET(CRN,I,J)
              IF(J.EQ.40B) THEN
                  CALL SPUT(CRN,I,2H -)
                  I=0
              ENDIF
13            CONTINUE
          ENDIF
      ENDIF
C
C     Now update /BLD\
C
      CALL SMOVE(NAME,1,6,LINE2,5)
      CALL SMOVE(SEC,1,6,LINE2,12)
      CALL SMOVE(CRN,1,6,LINE2,19)
      CALL WRITF(DCB1,ERROR,LINE2,14)
      IF(ERROR.NE.0) GO TO 999
C
C     Now update /RSTR\
C     First check if file to be moved is type 2.
C
      IF(TYPE.NE.2) THEN
C
C         File is not type 2.  Record size is not needed.
C
          LINE1(14)=2HBN
          IF(TYPE.EQ.3.OR.TYPE.EQ.4) LINE1(14)=2HAS
          IF(TYPE.EQ.5) LINE1(14)=2HBR
          IF(TYPE.EQ.7) LINE1(14)=2HBA
          CALL CNUMD(TYPE,TYPE)
          CALL SMOVE(NAME,1,6,LINE1,8)
          CALL SMOVE(TYPE,5,6,LINE1,21)
          CALL WRITF(DCB2,ERROR,LINE1,14)
          IF(ERROR.NE.0) GO TO 999
      ELSE
C
C         Type two files need record size passed also
C
          CALL CNUMD(LEN,LEN)
          CALL CNUMD(TYPE,TYPE)
          CALL SMOVE(NAME,1,6,LINE4,8)
          CALL SMOVE(TYPE,5,6,LINE4,21)
          CALL SMOVE(LEN,1,6,LINE4,27)
          CALL WRITF(DCB2,ERROR,LINE4,18)
          IF(ERROR.NE.0) GO TO 999
      ENDIF
C
C     Now update /PURG\
C
      CALL SMOVE(NAME,1,6,LINE3,5)
      CALL WRITF(DCB3,ERROR,LINE3,8)
      IF(ERROR.NE.0) GO TO 999
```

```
C
C       Now go back for more
C
        GO TO 100
C
C       Close out /BLD\
C
1000    CALL WRITF(DCB1,ERROR,10H:SV,2,,IH ,5)
        IF(ERROR.NE.0) GO TO 999
        CALL WRITF(DCB1,ERROR,LINE8,5)
        IF(ERROR.NE.0) GO TO 999
        CALL WRITF(DCB1,ERROR,LINE5,3)
        IF(ERROR.NE.0) GO TO 999
        CALL WRITF(DCB1,ERROR,10H:PU,/RSTR\,5)
        IF(ERROR.NE.0) GO TO 999
        CALL WRITF(DCB1,ERROR,10H:PU,/PURG\,5)
        IF(ERROR.NE.0) GO TO 999
        CALL WRITF(DCB1,ERROR,10H:PU,/BLD\ ,5)
        IF(ERROR.NE.0) GO TO 999
        CALL WRITF(DCB1,ERROR,4H:EX ,2)
        IF(ERROR.NE.0) GO TO 999
C
C       Now close out /RSTR\
C
        CALL WRITF(DCB2,ERROR,10H:SV,2,,IH ,5)
        IF(ERROR.NE.0) GO TO 999
        CALL WRITF(DCB2,ERROR,6H:CN,1G,3)
        IF(ERROR.NE.0) GO TO 999
        CALL WRITF(DCB2,ERROR,10H:PU,/PURG\,5)
        IF(ERROR.NE.0) GO TO 999
        CALL WRITF(DCB2,ERROR,10H:PU,/RSTR\,5)
        IF(ERROR.NE.0) GO TO 999
        CALL WRITF(DCB2,ERROR,10H:DP,Done! ,5)
        IF(ERROR.NE.0) GO TO 999
        CALL WRITF(DCB2,ERROR,6H:SV,9G,3)
        IF(ERROR.NE.0) GO TO 999
        CALL WRITF(DCB2,ERROR,4H::-2,2)
        IF(ERROR.NE.0) GO TO 999
C
C       Now close out /PURG\
C
        CALL WRITF(DCB3,ERROR,6H:PK,3G,3)
        IF(ERROR.NE.0) GO TO 999
        CALL WRITF(DCB3,ERROR,10H:SV,2,,IH ,5)
        IF(ERROR.NE.0) GO TO 999
        CALL WRITF(DCB3,ERROR,2H::,1)
        IF(ERROR.NE.0) GO TO 999
```

```
C
C     Now close the files
C
      CALL CLOSE(DCB1,ERROR)
      IF(ERROR.NE.0) GO TO 999
      CALL CLOSE(DCB2,ERROR)
      IF(ERROR.NE.0) GO TO 999
      CALL CLOSE(DCB3,ERROR)
      IF(ERROR.NE.0) GO TO 999
      IF(LU.NE.0) GO TO 93
      CALL CLOSE(DCB5,ERROR)
      IF(ERROR.NE.0) GO TO 999
C
C     Check count for .GT. zero
C
93    IF(COUNT.EQ.0) CALL REIO(2,OU,18H No files to move!,9)
      IF(COUNT.EQ.0) GO TO 995
C
C     Schedule FMGR with /BLD\ to be run
C
      CALL EXEC(23,6HFMGR  ,2HXX,1,2,1,0,8H,,/BLD\ ,4)
C
C     Now end
C
      CALL REIO(2,OU+2000B,12H Tape built!,6)
      CALL CNUMD(COUNT,COUNT)
      CALL REIO(2,OU+2000B,COUNT,3)
      CALL REIO(2,OU,14H files moved. ,7)
      CALL EXEC(6)
C
C     Error return for IFBRK or count=0
C
994   CALL REIO(2,OU,18H Break requested! ,9)
995   IBUF=2H/A
      GO TO 1003
C
C     FMGR error exit handler
C
999   IF(ERROR.LT.0) J=IXOR(ERROR,177777B)+1
      CALL CNUMD(J,TEMP)
      IF(ERROR.LT.0) TEMP(2)=IOR(IAND(TEMP,377B),26400B)
      CALL EXEC(2,2000B+OU,10HFmgr error,5)
      CALL EXEC(2,OU,TEMP,3)
C
C     Also post the error to the SCB for help to find
C
      LUBUF(1)=2HFM
      LUBUF(2)=2HGR
      CALL SMOVE(TEMP,3,6,LUBUF,5)
      IF(ERROR.LT.0) CALL SPUT(LUBUF,5,2H -)
      IF(ERROR.GE.0) CALL SPUT(LUBUF,5,2H  )
      CALL PTERR(LUBUF)
```

```
C
C     Now close all the files
C
1003  CALL CLOSE(DCB1,ERROR)
      IF(ERROR.NE.0.AND.ERROR.NE.-11)
     1CALL REIO(2,OU,24H Problems closing /BLD\!,12)
      CALL PURGE(DCB1,ERROR,6H/BLD\ ,2HDL)
      IF(ERROR.NE.0.AND.ERROR.NE.-6)
     1CALL REIO(2,OU,24H Problems purging /BLD\!,12)
      CALL CLOSE(DCB2,ERROR)
      IF(ERROR.NE.0.AND.ERROR.NE.-11)
     1CALL REIO(2,OU,26H Problems closing /RSTR\! ,13)
      CALL PURGE(DCB2,ERROR,6H/RSTR\,2HDL)
      IF(ERROR.NE.0.AND.ERROR.NE.-6)
     1CALL REIO(2,OU,26H Problems purging /RSTR\! ,13)
      CALL CLOSE(DCB3,ERROR)
      IF(ERROR.NE.0.AND.ERROR.NE.-11)
     1CALL REIO(2,OU,26H Problems closing /PURG\! ,13)
      CALL PURGE(DCB3,ERROR,6H/PURG\,2HDL)
      IF(ERROR.NE.0.AND.ERROR.NE.-6)
     1CALL REIO(2,OU,26H Problems purging /PURG\! ,13)
      IF(LU.NE.0) GO TO 1005
      CALL CLOSE(DCB5,ERROR)
      IF(ERROR.NE.0.AND.ERROR.NE.-11)
     1CALL REIO(2,OU,30H Problems closing INPUT FILE! ,15)
1005  IF(IBUF.EQ.2H/A) CALL REIO(2,OU,16H Mover aborted! ,8)
      END
      END$
```

CUSTOMIZED SERVICE USING THE HELLO FILE

Don McLaren/Martin Marietta

The system manager of an RTE-IVB Operating System is responsible for setting up and maintaining all user and group accounts. This becomes a full time responsibility when the number of accounts is not only large but also requires a unique operating environment for each user within each general classification.

The 'HELLO' file can become a very valuable asset and means by which each user account can be customized to fit the needs of the user. Ideally the same 'HELLO' file would be used by all users of the system. A single 'HELLO' file to be maintained and updated would make the system manager's task somewhat easier. The requirements for a common 'HELLO' file include some unique characteristics and features utilized to provide the customized services.

A general section for common user requirements and common system messages would be needed. This general section would also be used by the system manager to abort log-on processes which were not desired. It may be desirable to abort all log-on attempts due to such reasons as maintenance, installation of new system resources or other site oriented reasons.

A specialized section would be required to serve the needs of each individual user or set of users. Specific resources might need to be assigned to individual users. A line printer may need to be assigned to users utilizing a specific terminal. Other needs peculiar to certain users must also be provided for. This specialized section would serve all of these needs.

The structure of a 'HELLO' file custom designed to fulfill these needs becomes a file with a single entry point and several exit points as depicted in figure 1.

The single entry point provides the means to display messages of a general nature and also to perform functions which are required by all users. It is in this section that the log-on process may be terminated when deemed necessary by the system manager.

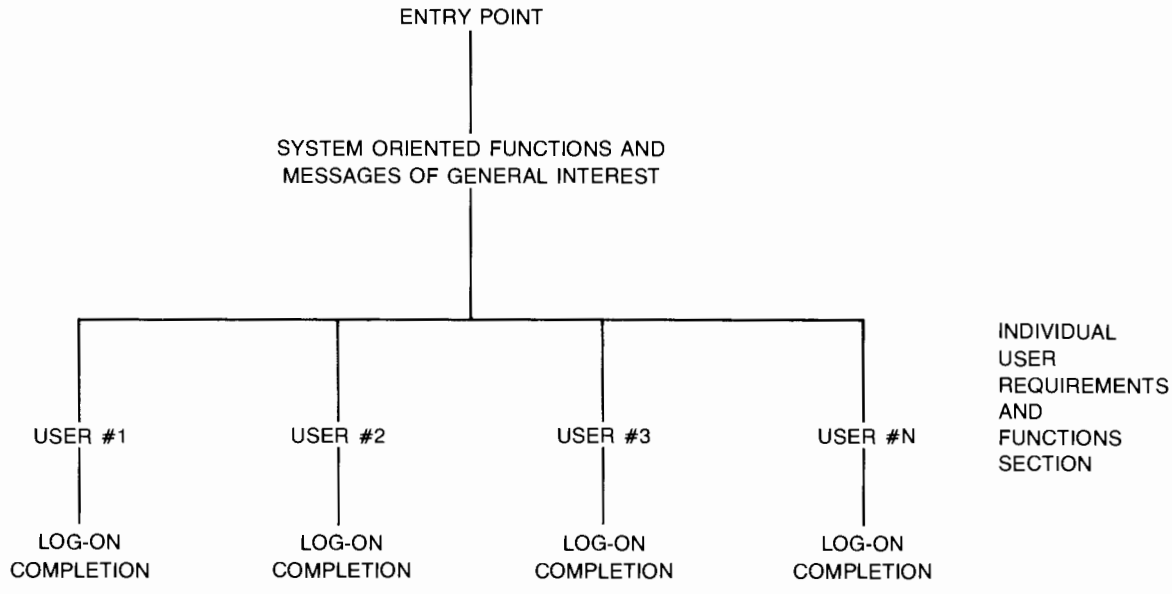


Figure 1

The individual requirements and functions section is utilized to perform services for specific users. These would include functions such as enabling a special soft key set, enabling spooling for specific applications, scheduling programs, etc.

Chapter 3 of HP 92068A RTE-IVB Terminal User's Reference Manual defines and describes all of the File Manager commands. All of these commands are candidates for use in the 'HELLO' procedure file. The 'IF' command described on page 3-65 is of particular interest. This command will allow testing of globals and logical decisions to be made as a result of these tests. As an example, assume that the soft keys must be enabled from file 'SFKEYS' if global '9P' equals 42. The procedure file would take the following form.

```
.
.
.
:IF,9P,NE,42,1
:DU,SFKEYS,1
.
.
.
```

The 'IF' statement will skip one command if '9P' is not equal to 42 (i.e. will not enable the soft keys). If '9P' is equal to 42 then no commands are skipped and the soft keys are enabled from 'SFKEYS'. This technique will be expanded to enable specified functions to be performed for specific users.

The means to identify specific users within the 'HELLO' file is accomplished through strategic use of the user capability level and globals 8P and 9P.

Globals 8P and 9P are new 'P' type globals which were introduced in the software revision update for RTE-IVB. Globals 8P and 9P are provided for session use and identification. These globals may be tested or read with standard procedure file commands. They can not be altered by procedure file commands.

The actual value contained in global '9P' is the capability level of the user who is currently utilizing the 'HELLO' file and is in the process of logging on. This value is defined when the user accounts are defined by the system manager (see HP 92068A System Manager's Manual, page 6-18).

The actual value contained in global '8P' is the system LU number of the log-on terminal. This system LU number is defined during system generation.

The system manager has a fair degree of freedom in choosing the capability level of a particular user. An examination of the capability level table on page 6-22 of HP 92068A RTE-IVB System Manager's Manual reveals that there are actually ten (10) different numbers available to specify a particular capability level. As an example, capability level 40 can be specified by selection of any of the numbers 40 through 49. In selecting one of the numbers 40 through 49 keep in mind that this number will be placed in global '9P' every time the user logs on and this number can be used to identify this specific user. The 'HELLO' procedure file can interrogate this number which provides the means whereby this user can be given special services.

The limit for identification is limited to ten (10) specific users for each capability level which should suffice for most systems. The system terminal LU number in global '9P' is also available and can be used to identify users who are restricted to a specific terminal or set of terminals.

A hypothetical system will be defined to illustrate the construction of a 'HELLO' file to provide customized service for its users. The 'SYSTEM' will be kept simple with only the variations necessary to demonstrate the ideas and concepts already described.

Assume the following system:

1. All users will receive the 'NEWS BULLETIN' each time they log-on.

2. All users have automatic spooling to the line printer 'LU 6', except users of terminal #83. The output from terminal #83 will have automatic spooling to the auxiliary line printer 'LU 7'.

3. The user with capability 42 is to be provided with a set of soft keys enabled from file 'SFKEYS'.

4. The user with capability 43 has a procedure file printed on his terminal each time he logs on. He uses auxiliary mag tape 'LU 9' instead of the standard mag tape 'LU 8', also disc 'LU 33' is to be mounted as a private cartridge.

5. For simplicity assume that the equipment numbers are the same as the LU numbers in all cases.

6. The user accounts are defined by the system manager to include the line printer 'LU 6' and the mag tape 'LU 8'. At least two spare SST entries are allowed and an aditional disc cartridge for the user with capability level 43.

The 'HELLO' procedure file takes the form shown following:

```
* * * *  H E L L O     P R O C E D U R E     F I L E   * * * *

0001  :SV,0,,IH
0002  :*   ***********************************************
0003  :*   *                                             *
0004  :*   *  N E W S     B U L L E T I N                *
0005  :*   *                                             *
0006  :*   ***********************************************
0007  :*   *                                             *
0008  :*   *   THE NEW SORT ROUTINE WILL BE AVAILABLE    *
0009  :*   *   MONDAY MORNING.                           *
0010  :*   *   TO SCHEDULE THIS PROGRAM FOLLOW           *
0011  :*   *   INTERACTIVE INSTRUCTIONS AFTER TYPING     *
0012  :*   *                                             *
0013  :*   *  RU,NSORT                                   *
0014  :*   *                                             *
0015  :*   ***********************************************
0016  :SV,4,,IH
0017  :IF,8P,EQ,83,4
0018  :SYLU,6,6
0019  :SYUP,6
0020  :SL,6,,,6
0021  :IF,8P,EQ,8P,4
0022  :SL,7,7
0023  :SYLU,7,7
0024  :SYUP,7
0025  :SL,6,,,7
0026  :IF,9P,EQ,43,6
0027  :IF,9P,EQ,42,2
0028  :SV,0,,IH
0029  ::
0030  :DU,SFKEYS,1
0031  :SV,0,,IH
0032  ::
0033  :SL,8,9
0034  :SYLU,9,9
0035  :SYUP,9
0036  :MC,33,P
0037  :SV,0,,IH
0038  :*
0039  :* BE SURE TO REMOVE THE WRITE ENABLE RING
0040  :* AND MOUNT THE LATEST TEST TAPE ON TAPE DRIVE #2
0041  :*
```

The following section details the steps in the sample 'HELLO' file:

> This section is the general section and is executed each time a user logs on.
>
> The following command used in this section will abort the log-on process for all users.
>
> :EX

line #1 Sets the severity code to 0 which allows the 'NEWS BULLETIN' to be printed on the terminal. The 'IH' parameter inhibits the echo of the severity code command on the screen.

lines #2-15 This is the news bulletin and is updated with the news of the day. The length of the news flash is variable as required.

line #16 This command sets the severity code to 4. Severity code 4 inhibits the command echo back to the terminal and forces continuation in case of error.

line #17 This command tests the number of the log-on terminal.

If the terminal is terminal #83 then, skip four (4) lines and execute line #22 next.

If the terminal is not terminal #83 then, execute line #18 next.

> Lines #18 through #20 set up spooling to the line printer 'LU 6'.

line #18 This command reinitializes 'LU 6' to be 'EQ 6'. The purpose of this command is to insure that 'LU 6' is not pointing to another EQT number from a previous operation.

line #19 This command 'UPs' the line printer just in case it was down from a previous operation.

line #20 This command sets out-spooling to 'LU 6'. All output to the line printer will be spooled and normally printed when the user logs off.

line #21 This command is a form of a 'GO TO'. It forces four (4) lines to be skipped unconditionally. The next command for execution is on line #26.

> Lines #22 through #25 set up spooling to the auxiliary line printer 'LU 7' for the users of terminal #83.

line #22 This command adds 'LU 7' to the session switch table (SST). this must be done prior to addressing 'LU 7'.

line #23 This command reinitializes 'LU 7' to be 'EQ 7'. The purpose of this command is to insure that 'LU 7' is not pointing to another EQT number from a previous operation.

line #24 This command 'UPs' the line printer just in case it was down from a previous operation.

line #25 This command sets up out-spooling to 'LU 7'. All output normally addressed to 'LU 6' will be spooled to 'LU 7' and printed when the user logs off.

line #26 This command tests the capability level of this user.

If the capability level is 43 then, skip 6 lines and execute line #33.

If the capability level is not 43 then, execute line #27 next.

line #27 This command tests the capability level of this user.

If the capability level is 42 then, skip 2 lines and execute line #30.

If the capability level is not 42 then, execute line #28 next.

line #28 Sets the severity code to 0 which allows the operator entered commands to be printed on the terminal. Severity code 0 also allows operator intervention in the case of errors. The 'IH' parameter inhibits the echo of the severity code command on the screen.

line #29 This command is executed if the capability level is not 42 or 43. It serves as an exit for the 'HELLO' file and the next command is taken from the user.

Lines #30 through #32 provide special services for the user with capability level 42.

line #30 This command enables the soft keys for the current users terminal.

line #31 Sets the severity code to 0 which allows the operator entered commands to be printed on the terminal. Severity code 0 also allows operator intervention in the case of errors. The 'IH' parameter inhibits the echo of the severity code command on the screen.

line #32 Serves as an exit for the 'HELLO' file and the next command is taken from the user.

Lines #33 through #41 provide special services for the user with capability level 43.

line #33 This command modifies 'LU 8' in the session switch table (SST) to select 'LU 9'. This will direct all I/O for 'LU 8' to the auxiliary mag tape, 'LU 9'.

line #34 This command reinitializes 'LU 9' to be 'EQ 9'. The purpose of this command is to insure that 'LU 9' is not pointing to another EQT number from a previous operation.

line #35 This command 'UPs' the mag tape unit just in case it was down from a previous operation.

line #36 This command mounts disc cartridge 'LU 33' as a private cartridge.

line #37 Sets the severity code to 0 which allows the operator entered commands to be printed on the terminal. Severity code 0 also allows operator intervention in the case of errors. The 'IH' parameter inhibits the echo of the severity code command on the screen.

lines #38-41 These lines are the procedure message and are printed on the users terminal.

NEW ATS DTU MANUAL UPDATE

Bob Desinger/HP Data Systems Division

Newly updated for RTE-IVB is the HP 9415A Digital Test Unit Subsystem Manual, part number 09580-93091. If you have an ATS/1000 with RTE-IVB and a Digital Test Unit (DTU), you'll want to obtain the update.

The old information, Update 1 from July 1979, pertained to RTE-IVA. The new Update 2 explains the differences between the standard DTS-70 software and its implementation in RTE-IVB ATS/1000 systems, and tells how to configure TESTAID/FASTRACE and the SFTs/SPTs into your ATS system. Additionally, the update contains listings of MANDTU, TSEQ15, __#SFT, __#SPT, &FTSF5, and &XSIN5.

Contact your sales representative for a copy of Update 2 or for the combined package of the DTU Manual and Update 2.

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 USER 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. 9/81

Part No. 5951-6111