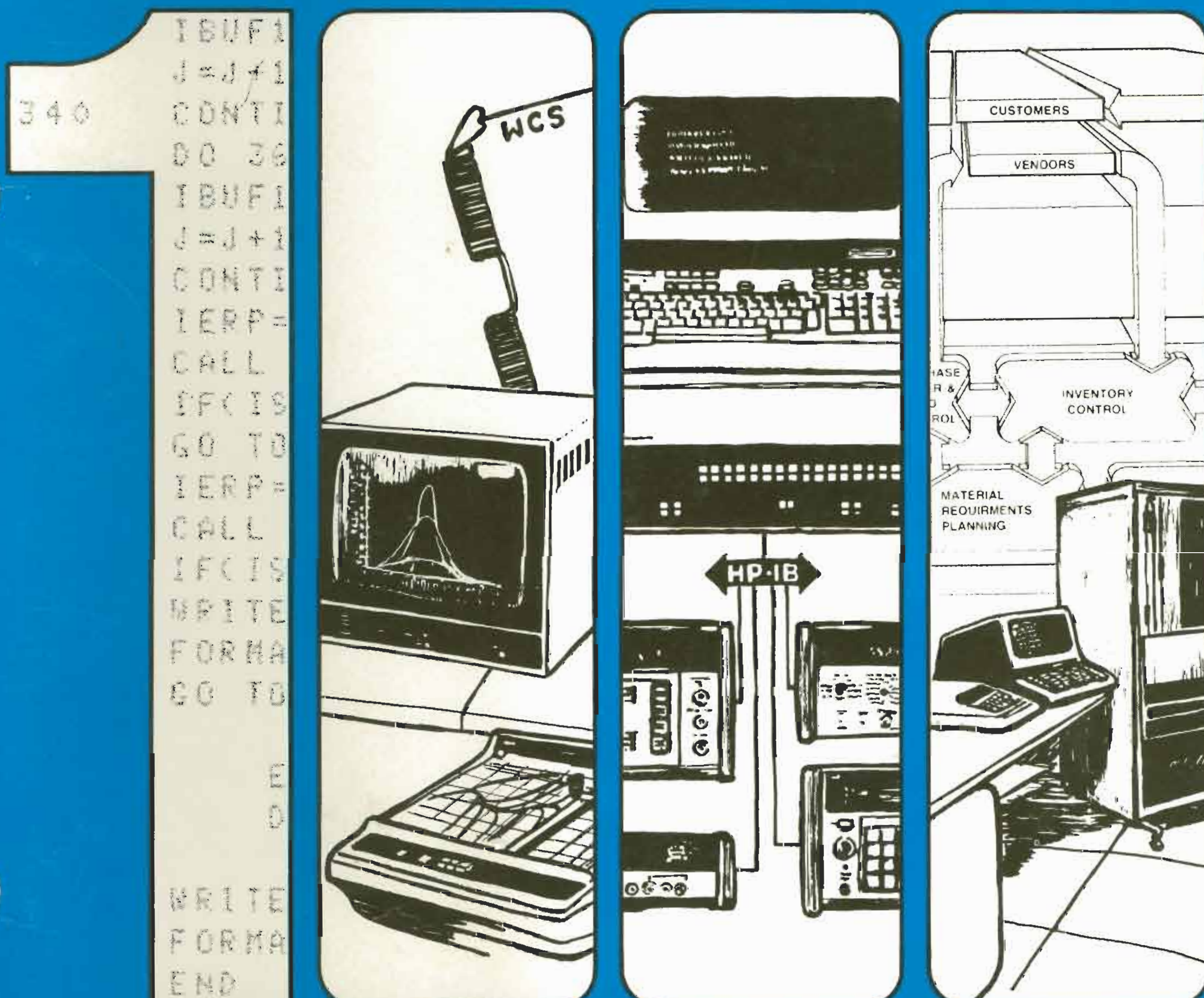
Hewlett-Packard Computer Systems COMMUNICATOR



1980

Volume IV Issue 2

HEWLETT-PACKARD
COMPUTER SYSTEMS

Volume IV
Issue 2

COMMUNICATOR/1000

Feature Articles

Departments

EDITOR'S DESK

About This Issue

Nearly all the feature articles in this issue of the Communicator/1000 were written by HP customers. This is because all of the articles we have received recently were sent in by customers. The articles cover a wide variety of subjects.

In the OPERATIONS MANAGEMENT section there are two articles. The first, "Multi-terminal ID Segment Handler", by Michael Wingham of Ducros Meilleur & Associates Ltd. explains a method of managing ID segments of EXEC scheduled programs in an RTE-IVA MTM environment. This article will prove useful for system managers with many heavily used segmented programs on their systems. In the second article, "An Interface to IMAGE", Mike Wells of Technical Analysis Corporation describes how he manages the data base operations of many users. His article gives ideas for providing increased security and data base integrity.

The third and final customer article is in the area of DATA COMMUNICATIONS. Jean-Luc de Schutter of Distrigaz writes an article describing a set of programs he wrote which use DS/1000 to talk to a secondary system terminal as though it were on his primary system. I found "Using DS/1000 and RTE-IVA to Achieve Virtual Peripherals" to be quite interesting; the concept of virtual peripherals is a natural extension of DS/1000.

The OPERATING SYSTEMS section contains the third article in the LINKS/1000 series. Bob Niland's easy-to-understand explanations of HP's subroutine linkage conventions will be helpful to even the most experienced readers.

Each of these articles contains interesting ideas for thought. However, we could only award one calculator. Since Bob Niland has already been awarded a calculator for his series he is ineligible. Our panel of judges concurred that the calculator winner is:

Best Feature Article by a Customer	AN INTERFACE TO IMAGE Mike Wells

Mr. Wells' article can be understood by someone knowing little about IMAGE and yet it can stimulate ideas in the minds of experienced users. I hope that the increase in competition will not deter entrants in the customer category. We try to have at least one entry in each calculator category while maintaining no more than two entries in a particular topic section.

One last note I wish to make is a correction to the Editor's Desk of Volume III, Issue 6. The PENNY program was written by Jim McClure rather than Jim Long. My apologies to Jim McClure for this slip of the pen.

The Editor

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 not in the Data Systems Division Technical Marketing Dept.

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

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. Employees of Technical Marketing at HP's Data Systems Division factory in Cupertino are not eligible to win a calculator.

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

LETTERS TO THE EDITOR

Dear Editor,

I am developing software for an RTE- (A, soon to be B) system and must be able to unambiguously identify peripheral hardware to my I/O calls.

Printer output can be routed to:

2621P through DVA05
2631A through DVA12
2608A through DVB12

Interactive I/O can take place with:

2645A through DVA05
2621A through DVA05
(others) through DVR00

A few id's are easy, e.g. printer output to an EQT of DV.05 can be assumed a 2621P, or a terminal on DV.00 is assumed to be a teletype device. But I need to know how to determine the type of a terminal on DV.05. On page 6-3 of the 2645 Reference Manual (02645-90005) bit 4 of byte 3 of the primary status word is identified as "terminal type". It is shown as being set. However, the corresponding datum for the 2621 is also shown as set (page C-3, manual 02620-90001).

I need not only a method to differentiate among devices, but it must also be a method devoid of "tricks", i.e. it must be supported in the HP documentation of the devices and drivers.

Many thanks in anticipation of your solution.

Yours truly,

Craig B. Spengler
Aim Management Services, Inc.

Dear Sir,

According to Data Terminals Division terminal types can be distinguished from the secondary status word (page 6-5 of the 2645 Reference Manual). Byte 8 of the secondary status word contains information about terminal firmware. Terminal firmware is what makes a 2648 different from a 2645, etc. In a 2645, bit 1 of byte 8 will be the only bit set. In a 2648 bits 1 and 3 will be set, and in a 2647 bits 1,3, and 4 will be set. A 2621 can be differentiated in that it doesn't have a secondary status word. Nothing is returned from a secondary status request to the 2621.

I hope this answers your questions regarding peripherals of DVR05. If you have trouble identifying different peripherals of another driver please write again.

With best regards,

Editor, Communicator/1000

EDITOR'S DESK

Dear Editor,

Here is an update to the article "Remote System Control Via DS/1000" in Volume III, Issue 5 of the Communicator.

The example

```
CALL XMSG (''ENCOUNTERED ON WRITE'',12,5,-3,-4)
```

should read

```
CALL XMSG (20HENCOUNTERED ON WRITE,10,5,-3,-4)
```

The continuous "modem" problems described turned out to be caused by a too-low EQT timeout value which was set according to the instructions on page 3-52 of the Network Manager's Manual, which was misleading. In fact, errors are few, occurring only every 500-1000 records on the average.

The dummy driver (DVD00) described was not fully debugged at time of writing (and has been put aside for more important work).

None of the authors are at Holloman anymore although the work has been continued by others. Sgts. Beyer and Reynolds are studying for their batchelors degrees at University of Arizona, and I am with HP in El Paso.

Keep up the good work, and thanks for the calculator!

Best regards,

John Pezzano
HP, El Paso

Dear Sir,

Thanks for keeping us up-to-date on the whereabouts of the authors of "Remote System Control Via DS/1000". The information will be helpful to anyone wishing to contact you with questions regarding your application. Thanks also for updating your article with the correct XMSG calls. I appreciate your correspondance, and hope you will continue to submit articles for publication in the Communicator/1000.

Sincerely,

Editor, Communicator/1000

Dear Editor,

The "RTE-III, RTE-II, and BATCH-SPOOL MONITOR Pocket Guide" (part number, 92060-90010) is a very useful publication. Is a similar publication available for RTE-IVB? If a similar publication exists, how do I get one? If a similar publication does not exist at this time, what is its anticipated publication date?

Sincerely,

W.B. Fegley
Hughes Aircraft Company

Dear Sir,

I agree that the Pocket Guide is very useful. The RTE-IVB Pocket Guide has been in the workings since last summer, and is finally available. For ordering information see the bulletins section of this Communicator.

With Best Regards,

Editor, Communicator/1000

LIST OF DRIVERS AND THEIR SIZES

Ed Gillis/HP Pittsburgh

Drivers are loaded into partitions during a generation by a first fit — first used algorithm. Therefore, to help optimize your memory usage, you should relocate the drivers being used in descending word-length size. The following list of drivers and decimal word sizes should help in determining the optimal order of drivers in a particular generation.

DVR00	600	
DVR00	600	
0DV05	925	
4DV05	1450	
DVA05	1600	
1DV10	1000	
2DV10	200	
DVR11	600	
DVR12	350	
DVA12	450	
DVB12	900	
DVZ12	200	
DVA13	160	
DVR15	630	
DVR23	475	
DVR31	700	
DVR32	950	
DVR33	625	
DVR36	1100	
1DV37	1100	
2DV37	1375	
4DP43	425	
2DV47	1750	— Should be placed in System Driver Area
3DV47	1850	— Should be placed in System Driver Area
DVR50	4800	— Should be placed in System Driver Area
2DV62	550	
3DV62	680	
4DV62	600	
DVA65	650	
DVG67	775	
DVA72	900	
DVM72	500	

UTILITY FOR DUPLICATING MAGNETIC TAPES

Don Pottenger/HP Data Systems Division

As one uses any computer system a need always arises to copy a magnetic tape. Since no such utility currently exists, MTDUP was born. MTDUP only works with 9 and 7 track, 800 and 1600 BPI tape drives. It has the capability to deblock and/or convert EBCDIC to ASCII while copying.

The deblocking feature will output records of the same length in a given output file, and requires that the blocking factor divide evenly into the record length of the input file. Otherwise, data may be lost while deblocking. The largest input record size allowed by MTDUP is 10000 words.

Scheduling is done as follows:

```
:RU,MTDUP,mt#1,mt#2,#files,blocking factor,EBCDIC flag
```

where data is being copied from mt#1 to mt#2. The number of files to copy is specified by #files, the number of records per block on mt#2 is specified by blocking factor, and EBCDIC characters on mt#1 are converted to ASCII on mt#2 if the EBCDIC flag is a "1".

```
FTN4,L
C
C      MTDUP HAS BEEN ENHANCED TO UNBLOCK TAPES AND CONVERT
C      EBCDIC TO ASCII.  IN ADDITION IT WILL CHECK FOR AND
C      STOP AT A DOUBLE END OF FILE.
C
C      SCHEDULING PARAMETERS:
C
C        RU,MTDUP,FROM MT,TO MT,# FILES,BLOCKING,EBCDIC FLAG
C
C           #FILES = -1 GO TO DOUBLE EOF
C
C          BLOCKING = NUMBER OF RECORDS/BLOCK (DEFAULT: 1)
C
C      EBCDIC FLAG = IF #0 CONVERT FROM EBCDIC TO ASCII
C                    (DEFAULT: ASCII TO ASCII)
C
C      TYPICAL SCHEDULING WOULD BE:
C
C          RU,MTDUP,10,8,-1
C
C      THIS WOULD COPY A TAPE FROM LU 10 TO LU 8 UNTIL DEOF IS
C      ENCOUNTERED.
C
       PROGRAM MTDUP(3,70),MTCOPY W EBCDIC CONVERSION OPTION 800215
C
       DIMENSION IBA(10000),IPRAM(5),IMES1(15),IMES2(21),IMES3(13)
       EQUIVALENCE (IPRAM(4),IBLK),(IPRAM(1),MT1),(IPRAM(2),MT2)
       EQUIVALENCE (IPRAM(3),ICNT),(IPRAM(5),EBCDIC)
C
       DATA IMES1/2HRU,2H,M,2HTD,2HUP,2H,F,2HRO,2HM ,2HMT,2H,T,2HO ,2HMT,
      #2H,#,2HFI,2HLE,2HS /
       DATA IMES2/2H E,2HND,2H O,2HF ,2HFI,2HLE,7*2H  ,2H R,2HEC,2HOR,
      #2HDS,2H W,2HRI,2HTT,2HEN/
       DATA IMES3/2H D,2HOU,2HBL,2HE ,2HEN,2HD ,2HOF,2H F,2HIL,2HE ,
      #2HFO,2HUN,2HD /
C
```

```
      CALL RMPAR(IPRAM)
      LOG  = LOGLU(ISES)
      IF(IBLK .LE. 0)IBLK = 1
      IF(ICNT .LT. 0)ICNT = 32767
      IF(ICNT .EQ. 0)ICNT = 1
      IFILE = 1
      IEOF = 0
C
C  CHECK FOR MT DEVICE
C
      DO 5 I=1,2

      CALL EXEC(13,IPRAM(I),ISTAT)
C
      IF(IAND(ISTAT,37400B)*2 .EQ. 23000B)GO TO 5
      IF(IAND(ISTAT,37400B)*2 .EQ. 24000B)GO TO 5
      CALL EXEC(2,LOG,IMES1,15)
      GO TO 999
C
  5   CONTINUE
C
      DO 10 I=1,ICNT
      IRCNT = 0
      IB1 = 0
1     IF(IFBRK(IDUM))999,2
C
C   READ UP TO 10000 WORDS FROM MAG TAPE 1 INTO IBA ARRAY
C
2     CALL EXEC(1,100B+MT1,IBA,10000)
      CALL ABREG(IA,IB)
C
C   CHECK FOR END OF FILE ON MAG TAPE 1
C
      IF(IAND(IA,200B).EQ.200B)  GO TO 990
C   RESET EOF FLAG IF ONLY SINGLE EOF IS FOUND
      IEOF = 0
      IF(IB1 .EQ. 0)IB1 = IB
C
C   CHANGE ASCII TO EBCDIC IF SPECIFIED IN RUN STRING
C
      IF(EBCDIC .NE. 0)CALL CONVT(IBA,IB)
C
C   CHECK FOR LAST BLOCK HAVING BEEN REACHED
C
      IF(IB .NE. IB1 .AND. IBLK .GT. 1)GO TO 900
      J = 1
C
C   CALCULATE LENGTH OF RECORDS TO BE WRITTEN TO MAG TAPE 2
C
      ILEN = IB/IBLK
C
C   WRITE A BLOCK OF RECORDS TO MAG TAPE 2
C
      DO 20 K=1,IBLK
      CALL EXEC(2,100B+MT2,IBA(J),ILEN)
      J = J + ILEN
C  COUNT NUMBER OF RECORDS WRITTEN TO MAG TAPE 2
      IRCNT = IRCNT + 1
 20   CONTINUE
```

```
      GOTO 1
900   J = 1
C
C   CALCULATE NUMBER OF OUTPUT RECORDS IN AN
C   UNEVEN BLOCK OF INPUT
C
      IFBLK = IB/ILEN
C
C   WRITE BLOCK AND RETURN TO READ NEXT RECORD
C
      DO 30 K=1,IFBLK
      CALL EXEC(2,100B+MT2,IBA(J),ILEN)
      J = J + ILEN
      IRCNT = IRCNT + 1
 30   CONTINUE
      GO TO 1
C
C   CHECK FOR DOUBLE END OF FILE ON MAG TAPE 1
C
990   IF(IEOF .EQ. 1)GO TO 995
C
C   IF EOF, WRITE A MESSAGE TO LOG DEVICE STATING THE NUMBER
C   OF THE FILE JUST WRITTEN AND HOW MANY RECORDS IT CONTAINED
C
      CALL CNUMD(IFILE,IMES2(7))
      CALL CNUMD(IRCNT,IMES2(11))
      CALL EXEC(2,LOG,IMES2,21)
      IFILE = IFILE + 1
      IEOF = 1
      ENDFILE MT2
 10   CONTINUE
      GO TO 999
C
C   WRITE DOUBLE END OF FILE TO MAG TAPE 2
C
 995  CALL EXEC(2,LOG,IMES3,13)
      ENDFILE MT2
999   END
```

```
                      ASMB,L,Q
**  NO ERRORS PASS#1 **RTE ASMB 92067-16011**


                      ASMB,L,Q
     00000                     NAM  CONVT,7 EBCDIC>ASCII CONVERSION 790419
                               EXT  .ENTR
                               ENT  CONVT
                               SUP
*       A FORTRAN-CALLABLE CONVERSION ROUTINE WHICH REPLACES
*    THE CONTENTS OF AN EBCDIC ARRAY WITH ITS ASCII EQUIVALENT.
*    CONVERTS ON THE BASIS OF TWO CHARACTERS PER WORD.
*
```

```
*     NOTE: OCTAL VALUE 255 WILL BE TRANSLATED TO A '['
*           OCTAL VALUE 275 WILL BE TRANSLATED TO A ']'
*           THIS IS NOT A DEFINED EBCDIC CODE.
*
*   CALLING SEQUENCE:
*
*      CALL CONVT (IARRAY,LENGTH)
*
*   WHERE "IARRAY" IS THE SYMBOLIC ARRAY NAME AND "LENGTH"
*   IS THE NUMBER OF 16-BIT WORDS TO BE CONVERTED.
00000 000000  POINT  NOP
00001 000000  BLKSZ  NOP
00002 000000  CONVT  NOP
00003 000001X        JSB  .ENTR     RETRIEVE PARAMETER ADDRESSES.
00004 000000R        DEF  POINT
00005 000001R        LDA  BLKSZ,I   GET NUMBER OF RECORDS.
00006 003004         CMA,INA        CREATE A WORD COUNTER.
00007 000001R        STA  BLKSZ
00010 000000R GO     LDA  POINT,I   GET A PAIR OF EBCDIC CHARACTERS
00011 006400         CLB            A-REG: (E1,E2) B-REG: (00,00)
00012 101110         RRR  8                (00,E1)        (E2,00)
00013 000024R        JSB  .CNVT            (A1,00)        (E2,00)
00014 100110         RRL  8                (00,E2)        (00,A1)
00015 000024R        JSB  .CNVT            (A2,00)        (00,A1)
00016 101110         RRR  8                (A1,A2)        (00,00)
00017 000000R        STA  POINT,I   AND STORE IN PLACE OF EBCDIC.
00020 000000R        ISZ  POINT     NEXT PAIR.
00021 000001R        ISZ  BLKSZ     FINISH CHECK.
00022 000010R        JMP  GO        NOPE.
00023 000002R        JMP  CONVT,I
00024 000000  .CNVT  NOP
*                    THE EBCDIC CHARACTER IN THE LOW BYTE OF THE A-REG IS
*                    USED AS A TABLE ADDRESS OF ITS ASCII EQUIVALENT,
*                    WHICH IS THEN RETURNED IN THE HIGH BYTE OF A-REG.
00025 000065         CLE,ERA        (A-REG)/2, REMAINDER IN E.
00026 000035R        ADA  TABL.     FORM TABLE WORD ADDRESS.
00027 000000         LDA  A,I       GET TWO CHARACTERS.
00030 002040         SEZ            REMAINDER INDICATES WHICH IS
00031 001727         ALF,ALF        TO BE RETURNED IN A-REG.  THE
00032 000034R        AND  CMASK     UNWANTED ONE IS MASKED OUT.
00033 000024R        JMP  .CNVT,I
*
00000         A      EQU  0
00001         B      EQU  1
*
00034 177400  CMASK  OCT  177400
00035 000036R TABL.  DEF E000      LOCATION OF TRANSLATION TABLE
*
*  THIS IS THE TABLE FOR CONVERSION FROM EBCDIC TO ASCII
*   XXX INDICATES NO TRANSLATION, THE RESULTING CHARACTER
*   HAS BIT 7 SET (HIGH ORDER BIT) AND BITS 0 THRU 6 REMAIN
*   THE SAME AS THE SOURCE CODE
*********************************************************************
*
*                     EBCDIC _  0  1   2  3   4  5   6  7
*                     ^^^
00036 000001  E000  OCT 000001,001003,102011,103177
*                     ASCII--> NULSOH STXETX XXX HT XXXDEL
*
```

```
     00042 104211   E010   OCT 104211,105013,006015,007017
*                              XXXXXX XXX VT  FF CR  SO SI
*
     00046 010021   E020   OCT 010021,011023,112012,004000
*                              DLEDC1 DC2DC3 XXX LG BS NUL
*
     00052 014031   E030   OCT 014031,115233,016035,017037
*                              CAN EM XXXXXX  FS GS  RS US
*
     00056 120241   E040   OCT 120241,121243,122012,013433
*                              XXXXXX XXXXXX XXX LF ETBESC
*
     00062 124251   E050   OCT 124251,125253,126005,003007
*                              XXXXXX XXXXXX XXXENQ ACKBEL
*
     00066 130261   E060   OCT 130261,013263,132265,133004
*                              XXXXXX SYNXXX XXXXXX XXXEOT
*
     00072 134271   E070   OCT 134271,135273,012025,137032
*                              XXXXXX XXXXXX DC4NAK XXXSUB
*
     00076 020301   E100   OCT 020301,141303,142305,143307
*                              SP XXX XXXXXX XXXXXX XXXXXX
*
     00102 144311   E110   OCT 144311,055456,036050,025441
*                                XXXXXX  [  .   <  (   +  !
*
     00106 023321   E120   OCT 023321,151323,152325,153327
*                               & XXX XXXXXX XXXXXX XXXXXX
*
     00112 154331   E130   OCT 154331,056444,025051,035536
*                              XXXXXX  ]  $   :  )   :  7
*
     00116 026457   E140   OCT 026457,161343,162345,163347
*                               -  /  XXXXXX XXXXXX XXXXXX
*
     00122 164351   E150   OCT 164351,076054,022537,037077
*                              XXXXXX  !  ,   %  -   >  ?
*
     00126 170361   E160   OCT 170361,171363,172365,173367
*                              XXXXXX XXXXXX XXXXXX XXXXXX
*
     00132 174140   E170   OCT 174140,035043,040047,036442
*                              XXX \   :  #   @  '   =  "
*
*************************************************************************
*  THE FOLLOWING TRANSLATION IS FROM LOWER CASE EBCDIC
*     TO LOWER CASE ASCII
*************************************************************************
*
     00136 100141   E200   OCT 100141,061143,062145,063147
*                              XXX A   B C   D E   F G
*
     00142 064151   E210   OCT 064151,105213,106215,107217
*                                H  I  XXXXXX XXXXXX XXXXXX
*
     00146 110152   E220   OCT 110152,065554,066556,067560
*                              XXX J   K L   M N   O P
*
```

```
0133  00152 070562  E230   OCT 070562,115233,116235,117237
0134*                            Q  R  XXXXXX XXXXXX XXXXXX
0135*
0136  00156 120176  E240   OCT 120176,071564,072566,073570
0137*                          XXXESC  S  T   U  V   W  X
0138*
0139  00162 074572  E250   OCT 074572,125253,126133,127257
0140*                            Y  Z  XXXXXX XXX [  XXXXXX
0141*
0142  00166 130261  E260   OCT 130261,131263,132265,133267
0143*                          XXXXXX XXXXXX XXXXXX XXXXXX
0144*
0145  00172 134271  E270   OCT 134271,135273,136135,137277
0146*                          XXXXXX XXXXXX XXX ]  XXXXXX
0147*
0149********************************************************************
0150*  THE FOLLOWING TRANSLATION IS FROM UPPER CASE EBCDIC
0151*    TO UPPER CASE ASCII
0152********************************************************************
0153*
0154  00176 075501  E300   OCT 075501,041103,042105,043107
0155*                               A   B  C   D  E   F  G
0156*
0157  00202 044111  E310   OCT 044111,145313,146315,147134
0158*                            H  I  XXXXXX XXXXXX XXX \
0159*
0160  00206 076512  E320   OCT 076512,045514,046516,047520
0161*                            \  J   K  L   M  N   O  P
0162*
0163  00212 050522  E330   OCT 050522,155333,156335,157337
0164*                            Q  R  XXXXXX XXXXXX XXXXXX
0165*
0166  00216 160134  E340   OCT 160134,051524,052526,053530
0167*                          XXX \   S  T   U  V   W  X
0168*
0169  00222 054532  E350   OCT 054532,165353,166355,167357
0170*                            Y  Z  XXXXXX XXXXXX XXXXXX
0171*
0172  00226 030061  E360   OCT 030061,031063,032065,033067
0173*                            0  1   2  3   4  5   6  7
0174*
0175  00232 034071  E370   OCT 034071,175373,176375,177377
0176*                            8  9  XXXXXX XXXXXX XXXXXX
0177*
0178                         END
**  NO ERRORS *TOTAL **RTE ASMB 92067-16011**
```

HP SUBROUTINE LINKAGE CONVENTIONS

Robert Niland/HP Lexington

[**Editor's Note:** This is the third part in a series of articles taken from Bob Niland's manual on HP Subroutine Linkage Conventions.]

4-1. PARAMETERS IN BLANK COMMON

A fundamental concept in parameter or argument passing is that the calling program leave or store the parameters in a location which is known to the subroutine being called. The register-passing technique discussed in sections 3-2 is an example of this, but it is apparent that it is a very limited technique, there being only four 16-bit registers in which to pass data. The address-definition techniques of section 3-4 are an improvement, but still limit the amount of data passed, and both methods are only useful in assembly language.

Where a larger quantity of parameter data must be passed, the calling routine can store it (or expect to find) it in a block of memory which is declared in both the caller and the subroutine, and is thus COMMON to both. In RTE systems this location is either a memory resident area (SYSTEM/REVERSE COMMON) or is within the program's own bounds (LOCAL COMMON). Selection of common location is determined at relocation time via a LOADR option. Access is via declarations in RTE Assembler or Fortran IV. These are fully documented in the RTE-IV ASSEMBLER and RTE FORTRAN IV manuals. Typical invocations are shown below. These are examples of BLANK COMMON and should not be confused with NAMED COMMON which is discussed in section 4-2.

Sample of Fortran code to access common:

```
FTN4,L,T,C
      ...
      COMMON ARRAY(2,4,6), WORDS(32), IWORD [,<member>[(<size>)]]
      ...
      IWORD = 123
      WORDS(7) = 456.7
      ARRAY(INDEX,LENGTH,NUMBER) = 890.1
      ...
```

Where: <member> is any supported FTN4 variable type which may appear in a DIMENSION statement.

<size> is the number of elements in the variable or array. The number of 16 bit words in [member] is a function of both [(size)] and [member's] data type. Default size = 1 element.

Data items in common are accessed as if they were located within the declaring routines.

Sample of functionally equivalent assembly code:

```
ASMB,R,L,C
      ...
[lbl] COM ARRAY(96), WORDS(64), IWORD [,<membr>[(<size>)]]
      ...
      LDA =D123         A=123
      STA IWORD         IWORD=A
      ...
      DLD =F456.7       A&B = 456.7
      DST WORDS+12      WORDS(7) = A&B (see note)
```

```
      . . .
      JSB ..MAP          Resolve an array address.
      DEF ARRAY          Define array start address.
      DEF INDEX          Pass 1st subscript.
      DEF LENTH          Pass 2nd subscript.
      DEF NUMBR          Pass 3rd subscript.
      DEF D0002          Length of 1st dimension.
      DEF D0004          Length of 2nd dimension.
*                        End of ..MAP call
      STA ELEM          Save element address returned by ..MAP
      DLD =F890.1        Get constant.
      DST ELEM,I        Store into element.
```

Where: [lbl]: is an optional label which will be ignored by the assembler, but which may be used for documentation where multiple COM pseudo instructions are present.

membr: is any legal ASMB external label.

size: is the size of common block [membr] in 16 bit words. Note that arrays may have but a single dimension, and that when mapping to other than INTEGER data types (as in this example), the user must compute the number of integer words in the non-integer array.

Data items in common are accessed as if they were located within the declaring routines.

Note

Very early versions of the RTE relocating LOADR cannot resolve references-with-offset to COM or EXT externals, such as DST WORDS+12 above.

The principal advantage to using common over using registers is the dramatic increase in the quantity of parameters which can be passed. The method is not without disadvantages however. Among these are:

1. Each program and/or subroutine requiring access to data items in blank common must declare each member of that common in the same order and for the same size as the originally declaring routine.

   Non-accessed members, at least, may be "skipped" by declaring single blocks of matching size. For example if a routine required access only to IWORD in our sample it could declare:

   ```
   COMMON IGNORE(160),IWORD
   ```

2. Any change in the layout of common requires editing and recompiling all&d the programs and subroutines which access that common. From the standpoint of structured programming, this is by far the most serious objection to blank common.

3. Of the three memory areas where common may reside (LOCAL, SYSTEM, REVERSE), a single program with its overlays and subroutines may be linked to only one. A program using only blank common cannot have part in the system area and part in its own program code area.

4. If a program links to (or defaults to) LOCAL COMMON, only that program and its routines can access that common.

5. If a program is linked to SYSTEM or REVERSE system COMMON, the programmer must be aware that there is no protection for the data in that area. Any other program using common, and linked to system common may access (or corrupt) the data there. Further, if two or more different applications both require system common, all but the first user of that common must declare and ignore the preceding blocks (which are owned by the other programs), in a manner similar to that shown in objection 1 above.

4-2. PARAMETERS IN NAMED COMMON AND SSGA

The limitation that each program may access only one type of blank common, and the limitation that all routines sharing a common area declare all of that area is a result of the fact that each routine generates but a single external (COM) reference to be resolved by LOADR. All accesses to elements within the common area are via offsets with respect to word 1 of that area.

An improvement over simple (or blank) common is a technique known variously as NAMED COMMON, SUBSYSTEM GLOBAL, or EXT-ENT GLOBAL. It will be referred to as NAMED COMMON in this section. Although access to data elements in NAMED COMMON is still via offset with respect to word 1 of each area, this type of common offers several improvements over blank common, viz:

1. Multiple blocks are allowed, each having a local variable name used within the routine, and a unique EXTERNAL NAME which must match that appearing in a BLOCK DATA SUBPROGRAM (BDS) or in the SUBSYSTEM GLOBAL AREA (SSGA). There is no block size limit. Each separate word or array in blank common can be a distinct NAMED COMMON block.
2. In a program having multiple blocks of NAMED COMMON, each block may reside in either a BDS or in SSGA without regard to where any other block resides.

The essence of this method is that instead of generating a single COM (COMMON) reference per routine as in blank common, NAMED COMMON declarations result in one (or multiple) EXT (external) references per routine which must be satisfied during RTxGN or LOADR execution by a corresponding ENT (entry) reference in a BDS, SSGA, or ASMB subroutine. An exception to this linkage is EMA references which are treated differently and may be discussed in future editions of this manual.

As an example, two blocks will be created using the sample arrays from the previous section. In FTN-IV the EXT references will be generated for names "LARGE" and "SMALL" below. Although a name may represent a block containing only a single element or array, the name must not be the same as the element or array variable name. This is because:

1. EXT symbols must be 5 characters or less and local variable names can be 6.
2. A single common name may refer to a block containing more than one uniquely identifiable data item or array.

```
FTN4,L,T,C
      ...
      COMMON /LARGE/ ARRAY(2,4,6)
      COMMON /SMALL/ WORDS(32), IWORD
      ...
```

In ASMB the EXT references are generated explicitly.

```
ASMB,R,L,C
      ...
      EXT LARGE,SMALL
      ...
ARRAY EQU LARGE
WORDS EQU SMALL
IWORD EQU SMALL+64
      ...
```

In both cases references to the common elements are by the use of the local variable names, and access methods are identical to those for blank common.

The ENT (entry) reference required to satisfy an FTN4 or ASMB generated EXT can reside in the program or in the system area and can be created in either Fortran or Assembly language, or both.

In FTN4 an ENT is generated for each NAME in a NAMED COMMON statement in a BLOCK DATA SUBPROGRAM. For example . . .

```
FTN4,L,T,C
      BLOCK DATA MNAME,Extended NAM record for LOADR map.
      ...
      COMMON /LARGE/ ARRAY(2,4,6)
      COMMON /FILES/ NAMRS(3,2)
      ...
      DATA NAMRS /2HFI,2HLE,2H01,
     &            2HFI,2HLE,2H02/
      ...
```

In ASMB the ENT is generated explicitly. For example . . .

```
ASMB,R,L,C
      NAM MNAME,7 Extended NAM record for LOADR map.
      ...
      ENT LARGE,FILES
      ...
LARGE BSS 00096           BSS = Block Starting Symbol
      ...
FILES ASC 03,FILE01
      ASC 03,FILE02
      ...
```

The preceding examples are functionally identical, and the following comments apply to both.

MNAME — is the user defined 5 character name assigned to the binary module which will result from compilation or assembly. It is usually used in the file namr's as well, e.g. &MNAME and %MNAME.

,7 — A Fortran IV BLOCK DATA SUBPROGRAM is always a type 7 or UTILITY SUBROUTINE module. The ASMB programmer can, of course, set the module type to any value desired. The significance of the module type is discussed in subsequent paragraphs.

DATA & ASC — One of the features of BLOCK DATA SUBPROGRAMS is that the data items within may be set to initial values by the use of the DATA statement. An equivalent result can be achieved in Assembly language through the use of the pseudo instructions ABS,ASC,BYT,DEC,DEX, and OCT. In the case of character variables, this is frequently easier than long strings of 2H's.

In programs using blank common, the memory location of the common block is specified by the program type during RTxGn and by a default or user option during LOADR. In programs using named common, the location of the blocks is a result of how they are relocated.

If the program is to be linked to NAMED COMMON in SSGA, two steps are required:

1. The module containing the ENT's must have been supplied during RTxGN. If it was an FTN4 module, it must have had its type changed to 30 during the PARAMETER INPUT PHASE of RT4GN. If an ASMB module, it must have been type 30, or have been changed to type 30.

2. If the program was loaded during RTxGN: It must have had a type in its NAM or PROGRAM statement allowing it to have SSGA access (types 17, 18, 19, 20, 25, 26, 27 and 28 in RTE-IV), or must have had its type changed during the PARAMETER INPUT PHASE. If the program is to be loaded on-line, the LOADR must be supplied with the proper scheduling parameter or command (OP,xxSSxx for RTE-IV) to permit SSGA access.

If any NAMED COMMON declarations are to be satisfied LOCALLY the module(s) containing the ENT's must be relocated with the requesting program. During RTxGN this is done by including the module with a "RELocate" command. Note: If the module is type 7, it will also be included in the relocatable library of the system being generated. If the module is only to be used to satisfy the program being loaded, the type should be changed from 7 to 8.

On-line, using LOADR these rules apply.

1. Any ENT module relocated before the system library is searched will be used to satisfy NAMED COMMON EXT references which match its ENT's. The module will be appended to the program as any subroutine would.

2. If the system library is searched (e.g. SEA,, in RTE-IV) before a local module is relocated, LOADR will attempt to satisfy the EXT's from the relocatable library or SSGA.

A final word about assembly routines: The ASMB example shown in this section is what might be termed an ASMB BLOCK DATA SUBPROGRAM. It shows how to emulate the Fortran BLOCK DATA SUBPROGRAM. However, since all that is required to satisfy a NAMED COMMON declaration in a using routine is an entry point (ENT), an assembly routine need not be just a dummy module. It may contain executable code, and in fact could be the main program.

4-3. SYSTEM CONSIDERATIONS

The preceding sections discuss the limitations of common and SSGA as they apply to the programs involved. However, the use of SYSTEM COMMON and SSGA can also have an adverse effect on users who do not participate the use of these resources.

Specifically, the inclusion of blank and/or labeled (SSGA) common during system generation consumes logical address space permanently for the life of that generation. Depending on which RTE you are implementing, and particularly on whether or not it is dynamically mapped, the size of system data areas such as SSGA, Real-time and Background Common will always reduce the maximum size of System Available Memory (SAM), and may also affect the maximum size of some types of programs, and the number of entries in the system DRT, Interrupt, MAT and ID segment tables.

The combined programming and system considerations seem to make a good case for not using these system-resident data areas, and it is wise to avoid their use except when necessary. However, there are applications in which the disadvantages of SSGA and SYSTEM COMMON become advantages because of the unique properties of these areas. Among these are:

- I/O: Throughput is much higher when I/O is unbuffered. However, a program performing unbuffered I/O is not swappable, and is tying up a partition. By placing the program's data buffer in system common or SSGA, the program can be swapped irrespective of eqt buffering.

- Multiprogramming: If more than one program needs access to a data area, that area must reside in a location common to all the programs. There are only three choices:

  1. System
  2. Program
  3. Undeclared memory

  1. System: i.e. common or SSGA is the simplest method.

  2. A data-management program using CLASS I/O (slow), or shared EMA (unsupported), could manage data for cooperating programs. DATACAP/1000, in one implementation, uses CLASS I/O and an EMA manager program to handle multiple terminals which may be in different stages of different transactions. But the "high-performance" implementation uses COMMON!

3. Undeclared memory is similar to shared-EMA, and shared access to a memory-locked program containing the data areas. It requires a mapped machine, privileged user memory management routines, and can be rendered unusable by minor HP changes in the RTE dispatcher, and is therefore unsupported.

- Privileged drivers: May not call any RTE or user routines external to themselves. They may communicate with their host programs only through SSGA or system common.

MULTIPLE TERMINAL SCHEDULER AND ID SEGMENT MANAGER

Michael P. Wingham/Ducros Meilleur & Associates Ltd.

INTRODUCTION

The RTE-IV Multiple Terminal Monitor (MTM) manages ID segments so that each user can have his own copy of a program. If the user wishes to run a program with FMGxx as the father, then in certain circumstances, a copy of the program will be created belonging to the user's terminal.

MTM will perform this action whenever the program to be run is a son of FMGxx, and the program is a Type-6 FMGR file. A copy of the program will be created with the last two characters being xx, and be scheduled for execution to terminal xx.

For example, if the EDITR is loaded on-line as a temporary load and saved as a Type-6 file, the command:

```
:RU,EDITR
```

will create a program named EDIxx and schedule it to terminal xx. When EDIxx is finished, the ID segment will automatically be returned to the system.

The advantage of processing the ID segments in this way is that all terminals can run the same program but each user gets a personal copy of the program. Therefore, a user does not have to wait for other users to finish with a program before gaining access to it. Also, ID segments which are no longer needed do not accumulate in the system.

The above features do not apply to programs scheduled by EXEC calls. This article describes a method of extending these MTM features to programs scheduled by EXEC calls. Furthermore, this method allows HP 1000 systems not operating under RTE-IV to emulate the program scheduling capabilities of MTM.

MULTIPLE TERMINAL SCHEDULER AND ID SEGMENT MANAGER

Scheduling Copies of Programs

Suppose a program called AMAIN prompts the user to enter a file name. AMAIN then schedules a program called PROC which processes the file and returns control to AMAIN. AMAIN then displays the results at the user's terminal and terminates (Figure 2-1a).

```
FTN4
      PROGRAM AMAIN
      INTEGER PROC(3)
      DATA PROC/2HPR,2HOC,2H  /
        .
        .
        .
        .
        .
        .
        .
        .
        .
C**** PROMPT USER FOR FILE NAME
        .
        .
        .
C**** QUEUE SCHED PROC WITH WAIT
C
      CALL EXEC(23,PROC)
        .
        .
        .
C**** DISPLAY RESULTS
        .
        .
        .
      STOP
      END
```

Figure 2-1a

```
FMG24 ---> AMA24 --+
                   !
                   !
                   +---> PROC
                   !
                   !
FMG42 ---> AMA42 --+
```

Figure 2-1b

Now suppose 2 users run AMAIN at terminals 24 and 42. We have a program scheduling hierarchy as in Figure 2-1b. Note that since PROC is scheduled by an EXEC call, copies are not created, and both AMA24 and AMA42 are competing for the same program. Only the first program which attempts to schedule PROC will be successful; the second will be placed in a queue until PROC terminates. Thus the advantages of MTM creating copies of AMAIN have been nullified. What is required is a copy of PROC for each copy of AMAIN. A subroutine called MTMRN has been written to perform this function (Appendix A).

Thus if AMAIN calls MTMRN before scheduling PROC (Figure 2-1c), we have the program scheduling hierarchy shown in Figure 2-1d.

```
FTN4
      PROGRAM AMAIN
      INTEGER PROC(3)
      DATA PROC/2HPR,2HOC,2H  /
        .
        .
        .
C**** CREATE NAME OF COPY OF PROC
C
      CALL MTMRN(PROC)
        .
        .
        .
C**** PROMPT USER FOR FILE NAME
        .
        .
        .
C**** QUEUE SCHED PROC WITH WAIT
C
      CALL EXEC(23,PROC)
        .
        .
        .
C**** DISPLAY RESULTS
        .
        .
        .
      STOP
      END
```

Figure 2-1c

```
FMG24 ---> AMA24 ---> PRO24


FMG42 ---> AMA42 ---> PRO42
```

Figure 2-1d

Now suppose that PROC is a segmented program with 2 segments called P1SEG and P2SEG. The scheduling hierarchy will be as in Figure 2-2. Note that PRO24 and PRO42 are both sharing the same program segments. Since program segments are read-only, both PRO24 and PRO42 can simultaneously use the same program segments, i.e. copies of program segments are not required.

```
FMG24 ---> AMA24 ---> PRO24 --+
                              +---> P1SEG
                              !
                              !
                              !
                              +---> P2SEG
FMG42 ---> AMA42 ---> PRO42 --+
```

Figure 2-2

Finally, suppose that PROC spends most of its time outputting to the list device. In this case, there is no advantage in having copies of AMAIN schedule copies of PROC, since only one copy of PROC can use the list device at one time. Thus AMAIN should schedule PROC without a prior call to MTMRN.

In summary, when a father program schedules a son, the father has the option of:

i) scheduling a son which may be shared with other fathers (via an EXEC Schedule call without a prior call to MTMRN)

ii) scheduling a personal copy of the son (via a call to MTMRN followed by an EXEC Schedule call)

Managing ID Segments of Scheduled Programs

In practice, it is not practical to have permanent ID segments for all possible copies of all programs in the system. One would like to keep all programs as type-6 files, and restore their ID segments only when required, and only for as long as required. Note that MTM does this automatically for the main program scheduled by FMGxx. The problem that remains is to create and release ID segments of EXEC Call scheduled programs. One solution is to have a procedure file which:

i) creates all the ID segments for programs scheduled by the main program

ii) runs the main program

iii) releases the ID segments created in i)

This procedure file can have the same name as the main program, provided it is stored on a cartridge other than the one containing the program's type-6 file. Furthermore, if the procedure file is higher in the cartridge list than the type-6 file, and an ID segment does not already exist for the program, then entering :RU,AMAIN will transfer control to the procedure file rather than the program.*

For the hierarchy in Figure 2-1b, the procedure file would have to:

i) create an ID segment for PROC (unless it already exists)

ii) run AMAIN

iii) release the ID segment for PROC, but ONLY IF EVERY OTHER MAIN PROGRAM WHICH SCHEDULES PROC IS DORMANT

For the hierarchy in Figure 2-1d, the procedure file would have to:

i) create an ID segment for a copy of PROC (corresponding to the user's terminal logical unit number)

ii) run AMAIN

iii) release the ID segment of the copy of PROC created in i)

For the hierarchy in Figure 2-2, the procedure file would have to:

i) create ID segments for P1SEG and P2SEG (unless they already exist)

ii) create an ID segment for a copy of PROC (corresponding to the user's terminal logical unit number)

iii) run AMAIN

iv) release the ID segment for the copy of PROC created in ii)

v) release the ID segments for P1SEG and P2SEG, but ONLY IF EVERY OTHER MAIN PROGRAM WHICH SCHEDULES PROC OR A COPY OF IT IS DORMANT

*This will have to be amended to: TR,AMAIN for RTE-IVB.

To simplify the process of creating such procedure files, 3 utility procedure files have been written:

i) /MTMRN Renames a program name using the conventions of MTM

ii) /MTMRP Creates a copy of a type-6 program using the conventions of MTM

iii) /MTMST Determines the status of a program and its copies

In addition, a procedure file called XMAIN shows how the above procedures can be used to handle all the possible combinations of scheduling hierarchies, and procedure file AMAIN shows how this is applied to the hierarchy of Figure 2-2. Listings of these files can be found in Appendix B.

NOTE

i) In the listings in Appendix B, it is assumed that all type-6 files reside on cartridge 2. Note that in file XMAIN, the program XMAIN is scheduled by the command :RU,XMAIN::2. The cartridge must be explicitly specified to prevent a recursive transfer to procedure file XMAIN.*

ii) For fastest response, the files /MTMRN, /MTMRP, and /MTMST, should be stored at the beginning of the first cartridge.

iii) Problems may be encountered if programs scheduled by XMAIN are still running when XMAIN terminates (e.g. if XMAIN schedules without wait), since the procedure file may try to release the ID segments of these programs.

iv) For HP1000 systems not operating under RTE-IV, the program scheduling features of MTM can be emulated by replacing the command :RU,XMAIN::2 in procedure file XMAIN with the following:

```
:TR,/MTMRP,XMAIN
:RU,1G
:RP,,1G
```

This will create an ID segment for a copy of the main program,run the copy, and then release the ID segment.

APPENDIX A:

MTMRN — Renames a Program Name Using the Conventions of MTM

```
FTN4
C*
      SUBROUTINE MTMRN(NSON)
C*
C* Renames a program name using the same conventions as MTM.          *
C*                                                                    *
C* The RTE-IV Multiple Terminal Monitor (MTM) manages ID segments so  *
C* that each user can have his own copy of a program.  MTM will       *
C* perform this action whenever the program to be run is a son of     *
C* FMGxx, and the program is a Type 6 FMGR file.  A copy of the prog- *
C* ram will be created with the last 2 characters being xx, and be    *
C* scheduled for execution to terminal xx.  (RTE-IV Programmer's      *
C* Reference Manual, Sec. 9-9).                                       *
```

```
C*                                                                   *
C* When a FATHER program passes the name of a SON to MTMRN, MTMRN exam-*
C* ines the last 3 characters of the name of the father.  If they are  *
C* 2 numerics followed by a space, MTMRN assumes that the father is a  *
C* copy of a program scheduled by MTM, and modifies the name of the    *
C* son by replacing its last 3 characters with those of the father,    *
C* and replacing any embedded spaces with .'s.  Otherwise the name of  *
C* the son is not changed.  Examples are:                              *
C*                                                                     *
C*             NAME PASSED  NAME RETURNED                              *
C*   FATHER     TO MTMRN     FROM MTMRN                                *
C*                                                                     *
C*   APROG      BPROG         BPROG                                    *
C*   APR20      BPROG         BPR20                                    *
C*   APR20      B             B..20                                    *
C*                                                                     *
C*                                                                     *
C* The father can then use this revised name for scheduling and        *
C* unscheduling a personal copy of the program, provided the required  *
C* ID segment already exists.                                          *
C*                                                                     *
C*                                                                     *
C*                          *********************                      *
C*                                                                     *
C* INPUT PARAMETERS:                                                   *
C*                                                                     *
C*  NAME  TYPE                  DESCRIPTION                            *
C*                                                                     *
C* NSON     I(3)     Name of son to be renamed                         *
C*                                                                     *
C*                                                                     *
C*                          *********************                      *
C*                                                                     *
C*                                                                     *
C* OUTPUT PARAMETERS:                                                  *
C*                                                                     *
C*  NAME  TYPE                  DESCRIPTION                            *
C*                                                                     *
C* NSON     I(3)     Name of MTM personal copy of son                  *
C*                                                                     *
C*                                                                     *
C*                          *********************                      *
C*                                                                     *
C*                                                                     *
C* EXTERNAL REFERENCES:                                                *
C*                                                                     *
C*  NAME  TYPE                  DESCRIPTION                            *
C*                                                                     *
C* PNAME  SUB        FORTRAN-IV library routine for getting the name of *
C*                   the currently executing program.                  *
C*                                                                     *
C*                                                                     *
C**********************************************************************
C
C
C*********************** LOCAL  VARIABLES ****************************
C
C
      INTEGER NFATHR(3), NSON(3)
C
```

```
C
C**** ASCII 0, 9, SPACE, PERIOD
C
      DATA   IA0/060B/, IA9/071B/, ISP/040B/, IPR/056B/
C
C
C******************** EXECUTABLE STATEMENTS ************************
C
C
C**** GET NAME OF FATHER
C
      CALL PNAME(NFATHR)
C
C
C**** GET LAST 3 CHARACTERS
C
      ICHAR4 = IAND( NFATHR(2) , 000377B )
      ICHAR5 = IAND( NFATHR(3) , 177400B ) / 000400B
      ICHAR6 = IAND( NFATHR(3) , 000377B )
C
C
C**** DON'T MODIFY NAME OF SON UNLESS LAST 3 CHARACTERS OF
C     FATHER ARE NUMERIC-NUMERIC-SPACE
C
      IF ( (ICHAR4 .LT. IA0) .OR. (ICHAR4 .GT. IA9) )  RETURN
      IF ( (ICHAR5 .LT. IA0) .OR. (ICHAR5 .GT. IA9) )  RETURN
      IF   (ICHAR6 .NE. ISP)                           RETURN
C
C
C**** REPLACE LAST 3 CHARACTERS OF NAME OF SON WITH THOSE OF FATHER
C
      NSON(2) = IOR( IAND(NSON(2),177400B) , ICHAR4 )
      NSON(3) = IOR(      ICHAR5*000400B   , ICHAR6 )
C
C
C**** REPLACE EMBEDDED SPACES WITH .'S
C
      ICHAR2 = IAND( NSON(1) , 000377B)
      ICHAR3 = IAND( NSON(2) , 177400B) / 000400B
C
      IF (ICHAR2 .EQ. ISP)
     :    NSON(1) = IOR( IAND(NSON(1),177400B) , IPR )
      IF (ICHAR3 .EQ. ISP)
     :    NSON(2) = IOR( IAND(NSON(2),000377B) , IPR*000400B )
C
C
      RETURN
      END
```

APPENDIX B:

Procedure Files for Managing ID Segments

/MTMRN — Renames a Program Name Using the Conventions of MTM

```
:IF,,EQ,,48
:**                                                                  *
:** /MTMRN replaces the 4th and 5th characters of 1G with the contents *
:** of 2G (containing a 2-digit logical unit number), and the 6th    *
:** character with a space.  Embedded spaces are replaced with .'s.  *
:**                                                                  *
:**                                                                  *
:** INPUT PARAMETERS:                                                *
:**                                                                  *
:** NAME TYPE           DESCRIPTION                                  *
:**                                                                  *
:**  1G    A    Name of program to be renamed                        *
:**  2G    I    2-digit LU to be inserted in 1G                      *
:**                                                                  *
:**                                                                  *
:** OUTPUT PARAMETERS:                                               *
:**                                                                  *
:** NAME TYPE           DESCRIPTION                                  *
:**                                                                  *
:**  1G    A    Renamed program name                                 *
:**                                                                  *
:**                                                                  *
:** LOCAL VARIABLES:                                                 *
:**                                                                  *
:** NAME TYPE           DESCRIPTION                                  *
:**                                                                  *
:**  5P    I    Intermediate values                                  *
:**                                                                  *
:**                                                                  *
:** EXTERNAL REFERENCES: None                                        *
:**                                                                  *
:*********************************************************************
:**
:****** REPLACE BLANKS IN 2ND AND 3RD CHARACTERS OF 1G WITH .'S
:**
:CA,5:P,-35P,AND,000377B
:IF,5P,NE,040B
:CA,-35:P,-35P,AND,177400B,OR,056B
:**
:CA,5:P,-34P,AND,177400B
:IF,5P,NE,020000B
:CA,-34:P,-34P,AND,000377B,OR,027000B
:**
:****** REPLACE 4TH CHARACTER OF 1G WITH 1ST DIGIT OF 2G
:**
:CA,5:P,2G,/,10,+,060B
:CA,-34:P,-34P,AND,177400B,OR,5P
:**
:****** REPLACE 5TH CHARACTER OF 1G WITH 2ND DIGIT OF 2G,
:**          AND 6TH CHARACTER OF 1G WITH A SPACE
:**
:CA,5:P,5P,-,060B,*,10
:CA,5:P,2G,-,5P,+,060B,*,000400B
:CA,-33:P,5P,OR,040B
::
```

/MTMRP — Creates an MTM Copy of a Type-6 Program

```
:IF,,EQ,,51
:**                                                                    *
:**                                                                    *
:** Creates a copy of a type 6 program using the conventions of MTM.  *
:** It is assumed that all type-6 files reside on cartridge 2.        *
:**                                                                    *
:**                                                                    *
:** INPUT PARAMETERS:                                                  *
:**                                                                    *
:** NAME TYPE              DESCRIPTION                                 *
:**                                                                    *
:**  1G    A    Name of type-6 program to be copied                   *
:**                                                                    *
:**                                                                    *
:** OUTPUT PARAMETERS:                                                 *
:**                                                                    *
:** NAME TYPE              DESCRIPTION                                 *
:**                                                                    *
:**  1G    A    Renamed program name                                   *
:**  2G    I    LU of user's terminal                                  *
:**  3G    A    Original program name                                  *
:**                                                                    *
:**                                                                    *
:** LOCAL VARIABLES:                                                   *
:**                                                                    *
:** NAME TYPE              DESCRIPTION                                 *
:**                                                                    *
:**  5P    I    Current severity code                                  *
:**                                                                    *
:**                                                                    *
:**                                                                    *
:** EXTERNAL REFERENCES:                                               *
:**                                                                    *
:** NAME                   DESCRIPTION                                 *
:**                                                                    *
:** /MTMRN     Renames a program name using the conventions of MTM.   *
:**                                                                    *
:**********************************************************************
:**
:****** SAVE ORIGINAL PROGRAM NAME IN 3G
:**
:CA,3,1G
:**
:****** CREATE NAME OF PROGRAM COPY
:**
:TR,/MTMRN,1G,0G
:**
:****** CREATE ID SEGMENT FOR PROGRAM COPY
:**
:RN,3G::2,1G
:CA,5:P,7P
:SV,4
:RP,1G::2
:SV,5P
:RN,1G::2,3G
::
```

/MTMST — Determines the Status of a Program and its Copies

```
:IF,,EQ,,59
:**                                                                      *
:**                                                                      *
:** /MTMST tests the status of a program and each possible MTM           *
:** generated copy to see if at least one is not dormant.  It is         *
:** assumed that there are 4 interactive terminals whose logical unit    *
:** numbers are 20, 24, 41, and 42.                                      *
:**                                                                      *
:**                                                                      *
:** INPUT PARAMETERS:                                                    *
:**                                                                      *
:** NAME TYPE            DESCRIPTION                                      *
:**                                                                      *
:**  1G    A    Name of program whose status is desired                  *
:**                                                                      *
:**                                                                      *
:** OUTPUT PARAMETERS:                                                   *
:**                                                                      *
:** NAME TYPE            DESCRIPTION                                      *
:**                                                                      *
:**  6P    I    Status of program and its copies:                        *
:**               18  Program or a copy is not dormant                   *
:**               nn  Program and every copy is dormant                  *
:**                                                                      *
:**  1G    A    Name of program or 1st copy found which is not dormant;  *
:**             Name of last copy tested otherwise                       *
:**                                                                      *
:**                                                                      *
:** LOCAL VARIABLES:                                                     *
:**                                                                      *
:** NAME TYPE            DESCRIPTION                                      *
:**                                                                      *
:**  2G    I    LU's of interactive terminals                            *
:**  5P    I    Current severity code                                    *
:**                                                                      *
:**                                                                      *
:**                                                                      *
:** EXTERNAL REFERENCES:                                                 *
:**                                                                      *
:** NAME                 DESCRIPTION                                      *
:**                                                                      *
:** /MTMRN     Renames a program name using the conventions of MTM.      *
:**                                                                      *
:************************************************************************
:**
:****** CHECK STATUS OF PROGRAM
:**
:CA,5:P,7P
:SV,4
:RP,,1G
:SV,5P
:IF,6P,NE,18
::
:**
:****** CHECK STATUS OF COPY 20
```

```
:**
:TR,/MTMRN,,20
:CA,5:P,7P
:SV,4
:RP,,1G
:SV,5P
:IF,6P,NE,18
::
:**
:****** CHECK STATUS OF COPY 24
:**
:TR,/MTMRN,,24
:CA,5:P,7P
:SV,4
:RP,,1G
:SV,5P
:IF,6P,NE,18
::
:**
:****** CHECK STATUS OF COPY 41
:**
:TR,/MTMRN,,41
:CA,5:P,7P
:SV,4
:RP,,1G
:SV,5P
:IF,6P,NE,18
::
:**
:****** CHECK STATUS OF COPY 42
:**
:TR,/MTMRN,,42
:CA,5:P,7P
:SV,4
:RP,,1G
:SV,5P
:IF,6P,NE,18
::
::
```

XMAIN — Sample Procedure File for Managing ID Segments

```
:IF,,EQ,,43
 :**                                                                 *
 :** Manages ID segments of programs scheduled by XMAIN.             *
 :**                                                                 *
 :** In this example, it is assumed that XMAIN schedules:            *
 :**                                                                 *
 :**      i) program ASHAR                                           *
 :**     ii) segmented program BSHAR with segments B1SEG and B2SEG   *
 :**    iii) a copy of segmented program ACOPY with segments A1SEG and *
 :**         A2SEG                                                   *
 :**     iv) a copy of program BCOPY                                 *
 :**     iv) a copy of segmented program CCOPY with segments C1SEG and *
 :**         C2SEG                                                   *
```

```
:**                                                                    *
:** It is further assumed that:                                        *
:**                                                                    *
:**     i) only XMAIN and its copies schedule BSHAR and copies of      *
:**        ACOPY                                                       *
:**    ii) XMAIN and YMAIN and their copies schedule ASHAR and copies  *
:**        of CCOPY                                                    *
:**                                                                    *
:** For HP1000 systems not operating under RTE-IV, the command         *
:** :RU,XMAIN::2  should be replaced with:                             *
:**                                                                    *
:**     :TR,/MTMRP,XMAIN                                               *
:**     :RU,1G                                                         *
:**     :RP,,1G                                                        *
:**                                                                    *
:**                                                                    *
:** EXTERNAL REFERENCES:                                               *
:**                                                                    *
:** NAME                  DESCRIPTION                                  *
:**                                                                    *
:** /MTMRP    Creates an MTM copy of a type-6 file                     *
:** /MTMRN    Renames a program name using the conventions of MTM      *
:** /MTMST    Determines the status of a program and its copies        *
:**                                                                    *
:**                                                                    *
:*********************************************************************
:**
:****** SAVE CURRENT SEVERITY CODE AND CHANGE TO 4
:**
:CA,5:P,7P
:SV,4
:**
:****** CREATE ID SEGMENTS FOR 1) ALL SCHEDULED SHARED PROGRAMS
:******                        2) PROGRAM SEGMENTS OF ALL SCHEDULED
:******                           PROGRAMS
:**
:**:RP,ASHAR::2
:**:RP,BSHAR::2
:**
:**
:**:RP,B1SEG::2    (Program segment 1 for program BSHAR)
:**:RP,B2SEG::2    (Program segment 2 for program BSHAR)
:**
:**:RP,A1SEG::2    (Program segment 1 for program ACOPY)
:**:RP,A2SEG::2    (Program segment 2 for program ACOPY)
:**
:**:RP,C1SEG::2    (Program segment 1 for program CCOPY)
:**:RP,C2SEG::2    (Program segment 2 for program CCOPY)
:**
:**
:****** RESTORE SEVERITY CODE
:**
:SV,5P
:**
:****** CREATE ID SEGMENTS FOR ALL SCHEDULED PROGRAM COPIES
:**
:**:TR,/MTMRP,ACOPY
:**:TR,/MTMRP,BCOPY
:**:TR,/MTMRP,CCOPY
```

```
:**
:****** RUN THE MAIN PROGRAM
:**
:RU,XMAIN::2
:**
:****** REMOVE ID SEGMENTS OF PROGRAM COPIES
:**
:**:TR,/MTMRN,ACOPY,0G
:**:RP,,1G
:**
:**:TR,/MTMRN,BCOPY,0G
:**:RP,,1G
:**
:**:TR,/MTMRN,CCOPY,0G
:**:RP,,1G
:**
:****** REMOVE ID SEGMENTS OF SHARED PROGRAMS, AND ID SEGMENTS OF ALL
:****** PROGRAM SEGMENTS, ONLY IF EVERY OTHER PROGRAM THAT SCHEDULES
:****** THEM IS DORMANT
:**
:****** CASE 1)  PROGRAM XMAIN IS THE ONLY PROGRAM THAT SCHEDULES
:**              BSHAR AND A COPY OF SEGMENTED PROGRAM ACOPY
:**
:**:TR,/MTMST,XMAIN
:**:IF,6P,EQ,18,8
:**
:**:RP,,BSHAR
:**:RP,,B1SEG
:**:RP,,B2SEG
:**
:**:RP,,A1SEG
:**:RP,,A2SEG
:**
:****** CASE 2)  PROGRAMS XMAIN AND BMAIN BOTH SCHEDULE ASHAR AND
:**              A COPY OF SEGMENTED PROGRAM CCOPY
:**
:**:TR,/MTMST,XMAIN
:**:IF,6P,EQ,18
:**:TR,/MTMST,BMAIN
:**:IF,6P,EQ,18,6
:**
:**:RP,,ASHAR
:**
:**:RP,,C1SEG
:**:RP,,C2SEG
::
```

AMAIN — Procedure File for Managing ID Segments in Fig. 2-2

```
:IF,,EQ,,24
:**                                                                  *
:** Manages ID segments of programs scheduled by AMAIN.              *
:**                                                                  *
:** In this example, it is assumed that AMAIN schedules a copy of    *
:** segmented program PROC with 2 segments, P1SEG and P2SEG.  It is  *
:** further assumed that only AMAIN and its copies schedule copies of *
:** PROC.                                                             *
```

```
:**                                                                        *
:**                                                                        *
:** EXTERNAL REFERENCES:                                                   *
:**                                                                        *
:** NAME                  DESCRIPTION                                      *
:**                                                                        *
:** /MTMRP    Creates an MTM copy of a type-6 file                         *
:** /MTMRN    Renames a program name using the conventions of MTM          *
:** /MTMST    Determines the status of a program and its copies            *
:**                                                                        *
:**                                                                        *
:*************************************************************************
:**
:****** SAVE CURRENT SEVERITY CODE AND CHANGE TO 4
:**
:CA,5:P,7P
:SV,4
:**
:****** CREATE ID SEGMENTS FOR 1) ALL SCHEDULED SHARED PROGRAMS
:******                        2) PROGRAM SEGMENTS OF ALL SCHEDULED
:******                           PROGRAMS
:**
:RP,P1SEG::2
:RP,P2SEG::2
:**
:****** RESTORE SEVERITY CODE
:**
:SV,5P
:**
:****** CREATE ID SEGMENTS FOR ALL SCHEDULED PROGRAM COPIES
:**
:TR,/MTMRP,PROC
:**
:****** RUN THE MAIN PROGRAM
:**
:RU,AMAIN::2
:**
:****** REMOVE ID SEGMENTS OF PROGRAM COPIES
:**
:TR,/MTMRN,PROC,0G
:RP,,1G
:**
:****** REMOVE ID SEGMENTS OF SHARED PROGRAMS, AND ID SEGMENTS OF ALL
:****** PROGRAM SEGMENTS, ONLY IF EVERY OTHER PROGRAM THAT SCHEDULES
:****** THEM IS DORMANT
:**
:TR,/MTMST,AMAIN
:IF,6P,EQ,18,3
:**
:RP,,P1SEG
:RP,,P2SEG
::
```

AN INTERFACE TO IMAGE

Mike Wells/Technical Analysis Corporation

THE PROBLEM

Twenty to thirty terminal devices, all screaming 'Response Time', while executing a menu of fifty to seventy-five 'custom' transactions which compete for a fifteen to twenty-five megabyte IMAGE 1000 Data Base.

The transactions are to be segments to a mainline which will handle terminal I/O, screen maintenance and transaction management for the programmer. The programmers will be a small army of junior people with no data base experience. The system is dual and is to be redundant with automatic switchover.

How can I provide the programmers with instant data base experience? Not in terms of how to write a DBGET, but in terms of contention, locking and unlocking the data base, broken chains, logging of appropriate information, etc.?

How can I provide the programmer with the guarantees he or she needs in order to feel secure that his updates made it to the disc without someone else modifying them?

How can I provide decent response time to the terminals when I know that there are transactions which may have to hold large volumes of data in a static state while they process for four or five minutes?

How can I collect just the information I need to keep both my redundancy and recovery subsystems competent without having to log every find, get and information call?

A SOLUTION

Interface Image.

1. Let the programmer communicate with a program wholly responsible for answering the questions raised above.
2. Leave the data base open all the time to any user who can get a partition.
3. Lock only that data which a given user needs at a given time.
4. Prioritize data base services to the users in the system.
5. Reduce the number of root files to one.
6. Log information, transparent to the user, in a form consistent with redundancy and recovery capabilities.

IMPLEMENTATION OF OUR SOLUTION

What follows is a narrative description of a program we call the Data Base Administrator. In its original form it was entitled 'Image Interface Objectives'. A paragraph was written around each objective, and the final version has become a tool to familiarize programmers with the features of the Administrator.

As I come in contact with more and more HP OEMs and end users, I find that there are quite a few system managers with these same problems. With this in mind, I have contributed this information to the Communicator with the hopes that the product of my experience with these problems may prove useful to others.

A DATA BASE ADMINISTRATOR FOR HP'S IMAGE/1000

Scope

The Data Base Administrator acts as an interface between application programs and Hewlett Packard Image 1000. The purpose of the interface is to improve system performance where multiple, transaction-oriented, application programs are competing for data base resources and also to provide a central point for the collection of information needed to provide Recovery and/or Redundancy subsystems. A by-product of this interface is ease of data base oriented application programming.

Description

The Data Base Administrator is a mainline program, associated segments and a family of application program subroutines. The Administrator can provide for support of up to four data bases, serviced by one to four copies of the mainline program. The number of users which can concurrently access the data base is limited only by the number of available class numbers.

The subroutines provided with the Data Base Administrator are a direct replacement for the Image 1000 subroutines. For example, Image 1000 subroutine DBGET is replaced by Administrator subroutine DLGET. The administrator subroutines usually require a few more parameters than their Image counterpart. These parameters are used to store variable information, such as chain pointers, in the user's partition and also to control features of the Administrator not available in Image. The application program calls an Administrator subroutine which formats the parameters into a form suitable for the Administrator mainline and segments. The subroutine then communicates with the Administrator mainline via class I/O in order to perform the requested operation.

Some of the Image subroutines are replaced by 'dummy' subroutines or subroutines which perform alternate, but related, operations. For example, Image subroutine DBLCK is replaced by 'dummy' subroutine DLLCK since when using the Administrator it is not necessary to lock the data base in order to provide integrity of updates. Leaving the data base available to all users at all times improves total system performance while easing the responsibilities imposed on the application programmer who wants to update the data base. A second example is the DLOPN subroutine. This subroutine, which replaces the Image DBOPN subroutine, prepares the application program to interface with the Administrator. It is no longer necessary to open the data base because it is always open to the Administrator.

The Administrator contains a subroutine which has no Image counterpart. DLAOL is a subroutine which gives the programmer the ability to prearrange the Image 'open file list'. This can improve performance in an application where a certain group of data sets have the highest utilization.

The application program must contain a copy of the Administrator subroutine DHAND which is the user's interface to the Administrator. However, the applicaton program contains no Image subroutines, Root File or data base related Data Control Blocks. Having a single copy of the Root File which describes a data base is a distinct advantage in reducing the 'cratering' effect related to a system 'crash'.

The Data Base Administrator provides record level locking to the Application Program. This allows a programmer to 'hold' information in a static state when necessary and also ensures the programmer that information obtained from the data base is current and that it cannot be changed until it is unlocked. This is done with the data base unlocked, which means that a program is only denied data base access when it attempts to update a record currently being updated by another program. This is an improvement over being denied data base access during each update by every program executing. Record locking is provided in two modes: Wait and No Wait. In the Wait Mode a program which is attempting to lock a locked record is automatically suspended until the record is unlocked. This has the effect of making the wait time transparent to the programmer. In No Wait mode, a unique status is returned to the program if a desired record is locked.

The Administrator requires the first three words of each data base record in order to provide record level locking. A locked record can be read by any program; however, a record cannot be updated or deleted unless it is previously locked to the program requesting the update.

Allowing programs read-only access to locked records provides the user with the ability to write data base query programs that have little effect on total system performance.

Record level locking provides an additional feature usually required by transaction oriented applications, the ability for a given program to 'hold' a large volume of records in a static state while processing, without impacting other programs that are attempting to read and/or update other records. Normally this type of operation could only be performed in a batch environment since the data base would have to be locked.

The Data Base Administrator logically deletes a record when a record deletion request is received. This leaves chains intact for other users. Logically deleted records are transparent to the application program since any attempt to access them will have the same result as if the record did not exist. For example, if the Administrator detects a logically deleted record during a chain read, it will bypass it and go to the next record in the chain without user intervention.

Logical deletions reduce the problems faced by an application programmer. An application programmer, using Image directly, has two concerns when using a chain read. The first is that a record in the chain could be deleted by another program resulting in a 'broken' chain. This possibility generates a difficult error handling situation. The second, and more serious problem, is that after the chain is broken, and before the program has time to detect it, Image could put a record into the same position in the data base. In this case no error would result to indicate a 'broken' chain. Instead, the program would begin to read into a chain of records different from those that were started with. Logical deletions remove these concerns since the programmer will never need to concern himself with 'broken' chains.

However, logical deletions generate the requirement for a deletion program which physically deletes the logically deleted records. This program should accept a file of data set names as an argument. The deletion program then takes exclusive use of the data base and physically deletes records from the indicated data sets that are 'marked' for delete. Since the data base is unavailable during this time it is suggested that on a daily basis, after an orderly shutdown of real time operations, a file of names which include high utilization data sets be issued to the deletion program. Then a file with all the data set names can be issued to the deletion program weekly or monthly as dictated by the needs of a given system.

The mainline program and segments associated with the Data Base Administrator require configuration to a particular installation. Data set names, key lengths, record lengths, etc. as well as available partition size and the establishing of an FMP file for the Administrator's use are examples of the configuration. The Administrator includes a JCL procedure executed during system initialization which prepares the Administrator to service program requests. The mainline program operates at priority forty, or below, and at a higher priority (lower number) than any program requesting its services.

The mainline program of the Administrator services requests based on the priority of the requesting programs in such a way that high priority programs are serviced first. The mainline program and associated segments service the request and pass the result back to the appropriate Administrator subroutines in the user's partition. The subroutines then format the information and return it to the user. Successful and error returns are processed by the user as if the program was interfacing directly with Image. However, the Administrator can return a few error codes that Image does not contain.

Each request received by the Administrator mainline and the result of each request are written to a logging class number assigned to the Administrator. The request and its result are appended to form one logging buffer per operation. The information logged is in the form of a journal in that it contains the parameters issued by the request, the result of the request, the requesting program and the before and after image of each update. Logging information should be received by a program executing at a higher priority (lower number) than the Administrator mainline.

The logging features of the Administrator are controlled by a utility program and the program break feature of RTE. The utility, which is provided with the Administrator, interactively or programmatically, enables or disables logging. The utility also has the ability to assign a 'hard copy' device for the Administrator mainline. Using the program break features of RTE, an operator can reverse the status of hard copy logging. That is, if the Administrator is not currently displaying a hard copy log, breaking the Administrator mainline will begin the hard copy logging to the last device configured via the utility. Breaking the mainline again will stop the hard copy log. The information displayed is a valuable debugging tool.

Another logging feature is provided as part of the enable logging command. The option to log 'all' or 'partial' is available. Partial logging involves all information that affects the image of the data base. If the option to log all is elected, every request processed by the Administrator will be logged.

Once initialized, the Data Base Administrator need not be terminated. In order to pause the Administrator temporarily, a command can be issued to it which will cause it to close any open data bases and suspend. An RTE 'GO' command can then be used to resume Administrator services. While in a paused condition, requests for service can still be issued to the Administrator. Each user making a request for Administrator services, while it is suspended, will likewise be suspended until the Administrator is resumed.

The Administrator has two execution modes: Batch and Real Time. Real Time mode is the normal mode of execution for the Administrator. Batch mode is used to alter the record locking operation of the Administrator. That is, records requested with lock are not actually locked. This mode is provided so that a batch program which is ensured of exclusive use of the Administrator can bypass the overhead related to record locking. The batch program should still use the record locking features of the Administrator in order to provide accurate logging of data base updates, but the resulting overhead will be bypassed.

It should be noted that the Administrator is primarily intended for multiple user operation (Real Time). A batch program will execute more efficiently if it interfaces directly with Image. However, due to the nature of certain system applications, batch programs cannot always bypass the Administrator. It is strongly recommended that any batch programs which update the data base and execute concurrently with Real Time operations use the Administrator to perform updates. A batch program in this category can be coded to use the Administrator, and the execution mode of the Administrator can be used to control the resulting overhead. That is, if the batch program is executing concurrently with Real Time, record locking will be performed. However, if the batch program is executing exclusively, the operator can change the execution mode of the Administrator in order to improve performance. The batch program is coded and executes the same in both modes.

The execution mode of the Administrator can be set during initializaion and altered any time the Administrator is resumed after a pause.

MIRROR DATA BASES AND THE DBA

The Administrator lends itself to an application which calls for the maintenance of a mirror data base on a backup computer system. To implement this concept, the user should execute the required number of Administrators in both the primary and backup CPUs. Each copy of the DBA in the primary should have logging enabled in either all or partial mode. Partial logging should be adequate for most applications and is recommended since this will reduce the volume of CPU to CPU communication to be performed.

As described earlier, each DBA will log a buffer to a logging program which will receive the buffers and write them to the appropriate backup subsystem. The backup system will execute a receiving program which will accept the log buffers from the primary system and pass the parameter portion of them to the appropriate copy of the DBA which is executing on the backup system. The only difference between the Administrators running on the primary and those running on the backup system is the logging mode. Those Administrators which are running on the backup system must be executed with logging disabled.

To ensure that the data bases on the backup system match those on the primary, the receiving program on the backup system must wait for a response from the DBA for each parameter portion of a logging buffer sent to a DBA and then compare the result of that operation with the result portion of the logging buffer. If the two do not match, it is an indication that there is a discrepancy between the data base on the primary and the mirror data base on the backup system. Depending on the method of communication used between the primary and the backup systems, a handshake between the primary system logging program and the backup system receiving program may be needed.

A system manager may decide that, due to the volume of information being logged to the backup receiving program, logging buffers should be buffered on the backup system disc prior to issuing them to the appropriate DBA. In this case it is highly recommended that the disc unit used for buffering not be the same unit on which the data base(s) reside.

RECOVERY SUBSYSTEMS AND THE DBA

In a system configuration where there is only one CPU, or as an additional backup to a Primary/Backup system configuration, the program responsible for receiving and writing the logging buffers supplied by the DBA in the primary or stand-alone system could write the logged information to a dedicated magnetic tape unit.

A Recovery program could then be written which would read and process this information through the DBA in order to recover the system from a total failure.

SUMMARY

The Administrator has proven very useful to us as a tool to tame Image/1000. I hope that the reader finds this information useful.

If you have a similar solution or similar problems, I would enjoy corresponding with you to share your experience.

Mike Wells
Manager, Systems Development
TECHNICAL ANALYSIS CORPORATION
120 W. Wieuca Road, N.E.
Atlanta, Georgia 30342

USING DS/1000 AND RTE-IVA TO ACHIEVE VIRTUAL PERIPHERALS

Jean-Luc de Schutter/Distrigaz

[**Editor's Note:** Due to the lack of resources and time, I was not able to completely test this scheme. However, all the programs compile and run, and Mr. De Schutter is using the set-up on an RTE-IVA system in Belgium.]

When I bought a second disc-based RTE system to link to my old DOS and RTE-II system, I wanted to have "friendly" access to all the application programs and data files in both nodes without having separate copies for each system. The best solution seemed to be to construct a scheme of virtual peripherals. In designing my scheme there were two facts I considered:

a) the modem link between the systems is slow enough that speed is not a relevant consideration.

b) there is no restriction limiting the number of nodes.

The problem is presented in figure 1. A user sitting at node 2 wants to be virtually connected to system 1 (where program and data reside). To perform this access he starts the monitor VIRT. This in turn will schedule a second program in node 1, VIRTT (notice double T). VIRTT initializes a dummy driver, DVV05. DVV05 simulates an interrupt (like hitting any key on a terminal). This interrupt starts a session in node 1 on a LU attached to DVV05. The programs (PRMPT,FMGR, etc.) scheduled after this virtual interrupt will make requests to logical units associated with DVV05. DVV05 wil briefly analyze the requests and, very importantly, localize the I/O buffers in physical memory (i.e. calculate the physical page where they reside).

Finally, DVV05 passes the requests directly to the application program VIRTT. Now, VIRTT moves the request into its own program area, translates to a DEXEC, and performs I/O on peripherals located at node 2 (user terminal). As soon as the DEXEC is finished, VIRTT restarts the driver. DVV05 completes its dummy I/O and the program that made the request can execute further.

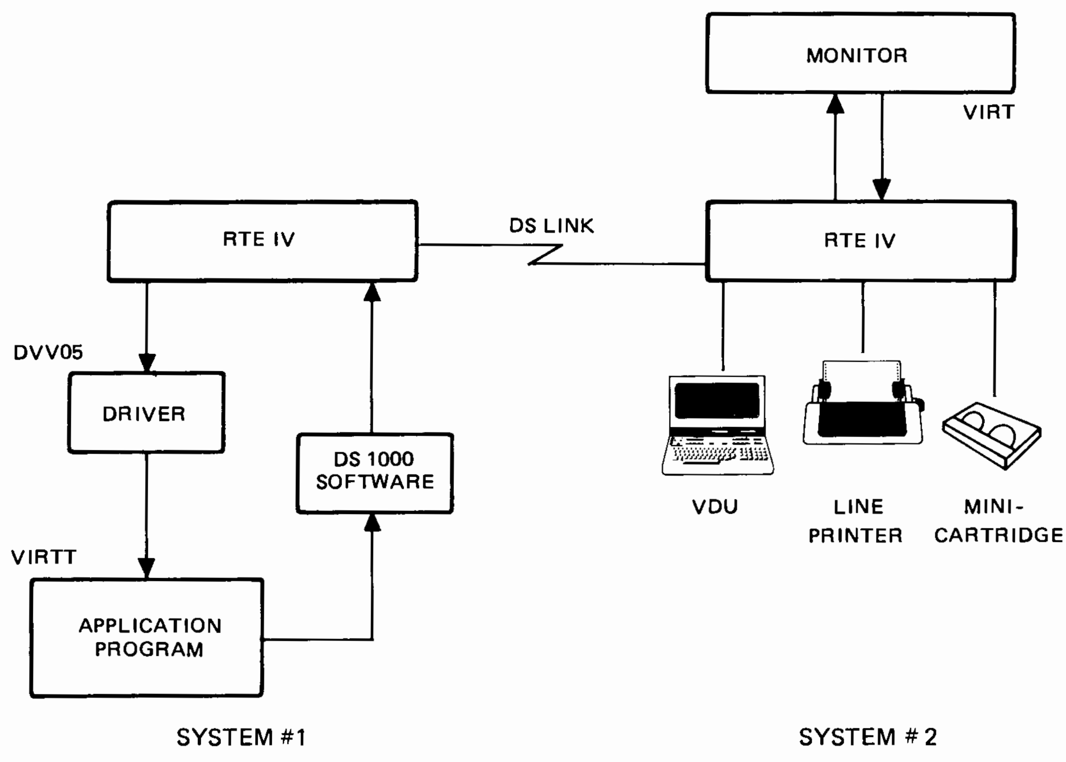


Figure 1

All these operations are very simple; yet sitting at a terminal and running "virtually" anything at another node gives you a tremendous feeling. The rest of this article details the technical aspects of VIRT and VIRTT, and explains the set-up of DVV05.

DVV05

DVV05 is set-up at generation time as follows:

PROGRAM INPUT PHASE

```
-
REL,%DVV05,B
-
```

EQUIPMENT TABLE ENTRY

```
-
EQT,12?
57,DVV05,B
-
```

DEVICE REFERENCE TABLE

```
-
34 = EQT ?
12,1
35 = EQT ?
12,2
36 = EQT ?
12,3
-
-
39 = EQT ?
12
-
```

INTERRUPT TABLE

```
-
57,PRG,PRMPT
```
(This is mandatory — DVV05 must have a program — to schedule on interrupt.)

The driver has two special calls. A call with subfunction 3700B passes the ID segment of VIRTT to the driver. This is used later by the driver to schedule VIRTT. The same call with a negative ID address clears the driver and stops the operations. The second special call is a control call with subfunction 3600B. This is used to ask the driver to simulate an interrupt and schedule PRMPT.

When receiving a normal request the driver locates the memory page where the buffer starts. Then VIRTT is scheduled immediately. Although the driver has the S bit set (driver handles time out), EQT 14 and 15 are set to zero. Therefore, the driver remains busy forever.

VIRTT

VIRTT is divided into two parts, the initialization section and the remote I/O section. The initialization section performs the following tasks:

a) stores the DS LU and the LU's linked with DVV05 in an array (LU(5));

b) initializes DVV05 and gets its EQT address; and

c) schedules PRMPT.

There is one EQT handled by VIRTT and DVV05, and each LU corresponds to a different subchannel of this EQT. Because VIRTT is initialized with five schedule parameters this version can handle 4 peripherals (one parameter indicates the node number). If the generation is done as specified in the DVV05 section, the four LU's passed to VIRTT (and stored in LU(2-5) would be 39,34,35, and 36.

The second task for VIRTT is execution of remote I/O. When rescheduled by DVV05, VIRTT analyzes the request. A control call is performed immediately (with special care for class request).

To perform a normal I/O request (i.e. read or write) VIRTT calls the routine BMAP to modify the user map in order to put the I/O buffer in the program area (note: the principles of BMAP are explained in Volume III, Issue 2, pp. 24-30, of the Communicator/1000.) Now we can perform the remote I/O paying a little attention to DS specific errors (e.g. time-out). When I/O is finished, VIRTT updates DVV05's EQT by passing back the status and the transmission log, or possibly an I/O error.

VIRTT then puts small negative values in EQT's 14 and 15 to force time-out completion and the driver is restarted by RTE. Next VIRTT goes dormant saving resources, ready for a new call. Note that VIRTT uses subroutines REMEX, BMAP and MWF. They are self explanatory and perform functions that are not available in FORTRAN.

VIRT: THE MONITOR PROGRAM

Actually, one may avoid using this program by using REMAT to start VIRTT. However, it is more convenient to use VIRT since it does some additional housekeeping. For example, VIRT sets the time-out, disables the terminal (this must be done again in the WELCOM file because FMGR re-enables the terminal), and regularly checks to see if the remote FMGR is running. If the remote FMGR is not running, VIRTT asks the user if he wishes to continue. If the remote FMGR is lost, the present version of VIRT allows the user to execute one last system command on the remote system (e.g. RU,WHZAT or RU,FMGR).

CONCLUSION

This is a set of programs for your sleepless nights because it can be improved in several different ways:

a) Replacing remote EXEC calls by program-to-program communication would improve speed and remove the burden of DS time-outs.

b) Allowing more than one set of virtual terminals per node without duplicating the driver and VIRTT would improve the capabilities of the scheme.

c) Adapting this set-up to RTE-IVB with its flexible logical unit and session monitor scheme would be just fantastic!

```
FTN4
      PROGRAM VIRTT (3,49),  JLD AUG 1979 REV.A
C
C  THIS PROGRAM IS AN EXAMPLE AND SHOULD BE ADAPTED TO FIT
C  THE LU'S AND NODES OF YOUR SYSTEM
C
C
C VIRTUAL TERMINAL HANDLER
C WORKS IN CONJUNCTION WITH DVV05
C MAX ONE VIRTUAL TERMINAL PER NODE ALLOWED
C THIS PROGRAM REQUIRES TWO FULL PAGES OF ADDITIONAL MEMORY
      IMPLICIT INTEGER (A-Z)
      DIMENSION IPAR(5),LU(5),MYNAM(3),EQT(15),MAP(32)
      DIMENSION BUFR(20)
      DATA IDUM/31/
C IDUM IS LU FOR DVV05
      CALL RMPAR(IPAR)
C
C MAKE ALL SCHEDULE PARAMETERS NEGATIVE
C 1> DESTINATION NODE
C 2> DEST. LU IF REQUEST LOCALLY MADE FOR SUBCHANNEL 0
C      ( = REMOTE LU )
C 3> IDEM SUBCH. 1
C 4> IDEM SUBCH. 2
C 5> IDEM SUBCH. 3
C
      NODE=-IPAR
9     FORMAT(5K10)
      DO2 I=2,5
      LU(I)=-IPAR(I)
      IF(IPAR(I).GE.0) GOTO 999
2     CONTINUE
C GET ID ADDRESS
      MYID=IGET(1717B)
      CALL PNAME(MYNAM)
C INITIALIZE DRIVER
      CALL EXEC (3,IDUM+3700B,MYID)
C GET EQT ADDRES
      IAB=IGET(1652B)+(IDUM-1)
      EQTN=IAND(IGET(IAB),77B)
      EQTAD=IGET(1650B)+(EQTN-1)*15
C SCHEDULE PRMPT
3     CALL EXEC(3+100000B,IDUM+3600B)
      GOTO 3
C STOP SAVING RESOURCES
1     CALL EXEC (6,0,1,99,99,99,99,99)
```

```
      CALL RMPAR(IPAR)
C
C  (HERE WE ARE STARTED BY DVV05)
C  RESTART PARAMETERS
C  1> EQT ADDRESS FOR CHECKING PURPOSE
C     NEGATIVE  CLEAR THE WHOLE PROCESS (ABNORMAL CONDITION)
C  2> ADD OF EQT6
C  3> ADD OF EQT7
C  4> ADD OF EQT8
C  5> PHYSICAL PAGE  OR CONTROL REQUEST PARAMETER
C
      IF(IFBRK(I))999,5
5     IF(IPAR.EQ.EQTAD) GOTO 55
C FORGET LOCAL ACCIDENTAL CALL
      IF(IPAR.GT.0) GOTO 1
      GOTO 999
55    DO 51 I=2,4
51    IPAR(I)=IGET(IPAR(I))
C MOVE EQT IN PROG AREA
      CALL MWF(EQTAD,EQT,15)
C RESET ERROR FLAG AND COMPUTE ACTUAL LU
C  (DEPENDS ON SUBCHANNEL)
      ERROR=0
      ACTLU=LU((IAND(EQT(4),3700B)/64)+2)
C CHECK FOR CONTROL CALL
      IF(IAND(EQT(6),3).EQ.3)GOTO100
C CORE LOCK AND MOVE BUFFER PAGE(5) IN PROGRAM AREA
      CALL EXEC(22,1)
C
C  HERE WE MODIFY THE USER MAP AND WE MAY RUN INTO TROUBLE
C  IF WE ARE SUSPENDED OR IF RTE RECOMPUTES OUR MAP
      CALL BMAP(IPAR(5),IPAR(4) ,IPAR(3) ,ADDR,ERROR)
      ERCNT=0
      IF(ERROR)6,6,900
C PERFORM REMOTE EXEC CALL
6     CALL REMEX(NODE,EQT(6),ADDR,EQT(8),ACTLU,ERROR)
      CALL ABREG(A,B)
      IF (ERROR)8,8,7
8     CALL EXEC(22,0)
      IP=0
      IF(IAND(EQT(6),3).GT.1) GOTO 88
      CALL DEXEC(NODE,13,ACTLU,IP)
88    IP=IAND(IP,177B)
      I=IGET(EQTAD+4)
      IP=IOR(I,IP)
      CALL IPUT(EQTAD+4,IP)
C SAVE TRANSMISSION LOG FOR DRIVER IN EQT12
      CALL IPUT(EQTAD+11,B)
C SET TIME-OUT TO -1 IN DRIVER EQT TO FORCE CONTINUATOR ENTRY
      CALL IPUT(EQTAD+13,-2)
      CALL IPUT(EQTAD+14,-2)
      GOTO1
C ELIMINATE DS SPECIFIC ERRORS (TO STOP FROM GETTING STUCK
C    BECAUSE OF A DS TIME-OUT)
7     IF(A.NE.2HDS) GOTO 8
      ERCNT=ERCNT+1
C   TRY AGAIN FOR +/- 10 MINUTES
      IF(ERCNT.LT.20) GOTO 6
```

```
C STOP AND CLEAR THE DRIVER
900    CALL DEXEC(NODE,2,LU(2),23H/VIRTT:MEMORY TOO SMALL,-23)
 999    CALL IPUT(EQTAD+13,-2)
       CALL IPUT(EQTAD+14,-2)
       CALL EXEC(3,IDUM+3700B,-1)
       CALL EXEC(6)
C      CHECK FOR CLEAR CALL
100    IF(IAND(EQT(6),3700B))101,101,110
C CLEAR CALL COMPLETES THE DRIVER IMMEDIATELY
C SYSTEM ALOWS ONE SECOND FOR THAT JOB
 101    CALL IPUT(EQTAD+13,-5)
       CALL IPUT(EQTAD+14,-5)
110    CN=IOR(IAND(EQT(6),3700B),ACTLU)
       CALL DEXEC(NODE,3,CN,IPAR(5))
       CALL ABREG(A,B)
       GOTO8
       END
       END$
C
C
C
```

```
FTN4
      PROGRAM VIRT (3,90), JLD SEP 1979  BRUSSELS
C
C   THIS PROGRAM IS AN EXAMPLE WHICH SHOULD BE ADAPTED TO FIT
C   YOUR SPECIFIC NEEDS.  NODE AND LUREM SHOULD REFLECT THE
C   VALUES IN YOUR SYSTEM.
C  IT IS USED TO SCHEDULE VIRTT (TERMINAL HANDLER)
      DIMENSIONLU1(3),NFMG(5),IMES(20),ITO(20)
      DATA NODE/500/,LUREM/31/
C  CONFIGURE DATA STATEMENT BEFORE RUN
C  LUREM IS LU USED BY DVV05 AT REMOTE NODE
      LU=LOGLU(LU)
C   CHECK TO SEE THAT VIRTT IS DORMANT
1     CALL DEXEC(NODE,99+100000B,6HVIRTT ,ISTAT)
      GOTO 1000
1001  IF(ISTAT.NE.0)GOTO1000
      IT=1000
C
C   DISABLE INTERRUPT AND SET TIME OUT
      CALL EXEC(3,LU+2100B)
      CALL EXEC(3,LU+2200B,15000)
C CONSTRUCT REMOTE FMGR NAME (CORRESPONDING TO LUREM)
C AND OTHER PARAMETERS FOR CALLING VIRTT LATER
      CALL CODE
      WRITE(NFMG,2999)LUREM
2999  FORMAT("FMG"I2,5X)
      DO10 I=1,3
      WRITE(LU,3000)I
3000  FORMAT("LOCAL LU FOR SUBCHANNEL"I3"?_")
      LU2=0
      READ(LU,*)LU2
      LU1(I)=-LU2
10    IF(LU2.EQ.0)LU1(I)=-LU
      CALL GNODE(LNODE)
      LNODE=-LNODE
      LU01=-LU
185   CALL DEXEC(NODE,24,6HVIRTT ,LNODE,LU01,LU1(1),LU1(2),LU1(3))
C SCHEDULE VIRTT NO WAIT AND SUSPEND (COME BACK REGULARLY TO SEE
C IF EVERYTHING IS OK)
      IT11=-IT
200   CALL EXEC(12,0,1,0,IT11)
      IF(IFBRK(I))1500,201,201
C  CHECK TO SEE IF REMOTE FMGR IS STILL RUNNING
201   CALL DEXEC(NODE,99,NFMG,IST)
      IF(IST)202,202,3333
202   CALL REIO(2,LU,24H DO YOU WANT TO STOP ? _,-24)
      IREP=2HYE
      CALL REIO(1,LU+400B,IREP,-2)
C CLEAR CALL
1500  CALL DEXEC(NODE,23,6HVIRTT ,-1,-1)
C IF YOU ANSWER YES RESTART ALL THE STUFF
C BUT IF REM. FMGR IS DEAD YOU WILL HAVE THE CHANCE
C TO EXECUTE ONE REMOTE SYSTEM COMMAND (I.E. RU,FMGR)
      IF(IREP.EQ.2HYE)GOTO2000
      IT=600
      GOTO185
3333  IT=100
C REMOTE FMGR IS O.K SO SHORT SUSPEND TIME
      GOTO200
```

```
1000  WRITE(LU,3002)
3002  FORMAT("VIRTT NOT AVAILABLE")
      CALL EXEC(6,0)
2000  WRITE(LU,3003)
      CALL EXEC(3,LU+2000B)
C SET TIME-OUT TO ZERO (THIS SHOULD BE MODIFIED IF THE
C TERMINAL PREVIOUSLY HAD A TIME-OUT SET
      CALL EXEC(3,LU+2200B,0)
3003  FORMAT("COMMUNICATION TERMINATED")
      END
      END$
```

```
                    ASMB,L
**  NO ERRORS PASS#1 **RTE ASMB 92067-16011**
```

```
                    ASMB,L
 00000                    NAM REMEX,7    DEXEC PROCESSOR   AUG 1979  REV.A
* THIS PROGRAM WAS WRITTEN ONLY BECAUSE THE FORMAT OF THE
* PARAMETERS WAS NOT DIRECTLY COMPATIBLE WITH A SIMPLE FORTRAN CALL.
                          ENT REMEX
                          EXT DEXEC,.ENTR
 00000 000000  NODE  NOP  DESTINATION  NODE
 00001 000000  CONTW NOP  ORIGINAL CONTROL WORD (EQT6)
 00002 000000  BUFAD NOP  BUFFER ADDRESSE
 00003 000000  LEN   NOP  BUFFER LENGHT
 00004 000000  ACTLU NOP  DESTINATION LU
 00005 000000  ERR   NOP  ERROR FLAG
 00006 000000  REMEX NOP
 00007 016002X       JSB .ENTR
 00010 000000R       DEF NODE
 00011 002400        CLA
 00012 172005R       STA ERR,I
 00013 162001R       LDA CONTW,I     RE-CONSTRUCT
 00014 012040R       AND =B177700    CONTROL WORD WITH'
 00015 132004R       IOR ACTLU,I     ACTUAL LU
 00016 072036R       STA CNW
 00017 162001R       LDA CONTW,I     ISOLATE REQUEST CODE
 00020 012041R       AND =B17
 00021 032042R       IOR =B100000    SET NO ABORT BIT
 00022 072037R       STA REQ
 00023 162002R       LDA BUFAD,I
 00024 072032R       STA ADD
 00025 016001X       JSB DEXEC       READY FOR DEXEC
 00026 000034R       DEF *+6
 00027 100000R       DEF NODE,I
 00030 000037R       DEF REQ
 00031 000036R       DEF CNW
 00032 000000  ADD   NOP
 00033 100003R       DEF LEN,I
 00034 136005R       ISZ ERR,I     ERROR RETURN SET FLAG
 00035 126006R       JMP REMEX,I
 00036 000000  CNW   NOP
 00037 000000  REQ   NOP
 00040 177700
 00041 000017
 00042 100000
                     END
**  NO ERRORS *TOTAL **RTE ASMB 92067-16011**
```

```
0001                    ASMB,L
**  NO ERRORS PASS#1  **RTE ASMB 92067-16011**


0001                    ASMB,L
0002  00000                    NAM MWF,7
0003                           ENT MWF
0004*  MOVES WORDS FROM SYSTEM MAP (FORTRAN CALLABLE)
0005                           EXT .ENTR
0006  00000 000000  I      NOP
0007  00001 000000  J      NOP
0008  00002 000000  L      NOP
0009  00003 000000  MWF    NOP
0010  00004 016001X        JSB .ENTR
0011  00005 000000R        DEF I
0012  00006 105745         LDX L,I
      00007 100002R
0013  00010 162000R        LDA I,I
0014  00011 066001R        LDB J
0015  00012 105706         MWF
0016  00013 126003R        JMP MWF,I
0017                       END
**  NO ERRORS *TOTAL **RTE ASMB 92067-16011**
```

```
                     ASMB,L
**  NO ERRORS PASS#1 **RTE ASMB 92067-16011**

                     ASMB,L
      00000                NAM BMAP,7      MAP ANY PAGE  JLD AUG 79 REV A
                           ENT BMAP
                           EXT .ENTR,EXEC,$LIBR,$LIBX
*********************************************
*   CALL BMAP (PAGE,LEN,ADD,RADD,ERROR)
*********************************************
* THIS PROGRAM MOVES ANY PHYSICAL PAGE IN
* PROGRAM AREA
      00000 000000  PAGE   NOP             PHYSICAL PAGE
      00001 000000  LEN    NOP             BUF LENGTH (MAX 512 WORDS)
      00002 000000  ADD    NOP             ORIGINAL BUFFER ADDRESS
      00003 000000  RADD   NOP             NEW ADDRESS
      00004 000000  ERROR  NOP             ERROR FLAG
      00005 000000  BMAP   NOP
      00006 016001X        JSB .ENTR
      00007 000000R        DEF PAGE
      00010 002400         CLA
      00011 172004R        STA ERROR,I
      00012 162001R        LDA LEN,I
      00013 002021         SSA,RSS          POSITIVE = WORD
      00014 026020R        JMP A1           YES
      00015 003004         CMA,INA          NO
      00016 002004         INA              MAKE POSITIVE
      00017 001100         ARS
      00020 072137R A1     STA LENW
      00021 042211R        ADA =D-513
      00022 002020         SSA              POSITIVE
      00023 026026R        JMP A2           NO
      00024 036004R        ISZ ERROR        YES BUFFER TOO LONG
      00025 126005R        JMP BMAP,I       PROCESSING REJECTED
      00026 016002X A2     JSB EXEC
      00027 000035R        DEF *+6
      00030 000140R        DEF D26          GET PARTITION INFO
      00031 000141R        DEF IFPG         AND MAPS
      00032 000142R        DEF ILMEM
      00033 000143R        DEF NPGS
      00034 000144R        DEF MAP
      00035 062142R        LDA ILMEM
      00036 012212R        AND =B76000
      00037 101052         LSR 10
      00040 042143R        ADA NPGS
      00041 072143R        STA NPGS        CAL TOTAL # OF PAGES
      00042 062141R        LDA IFPG
      00043 012212R        AND =B76000
      00044 052141R        CPA IFPG         IS IT A PAGE BOUNDARY
      00045 002001         RSS
      00046 042213R        ADA =B2000       NO
      00047 072204R        STA FREE        FIRST FREE PAGE
      00050 101052         LSR 10
      00051 012214R        AND =B37
```

```
0052*      ADA =D1
0053  00052 072206R       STA PAGNO     GET PAGE #
0054  00053 162002R       LDA ADD,I
0055  00054 012215R       AND =B1777
0056  00055 072205R       STA OFSET
0057  00056 070001        STA 1
0058  00057 042204R       ADA FREE
0059  00060 172003R       STA RADD,I
0060  00061 002404        CLA,INA        NPAGE =1
0061  00062 046137R       ADB LENW       OFSET+LENW
0062  00063 101100        SWP            GOES
0063  00064 012216R       AND =B6000     OVER ONE PAGE ?
0064  00065 002002        SZA
0065  00066 006004        INB            YES:NPAGE =2
0066  00067 007004        CMB,INB        MAKE NEGATIVE
0067  00070 076207R       STB NPAGE      AND SAVE
0068* MIN TWO FULL PAGES MUST BE FREE BEHIND THE PROGRAM
0069* IN ORDER TO HANDLE 512 W REQUESTS
0070* IFPG+ILMEM-2K>=FREE
0071* -(FREE+2K)+IFPG+ILMEM>=0
0072  00071 062204R       LDA FREE
0073  00072 042217R       ADA =B4000
0074  00073 003004        CMA,INA
0075  00074 042141R       ADA IFPG
0076  00075 042142R       ADA ILMEM
0077  00076 002020        SSA
0078  00077 136004R       ISZ ERROR,I
0079*   CALCULATE NEW MAP
0080  00100 066210R       LDB LMAP
0081  00101 046206R       ADB PAGNO
0082  00102 162000R       LDA PAGE,I
0083  00103 012220R       AND =B37777   SUPP PROTECT BITS
0084  00104 170001        STA 1,I
0085  00105 002004        INA
0086  00106 006004        INB
0087  00107 036207R       ISZ NPAGE
0088  00110 170001        STA 1,I
0089* GO PRIVILEGED
0090  00111 016003X       JSB $LIBR
0091  00112 000000        NOP
0092* GET HIDDEN BASE PAGE
0093  00113 062221R       LDA =D33
0094  00114 166210R       LDB LMAP,I
0095  00115 105745        LDX =B1
      00116 000222R
0096  00117 105721        XMS
0097  00120 062210R       LDA LMAP       MOVE NEW MAP REG
0098  00121 066223R       LDB =B3740      IN HIDDEN
0099  00122 105777        MVW =D32       B.PAGE
      00123 000224R
      00124 000000
0100  00125 062210R       LDA LMAP       AND IN
0101  00126 101711        USA            USER MAP
```

```
0102* RETURN NORMAL MODE
0103  00127 103105        CLF 5B
0104  00130 061770        LDA 1770B
0105  00131 002003        SZA,RSS
0106  00132 102705        STC 5B
0107  00133 016004X       JSB $LIBX
0108  00134 000135R       DEF *+1
0109  00135 000136R       DEF *+1
0110  00136 126005R       JMP BMAP,I
0111  00137 000000 LENW   NOP
0112  00140 000032 D26    DEC 26
0113  00141 000000 IFPG   NOP
0114  00142 000000 ILMEM  NOP
0115  00143 000000 NPGS   NOP
0116  00144 000000 MAP    BSS 32
0117  00204 000000 FREE   NOP
0118  00205 000000 OFSET  NOP
0119  00206 000000 PAGNO  NOP
0120  00207 000000 NPAGE  NOP
0121  00210 000144R LMAP  DEF MAP
      00211 176777
      00212 076000
      00213 002000
      00214 000037
      00215 001777
      00216 006000
      00217 004000
      00220 037777
      00221 000041
      00222 000001
      00223 003740
      00224 000040
0122                      END
**  NO ERRORS *TOTAL **RTE ASMB 92067-16011**
```

```
0001                ASMB,L
**  NO ERRORS PASS#1 **RTE ASMB 92067-16011**

0001                ASMB,L
0002  00000               NAM DVV05,0        VIRTUAL TERMINAL DRIVER JLD
0003                      ENT IV05,CV05
0004                      EXT $LIST
0005***********************************************************
0006*   DUMMY DRIVER TO INTERCEPT I/O CALLS
0007*   AND PASS THEM TO A SPECIAL TYPE 3
0008*   I/O ON ANOTHER LU (REMOTE LU)
0009*
0010*   SPECIAL CONTROLS:
0011*          37B INITIALIZE, I.E. PASS ID ADDRESS OF PROGRAM
0012*                               TO BE SCHEDULED.
0013*          36B SIMULATE UNEXPECTED INTERRUPT, I.E. PRMPT SCHEDULE
0014*   THE DRIVER MUST BE GENERATED WITH PRMPT (OR EQUIVALENT)
0015*   IT USES A DUMMY SELECT CODE
0016*
```

```
*************************************************************
*
*   THE SCHEDULED PROGRAM GETS 5 PARAMETERS:
*                       EQT1
*                       CNTRL WORD  (EQT 6)
*                       BUF ADD     (EQT 7)
*                       BUF LEN     (EQT 8)
*                       PHYSICAL PAGE OF BUF OR IPRAM (CALL EXEC 3)
*   EQT 11 CONTAINS ID OF PROGRAM TO BE SCHEDULED
*   EQT 12 ACTUAL TRANSMISSION LOG
*   EQT 13 = -1 >> FORCE I/O REJECT (REMOTE LU IN ERROR)
*
00000 000000  IV05  NOP
00001 062211R       LDA FLAG
00002 002002        SZA
00003 026023R       JMP NORM
* GET ID ADDRESS OF PRMPT
00004 161663        LDA EQT4,I
00005 012216R       AND =B77
00006 042217R       ADA =D-6
00007 041654        ADA INTBA
00010 164000        LDB A,I
00011 007004        CMB,INB
00012 006020        SSB
00013 026062R       JMP REJCT       SHOULD BE POSITIVE
00014 076212R       STB PRMPT       SAVE ID ADDRESS
00015 065660        LDB EQT1
00016 174000        STB A,I
00017 161663        LDA EQT4,I      WE HANDLE TIME-OUT
00020 032220R       IOR =B10000
00021 171663        STA EQT4,I
00022 036211R       ISZ FLAG
00023 161665  NORM  LDA EQT6,I
00024 012221R       AND =B3703       TEST FOR INITIALIZE
00025 052221R       CPA =B3703
00026 002001        RSS
00027 026036R       JMP I1         NO, CONTINUE
00030 161666        LDA EQT7,I      SAVE ID ADDRESS
00031 002020        SSA            POSITIVE?
00032 002400        CLA             NO, RESET TO 0
00033 171672        STA EQT11,I          AND SAVE
00034 065660        LDB EQT1
00035 026060R       JMP IMCOM       GO TO IMMEDIATE COMPLETION
00036 161672  I1    LDA EQT11,I  IS THE DRIVER INITIALIZED?
00037 002003        SZA,RSS
00040 026062R       JMP REJCT
00041 161665        LDA EQT6,I      YES, CONTINUE
00042 012222R       AND =B3603
00043 052222R       CPA =B3603      TEST FOR PRMPT SCHEDULE
00044 002001        RSS
00045 026064R       JMP I2          NO PRMPT SCHEDULE
00046 062212R       LDA PRMPT       YES, SCHEDULE PRMPT
00047 072054R       STA SCHAD
00050 065663        LDB EQT4
00051 076055R       STB SCHAD+1
00052 016001X       JSB $LIST
00053 000601        OCT 601
00054 000000  SCHAD NOP
00055 000000        NOP
```

```
*  TEST FOR SUCCESSFUL SCHEDULE
   00056 002002          SZA
   00057 026062R         JMP REJCT
   00060 062223R IMCOM   LDA =D4
   00061 126000R         JMP IV05,I
   00062 062224R REJCT   LDA =D3
   00063 126000R         JMP IV05,I
*
*  NORMAL CALL
*
   00064 161665  I2      LDA EQT6,I
   00065 012224R         AND =B3
   00066 052224R         CPA =B3        CONTROLL ACLL?
   00067 002001          RSS
   00070 026074R         JMP I3         NO
   00071 161666          LDA EQT7,I     YES
   00072 072214R         STA PAGE
   00073 026112R         JMP I6
*
*  NORMAL CALL (I.E. 1 OR 2) SO CALC PHYS. PAGE OF I/O BUFFER
   00074 161666  I3      LDA EQT7,I
   00075 101052          LSR 10
   00076 012225R         AND =B37      ISOLATE PAGE
   00077 101100          SWP
   00100 101730          RSA            READ DMS
   00101 012226R         AND =B30000     ARE SYSTEM MAP & MEMORY ON?
   00102 052227R         CPA =B20000
   00103 026105R         JMP I5           YES
   00104 042230R         ADA =D32         OFFSET TO USER MAP
   00105 101100  I5      SWP
   00106 066177R         LDB DPAGE        SAVE
   00107 105745          LDX =D-1         PHYSICAL STARTING PAGE
   00110 000231R
   00111 105720          XMM              OF I/O BUFFER IN 5TH SCHEDULE PARAM.
   00112 161664  I6      LDA EQT5,I
   00113 012232R         AND =B177400
   00114 171664          STA EQT5,I
   00115 062233R         LDA =D-2
   00116 171773          STA EQT14,I
   00117 171774          STA EQT15,I
   00120 062234R         LDA =D-100
   00121 072213R         STA COUNT
   00122 072215R         STA SCHED
   00123 002400          CLA
   00124 126000R         JMP IV05,I
```

```
*
*   CONTINUATION
*
00125 000000  CV05  NOP
00126 161660        LDA EQT1,I       TEST IF DRIVER BUSY
00127 002002        SZA
00130 026133R       JMP SUIT
00131 036125R       ISZ CV05
00132 126125R       JMP CV05,I        NO, IGNORE
00133 161663  SUIT  LDA EQT4,I   YES, CLEAR TIME-OUT
00134 012235R       AND =B173777
00135 171663        STA EQT4,I
00136 062215R       LDA SCHED
00137 002002        SZA               PROGRAM TO BE SCHEDULED?
00140 026153R       JMP SCH           YES
00141 165771        LDB EQT12,I
00142 161772        LDA EQT13,I       CHECK
00143 002021        SSA,RSS           IF FORCE ERROR REQUIRED?
00144 026151R       JMP END            NO
00145 002400        CLA
00146 171772        STA EQT13,I        CLEAR ERROR FLAG
00147 062236R ERROR LDA =D1      SET ERROR
00150 126125R       JMP CV05,I
00151 002400  END   CLA
00152 126125R       JMP CV05,I
00153 161672  SCH   LDA EQT11,I
00154 072172R       STA SCH1
00155 042237R       ADA =D15
00156 160000        LDA A,I
00157 012237R       AND =B17           DORMANT?
00160 002002        SZA
00161 026201R       JMP NOT
00162 002400        CLA
00163 072215R       STA SCHED
00164 072213R       STA COUNT
00165 171773        STA EQT14,I
00166 171774        STA EQT15,I
00167 016001X       JSB $LIST
00170 000001        OCT 1
00171 000200R       DEF RET
00172 000000  SCH1  NOP
00173 001660        ABS EQT1
00174 001665        ABS EQT6
00175 001666        ABS EQT7
00176 001667        ABS EQT8
00177 000214R DPAGE DEF PAGE
00200 026213R RET   JMP COUNT
00201 062240R NOT   LDA =D-10
00202 171773        STA EQT14,I
```

```
0169  00203 171774         STA EQT15,I
0170  00204 036213R        ISZ COUNT
0171  00205 002001         RSS
0172  00206 026147R        JMP ERROR
0173  00207 036125R CONT   ISZ CV05
0174  00210 126125R        JMP CV05,I
0175  00000         A      EQU 0
0176  00001         B      EQU 1
0177  01660         EQT1   EQU 1660B
0178  01663         EQT4   EQU 1663B
0179  01664         EQT5   EQU 1664B
0180  01665         EQT6   EQU 1665B
0181  01666         EQT7   EQU 1666B
0182  01667         EQT8   EQU 1667B
0183  01672         EQT11  EQU 1672B
0184  01771         EQT12  EQU 1771B
0185  01772         EQT13  EQU 1772B
0186  01773         EQT14  EQU 1773B
0187  01774         EQT15  EQU 1774B
0188  00211 000000  FLAG   NOP
0189  01654         INTBA  EQU 1654B
0190  00212 000000  PRMPT  NOP
0191  00213 000000  COUNT  NOP
0192  00214 000000  PAGE   NOP
0193  00215 000000  SCHED  NOP
      00216 000077
      00217 177772
      00220 010000
      00221 003703
      00222 003603
      00223 000004
      00224 000003
      00225 000037
      00226 030000
      00227 020000
      00230 000040
      00231 177777
      00232 177400
      00233 177776
      00234 177634
      00235 173777
      00236 000001
      00237 000017
      00240 177766
0194                       END
**   NO ERRORS *TOTAL **RTE ASMB 92067-16011**
```

RTE-IVB QUICK REFERENCE GUIDE

Helen Fuller/HP Data Systems Division

The RTE-IVB Quick Reference Guide is available and it has a new style — a small flexible looseleaf binder. The new RTE-IVB Guide contains succinct explanations of terminal commands, programmatic calls, error codes, tables, and more! It is easily handled, transported, and is indexed with tabs for super quick reference. Make this an important addition to your RTE-IVB manual set by ordering both the RTE-IVB text insert and the Quick Reference Guide and the Quick Reference Guide Binder (part number, 02177-90007) Binder today. Both items will be distributed to customers who have SSS or CSS as part of the RTE-IVB product during the 2026 update cycle.

Ordering Information

ITEM	PRICE
92068-90003 text insert	$ 9.50
02177-90007 binder	$ 5.00
TOTAL	$14.50

RTE-IVB ON-LINE DIAGNOSTICS AND VERIFICATION PACKAGE

John Koskinen/HP Data Systems Division

All those good diagnostics and verification routines placed on the RTE-4B primary systems are now available as a product — 91711A.

The on-line diagnostics and verification routines are now supplied as a relocatable product. The routines can be used on a customer's generated system simply by using the LOADR. No special off-line program loading is required. This product is not the same as the 24396 series. The 24396 series diagnostics are off-line - they need to be loaded into the system and are run completely standalone.

The new on-line product begins the way all diagnostics and verification routines will be done in the future. If a new diagnostic must be run standalone, it will be loadable into a standard RTE-4B system. The 24396A-F product will now be placed in the mature software category.

The diagnostic package handles:

Processor, Memory, and Firmware.
7900/06/20/25 MAC/ICD Drives.
7970 Mag Tape.
Line Printer.
2645/48 Point-to-Point/Multi-point.
3070/75/77 Terminals.
RS-232 Terminals.

The product is classified as Active Type II software, which means free right-to-copy.

Ordering Information

ITEM	PRICE
91711A	$500
-001 Cassette	30
-051 800 BPI MT	0
-052 1600 BPI MT0	
91711Q Manual Update Service	2/mo.
91711S Software Updates	20/mo.
-020 Cassette	10/mo.
-051 800 BPI MT	0
-052 1600 BPI MT	0

A NEW INDEPENDENT STUDY COURSE IN RTE FORTRAN IV

Jim Williams/HP Data Systems Division

A self-paced course in RTE FORTRAN IV is now available for order. This independent study course consists of six color videocassettes presenting the fundamentals of FORTRAN programming on an HP1000 computer under the RTE operating system. The course is fully modular in design, segmented by subject area into ten modules for ease of use by first-time FORTRAN programmers, and for selective review by previously experienced programmers.

A student workbook is required by each student to lead him or her through each module. Self-evaluation questions at the beginning of each module allow the student to evaluate the module for content prior to viewing the tape, and skip to the next session if he or she desires. Supplemental problems and lab exercises (as well as possible solutions) are provided for hands-on practice.

A person knowledgeable in RTE FORTRAN should be available to assist the inexperienced student when required. An instructor's guide is supplied to facilitate this "advisor" in providing aid to the student. The instructor's guide also provides a recommended "fast" classroom course for group on-site training by the user.

A brief description of the subject areas covered by each module follows. Note that this course is a pre-requisite to attendance of the RTE-IVB/Session Monitor User's Course for students without prior FORTRAN experience.

Session	Topic
1	Introduction, Course Organization History of Programming Languages
2	RTE-IVB Procedures
3	FORTRAN Character Set, Operators, Expressions
4	Input/Output Procedures
5	Branching, Testing and Looping
6	DO Loops
7	Array Processing
8	Functions and Subprograms
9	Debugging a FORTRAN Program
10	Additional FORTRAN Statements, FORTRAN Surprises VIS/EMA Features

Ordering information is as follows:

Place a Heart order to Data Systems Division 2200 for: (Orders placed prior to JUNE 1 must override the Heart system to order this course.)

PRODUCT NUMBER 22958B .... LIST PRICE $1000.00

OPTION 001 U-Matic 3/4 inch color videocassettes

OPTION 002 VHS 1/2 inch color videocassettes

OPTION 003 one student workbook .... LIST PRICE $50.00 each

OPTION 004 Betamax 1/2 inch color videocassettes

This course must be ordered with either option 001, 002, or 004 and includes the six videocassettes and one instructor's guide. As many student workbooks as required may be ordered as option 003. A set of overhead slides are separately orderable as part number 22999-90240 (LIST PRICE $190.00).

Note that for customers who desire off-site training in FORTRAN, this course is being offerred at HP Regional Training Centers as required, with systems and experienced instructors available to the student through the intensive three-day schedule.

JOIN AN HP 1000 USER GROUP!

Here are the groups that we know of as of April 1980. (If your group is missing, send the Communicator/1000 editor all of the appropriate information, and we'll update our list.) We apologize for the incorrect spelling of some names in the past. They have been corrected in this issue.

NORTH AMERICAN HP 1000 USER GROUPS

Area	User Group Contact
Boston	LEXUS P.O. Box 1000 Norwood, Mass. 02062
Chicago	Jim McCarthy Travenol Labs 1 Baxter Parkway Mailstop 1S-NK-A Deerfield, Illinois 60015
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
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

NORTH AMERICAN HP 1000 USER GROUPS (CONTINUED)

Area	User Group Contact
Toronto	Nancy Swartz Grant Hallman Associates 43 Eglinton Av. East Suite 902 Toronto M4P1A2
Washington/Baltimore	Paul Tatavull 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

London	Rob Porter Hewlett-Packard Ltd. King Street Lane Winnersh, Workingham Berkshire, RG11 5AR England (734) 784 774
Amsterdam	Mr. Van Putten Institute of Public Health Anthony Van Leeuwenhoeklaan 9 Postbus 1 3720 BA Bilthoven The Netherlands
South Africa	Andrew Penny Hewlett-Packard South Africa Pty. private bag Wendywood Sandton, 2144 South Africa
Belgium	Mr. DeFraine K.U.L. Celestijneulann, 300C B-3030 Heverlee Belgium

Printed in U.S.A. 5/80

Part No. 5951-6111