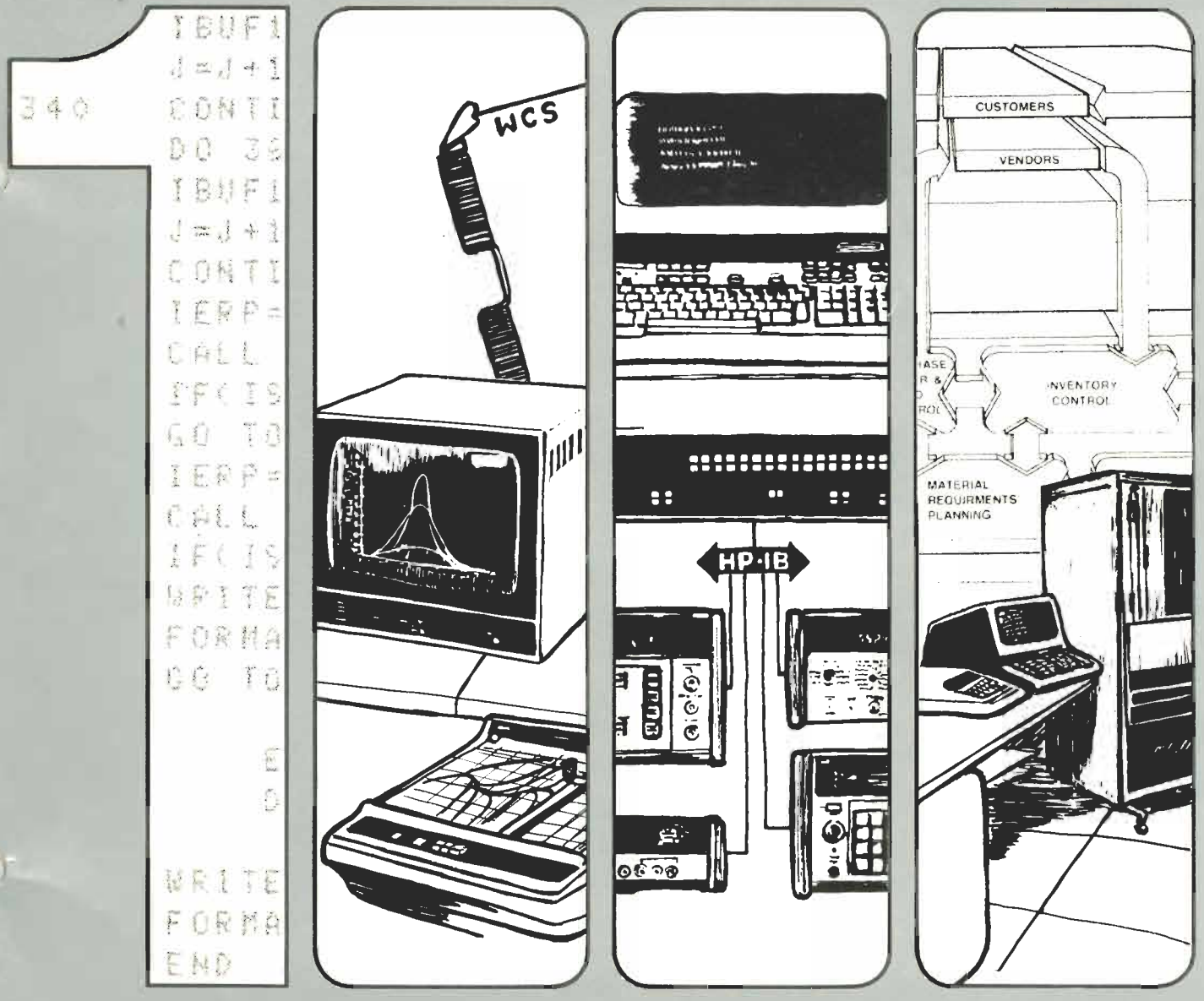
HEWLETT PACKARD

DJG
BAL
FILE

Hewlett-Packard Computer Systems

COMMUNICATOR



1979

Volume III Issue 5

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

IN THE NEXT ISSUE

Articles for the next issue, (and issues after that one) have not been selected yet. In fact, few of the articles that will be published in coming issues have even been written yet! The list of possible topics for these articles is endless, as was proven during a recent brainstorming session here at Data Systems Division. An extensive list of exciting and informative topics came out of that session, some of which I have listed below:

PROGRAMMING DATACAP/1000 FOR PEAK USAGE PERIODS
PROGRAM MODIFICATION TO/FROM NON-RTE-IV OPERATING SYSTEMS
USING GRAPHICS/1000 AND IMAGE/1000 TO RECORD AND DISPLAY DATA
OPTIMUM CONFIGURATIONS FOR DS/1000 NETWORKS
THE FEATURES AND BENEFITS OF ACCOUNTS IN RTE-IVB
HOW TO ENHANCE THE SECURITY OF A SYSTEM
TAKING ADVANTAGE OF PROGRAM PRIORITIES AND TIME-SLICING
TEXT PROCESSING ON AN HP 1000
THE MANY USES OF GLOBAL PARAMETERS

The Communicator is intended to provide a place to publish that unique super-efficient, or tricky way that you have found to use an HP 1000 product. As Editor of the Communicator, I am counting on YOU, one of our 2600 readers to take the initiative, and then the time, to grab one of the topics above, or one of your own, and put together an article. (Don't forget that for each issue, three HP 32E calculators are awarded to contributors, one each in the customer, HP field personnel, and HP division personnel categories.)

If English is a second language for you, I would welcome the opportunity to help you in any way I can with your English text. Let's not let language be a barrier.

For more information about submitting an article to the Communicator, see the Editor's Desk pages of this issue. If you need any further information, I invite you to contact me at the address below. If you have questions concerning the topic you have selected, or have a suggestion for a topic that is not shown in this list, please contact me.

I hope to be hearing from you in the near future. Thanks in advance for your continued support and participation in the Communicator 1000.

Best Regards,

Editor/Communicator 1000
HP Data Systems Division
11000 Wolfe Rd.
Cupertino, Ca. 95014

HEWLETT-PACKARD
COMPUTER SYSTEMS

Volume III
Issue 5

COMMUNICATOR/1000

Feature Articles

Departments

ABOUT THIS ISSUE

This fifth Communicator issue for 1979 includes articles in a variety of topical categories. The reader will also notice that articles in this issue present information from several different perspectives.

In the DATA COMMUNICATIONS category, two articles have been contributed which describe specific applications of DS/1000. Phil Shepard of HP's Data Systems Division describes an application of Program to Program communications to dump files from private cartridges on one system, to devices on another system. Phil's programs have saved undeterminate amounts of time here at Data Systems, by moving files that must be printed on our upper/lower case printer, which is located on a remote system. Every article in this issue of the Communicator took advantage of Phil's program. The second article in the DATA COMMUNICATIONS category is contributed by HP customers, John Pezanno, Bill Reynolds, and Howard Beyer of Holloman Air Force Base, in New Mexico. Their article describes how DS/1000 is used to transmit data to a remote computer via a modem. In itself, this is not a difficult task. However the unique capability of the system in use at Holloman is that it can completely control the computer at the remote site. The only operation that cannot be carried out over the modem line is mounting of the tapes! Thanks to all four of these authors for their fine contributions.

In the OPERATING SYSTEMS category, Gary McCarney, an HP Systems Engineering Manager working in the Rockville, Maryland office has contributed an article entitled, "An Introduction to Operating Systems Fundamentals". This article originated as a "warm up" speech to students about to take an RTE operating systems course. Gary put a great deal of effort into translating his speech to a written format, in hopes that it will help others gain an understanding the fundamentals of operating systems. It appears that his article will become another Communicator/1000 classic, (like the "Know Your RTE" series), a useful reference for years to come. Thanks to Gary for his efforts.

In the OPERATIONS MANAGEMENT category, two Data Systems lab engineers have put together an article entitled, "Performance Specifications for Datacap/1000". Ben Heilbronn and Steve Richard have spent several weeks studying how Datacap/1000 can be expected to perform on an HP 1000. It is Ben's and Steve's expectation that people with Datacap installations will be able to find the optimum way their system can be structured, and people contemplating installation of a Datacap system will have a realistic idea of what to expect from it. Thanks are in order for all Steve's and Ben's efforts.

The final feature article, also in the OPERATIONS MANAGEMENT category is by Todd Field, an HP Systems Engineer in the Woodbury, New York office. Todd's article "Easy Forms For The 2645A" follows up on two previous Communicator articles that dealt with capabilities of the 2645 terminal. Todd's article describes a simple and straightforward method of creating a form on the 2645, storing that form in a disc file, and then programmatically retrieving this form and dumping it to the screen. Anyone using the 2645 for the type of application Todd describes, will surely find his methods to be time savers.

The objective of the Communicator/1000 is to provide reading material for persons with various levels of technical expertise, and varied interests. Hopefully, the articles described above will accomplish this objective to your satisfaction.

EDITOR'S DESK

The following articles/authors have been awarded HP-32E calculators, both as a token of appreciation to the author, and as a salute to the article's overall quality.

Best Feature Article By A Customer	**Remote System Control Via DS/1000** John Pezanno Bill Reynolds Howard Beyer /Holloman AFB
Best Feature Article From an HP Division Employee	**Performance Specification for DS/1000** Ben Heilbronn Steve Richard /HP Data Systems Division
Best Feature Article From an HP Field Employee	**Easy Forms for the 2645A** Todd Field /HP Eastern Sales Region

Thanks to everyone whose effort helped to make this Communicator possible. It is hoped that this issue of the Communicator will be interesting and useful to our readers.

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

SOME NOTES FROM PREVIOUS ISSUES

In Volume III Issue 3, an article appeared by Ed DeMers entitled, "User Codes As An Aid To Disc Housekeeping". This article described the programs Mr. DeMers submitted to the Contributed Library. These programs are entitled the UCU Group, and issue 3 incorrectly reported the catalog number as 22683-13381. The correct catalog number is 22683-13341. The Editor of the Communicator apologizes for any inconvenience this may have caused our readers.

Also appearing in Volume III Issue 3, was a list of entry points for convenience in generating systems. Some questions have been raised regarding these RP's. The VIS entry points were included for completeness and should not be included in a generation unless VIS firmware has been installed. Likewise, the double integer entry points should be used only on F-Series computers with a date code of 1920 or greater, or when the appriopriate firmware update has been installed. Hopefully this will clear up some of the confusion surrounding these RP's.

A third error found in Volume III Issue 3 occured in Frank Fulton's article,"Scheduling Data Acquisition Programs Without Providing General System Access To Terminal Users". Mr. Fulton's article references an article by Gary McCarney about an MTM-like terminal handler. It was incorrectly stated that Mr. McCarney's article appeared in Volume III Issue 2 of the Communicator. Actually that article appeared one year earlier in Volume II Issue 2.

The following article should have been referenced throughout Larry W. Smith's article in Volume III Issue 2, titled, "A Method for Smooth Curve Fitting":

- Akima,H. (1970), "A New Method of Interpolation and Smooth Curve Fitting Based on Local Procedures", Journal of the ACM, Volume 17, Number 4, October, pp. 589-602.

LETTER TO THE EDITOR

Dear Editor,

We have noted with great interest your description of THE OEM CORNER in the past few issues of the Communicator/1000. I write you now in hopes that you might answer a few of our questions.

The MAGUS System Group has been working with RTE systems for the past five years. With the introduction of the new RTE-IVB Session Monitor system we have seen RTE at last come close to its full potential. We have a number of software add-ons for the RTE-IVB system which we believe both compliment and add additional power to the constructs introduced in RTE-IVB.

The MAGNUS System Group perceives the COMMUNICATOR/1000 to be an excellent vehicle for notifying other HP/1000 owners of our products, and hopefully to promote their sale.

Could you please let us know the restrictions placed on vendors submitting articles to the OEM CORNER? Is it possible to publish purchase prices and a mailing address? Would you forward any enquiries you recieved to us? What are the deadline dates for submission of articles?

Any information you could provide us with would be greatly appreciated.

Thanks Much,

Drew Lanza
Secretary,
MAGNUS System Group

Dear Mr. Lanza,

Thanks for giving me the opportunity to clarify exactly how the OEM CORNER is intended to work.

As far as the restrictions placed on submissions, there are none other than those mentioned in the Editor's Desk. Attached to an article can be up to 150 words of "pure product description", and it does not seem unreasonable to me that part of this description would include the price of the OEM's product, and information on how to obtain the product.

As the Editor of the Communicator, I would prefer not to become a middle-man between someone desiring information about an OEM's product, and that OEM. For this reason, inclusion of a mailing address and/or a phone number in the 150 words of product description would not only be permissible, but also quite appreciated.

In addition, note that a picture or illustration of the OEM's product may accompany the article. We would ask however that the picture or illustration be limited to half of a page or less.

Even though articles submitted to the OEM CORNER are not eligible to win calculators like feature articles, a good OEM CORNER article will follow the same guidelines used to judge the feature articles. Recall that these characteristics are the technical quality, clarity, thoroughness and general interest of the article.

Just as with feature articles, OEM CORNER articles are subject to editorship and minor revisions. If any significant changes seem necessary, I would of course contact you.

In answer to your final question, only one more issue remains to be published in 1979. The deadline date for submissions to that issue will be passed before this issue is distributed. The final issue for 1979 will contain a list of all the deadline dates for the 1980 issues.

I'll look forward to hearing from you, and hopefully also from some other interested OEMs, in the near future. Thanks again for giving me the chance to clarify this aspect of the Communicator/1000.

With best regards,

The Editor

BIT BUCKET

Software Samantha
HP-1000 Communicator
Hewlett-Packard Data Systems Division
11000 Wolfe Road, Cupertino, California 95014

Dear Samantha,

I added HP-IB to my RTE-IV system last week. The system generation had an undefined external named "SRQ.T". The system booted up and ran all night, but when I tried to load a program containing calls to the subroutine SRQ, the loader bombed with the same undefined external. Using the FORCE command I loaded the program and it seems to work.

What is "SRQ.T" and why does "The HP-IB In HP 1000 Computer Systems Users Manual" make no mention of this difficulty?

Sincerely,

Richard B. Gilbert
Gas Dynamics Lab
Princeton University

Dear Mr. Gilbert,

The entry point SRQ.T exists in the Basic memory resident library, %BAMLB. This routine is used for real time scheduling of BASIC subroutines in response to a service request sent over the HP-IB.

Obviously, %BAMLB has not been generated into your system. However this should cause no difficulties (other than the undefined external) in the use of HP-IB from FORTRAN programs, as long as the %HPIB and %MESS libraries are generated into your system.

Unfortunately, the undefined external SRQ.T can be expected anytime the loader relocates the %HPIB library to satisfy entry points in a FORTRAN program which uses HP-IB subroutines. One of the externals in this module (which your program will not need to reference) is SRQ.T.

This problem has been corrected in the latest version of the software. Therefore, one solution to the problem would be to update your system to software revision 1940. (Note that in the 1940 version of the HP-IB software the %MESS and %HPIB libraries are combined into a single library called %IB4A.) You have already found one alternate solution to the problem, and that is to force load all of your HP-IB FORTRAN programs. Another solution would be to include the %BAMLB module in your system generation, or write a small "dummy" routine, with an SRQ.T entry point, and generate it into your system.

The brand new revision of the HP-IB User's Manual, makes mention of this difficulty in Appendix C, which deals with system generation.

I hope this information has been of help to you, and perhaps other HP-IB users. Thanks for your letter.

Sincerely,

Samantha

Dear Samantha,

I believe I share with many others the frustration of having formatter errors unexpectedly routed to LU 6 (typically a line printer).

Routing the formatter errors from assembler programs to a "better" LU is done easily enough by the following coding:

```
EXT   FMT.E
      .
      .
      .
LDA   LU
STA   FMT.E
      .
      .
```

Where LU is the LU to which the errors should be routed, typically the terminal LU. All messages will be ignored by replacing the statement LDA LU in the program above with CLA, and thus routing the messages to the bit bucket. In my FORTRAN programs I have been calling an assembler routine which uses the scheme shown above to route format errors to the LU of my choice. My queston is, why can't I acomplish the same thing by putting the following code in my FORTRAN programs?

```
EXTERNAL FMT.E
         .
         .
         .
FMT.E = LU
```

Sincerely,

John McCabe
HP/Stanford Park Div.

Dear Mr. McCabe,

FORTRAN will not allow a variable to be declared as external to a program. The EXTERNAL statement in FORTRAN can be used only to declare an external subroutine or function name will be used as an argument in a program.

Even if FORTRAN would allow external variables, the name FMT.E is an illegal label, due to the decmal point.

Using a feature of RTE-IVB with session monitor, the SL command will allow the sesion LU 6 (to which the format errors will be routed) to point to any system LU desired. In this way, all line printer output could be directed to a terminal, and all format errors (and everything else your program sends to LU 6) will appear on that terminal.

If this method of re-routing formatter errors is not satisfactory, it apprears that the only other way to be able to select an output LU for formatter errors is to use the scheme that you describe above.

Thanks for your letter.

Sincerely,

Samantha

RESPONSIBILITIES OF THE SYSTEM MANAGER

Alan Housley/HP Data Systems Division

The system manager must perform certain tasks to keep his system running efficiently for himself and the other system users. A list of these tasks has been compiled below.

Above all, the system manager should keep close tabs on the software and hardware requirements of the system users. He should be the focal point for recieving bug reports, hardware malfunction reports, new software requests, new hardware requirements, and be notified of any system "crashes". It is wise to teach others what to do in the event of a system "crash", but since the manager should be maintaining the software and hardware, he should know of all problems so he may take the appriopriate action to find the cause of the "crash" and to make sure that it will not occur again.

Other suggestions for keeping the system operating smoothly are:

- Leave a system log near the system console, so users may mark down any errors that occured which they were unable to resolve themselves.
- Post the boot-up procedure near the virtual control panel.
- Perform new system generations whenever new software arrives, or when the needs of the system are altered.
- Mark cables near the interface cards specifying which peripheral device is connected to each card.
- Mark all peripheral devices with their LU numbers, so users can easily distinguish which device to access.
- Suggest strongly that all users keep a backup of their disc files in the event of a head crash or inadvertent file purge. The manager may also want to perform the disc file backup for all of the users, in order to keep the backups in a common location.
- Incorporate any manual updates or revisions into the manual library.
- Keep a listing of the generation map in a location that system programmers may easily find and use.

RESOURCE SHARING WITH DS/1000 AND SESSION MONITOR, A CASE STUDY

Phil Shepard/HPData Systems Division

Here in the Technical Marketing Department of Data Systems Division, we maintain a network of six HP 1000 computers with an additional link to an HP 3000. One of our functions is the generation and maintenance of manuals describing the software which HP produces. The network exists to serve many groups in the department and there is a reasonably constant flow of file data between the computers in the network.

Before the advent of Session Monitor, it was relatively easy to transfer data from one CPU in the network to another. With the concept of private and group cartridges not yet in existence, one could access data anywhere on any disc in the network.

ENTER SESSION MONITOR

Session Monitor has indeed been a blessing. The protection and isolation it provides users of the system is a great improvement over the past. I no longer delete other user's files with my clumsy fingers, and I now have a good idea where my files are going to be deposited when I allocate file space. I also enjoy the amazing difference in the ease with which I can spool my print files. Many fine enhancements have been made.

Unfortunately along with the goodies come new problems to be solved in accomplishing the old tasks.

Our main use of DS/1000 has been to allow free flow of file data from one system to another in the network. This was accomplished with relative ease with pre-Session Monitor RTE-IV. However, with Session Monitor, and Private, Group and System cartridges, moving data becomes a bit more awkward.

Once REMAT is activated you are no longer in the "Session Environment". This means REMAT, and DS in general, can't access your private cartridges. Only "System" cartridges are available outside the session environment, which means LU's 2 and 3, and any other cartridges the System Manager designates as "system" cartridges. In our case we have two additional cartridges, one for general system files, the other for the Spool System.

To move your file from your private or group cartridge to another node in the network, you must first move the file from your cartridge to a system cartridge. Once this is accomplished the file can be moved with REMAT to a system cartridge on another node.

In our department, we generally move files to get at the upper/lower case printer on the one system in the network where this resource exists. Even after moving the files, we had to go to a terminal on the other system to send the file into the Spool System for printing. This became too much work and was too inefficient, so it was decided to write a program to get around these problems.

ENTER PRT AND PRTSV

When REMAT is asked to move a file with the "ST" command the operation is performed via Remote File Access (RFA) calls and the RFA Monitor (RFAM). This monitor (and all DS monitors) operate outside the Session environment, and don't have access to the Private and Group cartridges, and therefore REMAT does not have access to your Group and Private cartridges as previously stated.

To solve part of the problem, a program could be written to read files with local file calls, and write the files to any other computer in the network with RFA calls. This would allow us to access the Private and Group cartridges locally, and move files to system cartridges of the other computers in the net. This program would not allow us to access the Spool System however, because this can be accomplished only with local program calls.

The decision was made to use the program-to-program communications features of DS/1000. PRT, a DS master program, and PRTSV, a DS slave program were brought into existance.

Using program-to-program communications the master program PRT, in the local system, reads the information from the file in either Private, Group, or System cartridges, and sends the information off to the slave program, PRTSV, in the destination CPU. The slave program can interface directly with the Spool System, receiving records from the master and writing them directly to a spool LU with a minimum of overhead.

Without PRT and PRTSV the following commands would have to be entered:

```
:ST,FILE::PC,FILE::SY      Move file from private to system cartridge
:RU,REMAT
$SW,100,600,DS             Indicate destination is node 600.
#ST,FILE,FILE              Move the file.
#EX
:
    .
    .
    .
```

Go to node 600, log on, and enter the following commands.

```
    .
    .
    .
:SL,11,,,11                Set up LU 11 as a spool LU.
                           (11 is our upper/lower case printer.)
:DU,FILE,11                Ship the file off to the Spool System.
:CS,11                     "Close" the spool LU. (Writes EOF, takes spool file out of a wait state
                           for printing.)
:EX                        Log off.
```

The programs PRT and PRTSV simplify the process so that the following single command is necessary:

```
:RU,PRT,FILE
```

With this command, entered on any system in the network, PRT will open the user's file, either on a private or group cartidge, schedule the slave program PRTSV, read records from the file and send them to the slave until an EOF is encountered.

The slave, when scheduled, acquires a spool LU from the spool system and goes into a "get" state waiting to receive the records. As the records come in, they are written to the spool LU. When the master encounters the EOF, a PCONT call is issued which is interpreted by the slave as time to terminate. The slave closes the spool LU, releasing the associated spool file for printing.

PERFORMANCE CONSIDERATIONS

Initially, sending the data a single record at a time seemed adequate. We were sending small files, and the time PRT was taking to send the files was reasonable. Of course this couldn't last, and along came a user with a 600 block file he was having trouble getting transferred. It turned out this file was overflowing the cartridge used for the spool file. Since the slave was running outside the session environment and only had system cartridges available for storing the spool data, the spool file was being created on LU 2.

Once the program was corrected so the data would go to the spool cartridge "SP", it turned out that it took nine and one half minutes (9.5 MINUTES!) to transfer the 600 block file. This was a little embarassing, and it was concluded that the programs must block and deblock the records rather than send one record at a time.

A 1024 word buffer was chosen based on previous performance studies and the transfer time was reduced to one minute (1 MINUTE!), a notable improvement.

THE PROGRAMS

The program listings are provided for persons interested in how all this magic is accomplished. These listings provide an introduction to those interested in the mechanics of using DS program calls — it's really quite easy.

There is one slight problem. When a non-Session program calls SPOPN to acquire a spool LU, LU's must have been allocated in the generation in the range 0 to 63. Session Monitor expands the range of available LU's to 0 to 255 but the expanded range is not available unless the program is running under the Session environment.

FUTURE ENHANCEMENTS TO DS/1000

The problems solved by this program pair are only temporary as work is in progress to enhance the capabilities of DS1000 to fit more comfortably in the Session environment.

```
FTN4,L
      PROGRAM PRT(3,99), DS/1000 Master  P. Shepard 10 11 79
C*********************************************************************
C     This routine provides spooling to  a printer on a remote cpu.
C     To accomplish this, the user file  is  opened  on  the  local
C     node, records are read  and  blocked in  a BFSZ  word  buffer
C     and  transmitted  to the slave, PRTSV,  via  the  Program-to-
C     Program  Communication features  of  DS/1000.
C
C     PRTSV allocates a Spool LU and writes the records to the Spool
C     System directly.  When  EOF  is  encountered  the  Spool LU is
C     closed and  passed to SMP for processing.
C
C*********************************************************************
C     To run the program:
C
C          RU,PRT,NAMR,NODE,PRINTLU
C*********************************************************************
C
       IMPLICIT INTEGER(A-Z)
       DIMENSION DCB(144),BUFFER(1024),IPBUF(10),NAME(3),PRTSV(3)
       DIMENSION PCB(4),TAG(20)
       DATA BFSZ/1024/
```

```
C
C*********************************************************************
C
C  The following variables are used:
C
C     DCB     - Data Control Block for local file access calls.
C
C     BUFFER  - BFSZ word buffer used to block and pass the records
C               to the slave program.
C
C     IPBUF   - Parameter buffer used by "NAMR" when parsing the run
C               string.
C
C     NAME    - Name array used in the OPENF call to open the input
C               file.
C
C     PRTSV   - Name array used in the POPEN call to schedule PRTSV.
C
C     PCB     - Program-to-Program Communication Control Block used
C               by DS to keep track of transactions going back and
C               forth between master and slave.
C
C     TAG     - Control array used to pass information back and forth
C               between master and slave.
C
C*********************************************************************
C
      DATA PRTSV/2HPR,2HTS,2HV /
C
C  Get the terminal LU, run string, string length, and set the
C  parse pointer (PTR) to 1.
C
      LU=LOGLU(I)
      CALL EXEC(14,1,BUFFER,-80)
      CALL ABREG(I,LEN)
      PTR=1
C
C*********************************************************************
C  Bypass  "RU,PRT"  in the run string and pick up the input file
C  Namr. Namr is described in the RTE Relocatable Library Manual.
C*********************************************************************
C
      CALL NAMR(IPBUF,BUFFER,LEN,PTR)
      CALL NAMR(IPBUF,BUFFER,LEN,PTR)
      I=NAMR(IPBUF,BUFFER,LEN,PTR)
C
C*********************************************************************
C  We're now supposed to have the namr in IPBUF
C*********************************************************************
C
      IF(I.EQ.0) GO TO 10
      WRITE(LU,100)
100   FORMAT(/"  PRT:Failed to parse run string"/)
      STOP
```

```
C
C  Grab the name, security code and crn out of the namr results.
C
10    DO 15 I=1,3
15    NAME(I)=IPBUF(I)
C
      SC=IPBUF(5)
      CR=IPBUF(6)
C
C*****************************************************************
C  Let's cross our fingers and open the file
C*****************************************************************
C
      CALL OPEN(DCB,ERR,NAME,1,SC,CR)
      IF(ERR.GE.0)GO TO 20
      WRITE(LU,110)ERR
110   FORMAT(/"  PRT: Input file open error:  FMGR"I3/)
      STOP
C
C  Get additional parameters. If none, default.  LU 11 = upper/lower
C  case printer, NODE = 600 is the node where we have this printer.
C
20    I=NAMR(IPBUF,BUFFER,LEN,PTR)
      IF(i.EQ.0)GO TO 30
      WRITE(LU,120)
120   FORMAT(/"  /PRT: Using Node=600, LU=11"/)
      NODE  = 600
      PRTLU = 11
      TAG   = PRTLU
      GO TO   55
C
C  Set node parameter and go parse next parameter in string
C
30    NODE=IPBUF
      I=NAMR(IPBUF,BUFFER,LEN,PTR)
      IF(I.EQ.0)GO TO 40
      WRITE(LU,140)
140   FORMAT(/"  PRT: I'm sorry but I need the print LU parameter"/)
      STOP
C
C  Set print LU, and pass it to the slave in TAG(1).
C
40    PRTLU=IPBUF
      TAG=PRTLU
C
C*****************************************************************
C  We've finally got enough goodies, we can now call the slave
C*****************************************************************
C
```

```
55    CALL POPEN(PCB,ERR,PRTSV,NODE,TAG)
      IF(ERR .EQ. 0)GOTO 60
      IF(ERR .EQ. 1)GOTO 57
      WRITE(LU,150)ERR
150   FORMAT(/"PRT: Failed to schedule slave: DS Error = "I3/)
      IF(.NOT.(ERR .EQ. -41))GOTO 56
      WRITE(LU,155)
155   FORMAT(/"  The slave 'PRTSV' is not installed in the destination
     " CPU"/)
      STOP
56    IF(.NOT.(ERR .EQ. -47))STOP
      WRITE(LU,156)
156   FORMAT(/"You need the DS LU's in your Session Switch Table."/
     +"Please enter:"//"      SL,dslu,dslu     and try again. "//
     +"For a permanent solution have your system manager enter the"/
     +"DS LU's in your account."/)
      STOP
C
C   Handle REJECT from slave
C
 57   IF(TAG(2) .EQ. 10)GOTO 58
      WRITE(LU,157)TAG(1),TAG(2)
157   FORMAT("Slave reject: error code = "A2,I3)
      STOP
C
C   Handle slave busy
C
58    WRITE(LU,158)
158   FORMAT("Slave busy, try again later")
      STOP
C
C*********************************************************************
C    Slave is ready and waiting:
C    Buffer the data in a BFSZ word buffer.
C
C        BUFFER(1) = Record count in buffer.
C        BUFFER(2) = Word length of 1st record\\ Repeat for number
C        BUFFER(3) = Start of 1st record      // of rec's in buff.
C*********************************************************************
C
60    CONTINUE
63    IX=3
      BUFFER(1) = 0
C
C  Now loop through reading rec's till EOF encountered or buff full
C
65    CONTINUE
      CALL READF(DCB,ERR,BUFFER(IX),80,LEN)
      IF(ERR .GE. 0)GOTO 70
      WRITE(LU,160)ERR
160   FORMAT(/"  PRT:READF ERR   FMGR"I3)
      STOP
C
C  Increment record count, enter current record length, and advance
C  buffer index to next free buffer space.
C
70    BUFFER=BUFFER+1
      BUFFER(IX-1)=LEN
      IX=IX+LEN+1
```

```
C
C   If we haven't filled the buffer yet or encountered EOF, go read
C   again.  Otherwise, send the current buffer to the slave.
C   (CALL PWRIT)
C
      IF(IX .LT. BFSZ-80 .AND. LEN .NE. -1)GOTO 65
      CALL PWRIT(PCB,ERR,BUFFER,IX,TAG)
      IF(ERR .NE. 0)GOTO 75
      IF(LEN .EQ. -1)GOTO 90
      GOTO 63
C
75    WRITE(LU,170)ERR
170   FORMAT(/"  PRT: DS PWRIT error "I3/)
      IF(.NOT.(ERR .EQ. -58 .OR. ERR .EQ. -47))GOTO 90
      WRITE(LU,175)
175   FORMAT(/"The remote node is busy, there is probably not"/
     +"enough SAM available to receive the incoming message."/)
C
80    CONTINUE
      CALL PCONT(PCB,ERR,TAG)
      CALL CLOSE(DCB)
      STOP
C
C*****************************************************************
C   Indicate to slave we're done
C*****************************************************************
C
90    CONTINUE
      CALL PCONT(PCB,ERR,TAG)
      CALL CLOSE(DCB)
      WRITE(LU,180)
180   FORMAT(/"  PRT: Transfer complete"/)
      END
```

```
FTN4,L
      PROGRAM PRTSV(3,80), Print to spool system. P. Shepard 10 9 79
C*****************************************************************
C   Receives data from master  "PRT", allocates a Spool LU, deblocks
C   the data in the incoming buffers, and writes the data to the
C   Spool LU.
C*****************************************************************
C
      IMPLICIT INTEGER(A-Z)
      DIMENSION BUF(1024),TAG(20),PCB(4),P(5),SMP(3)
      DIMENSION SPLBF(16),DCB(144),NAME(3)
      EQUIVALENCE (NAME(1),SPLBF(3))
      DATA SPLBF/0,00,2HPR,2HTS,2HVA,0,0,12B,402B,99,6*0/
      DATA SMP/2HSM,2HP ,2H  /
C
C  CALL RMPAR TO GET CLASS NUMBER FOR CLASS GET ISSUED BY
C  (CALL GET)
C
      CALL RMPAR(P)
      CLASS = P(1)
```

```
C
C*************************************************
C   NOW SIT HERE AN WAIT FOR SOME ACTION
C      TAG(1)=PRINT LU ON INPUT & ERROR INDICATOR ON
C             RETURN.
C      TAG(2)=ERROR CODE ON RETURN
C*************************************************
C
   5  CALL GET(CLASS,ERR,FCN,TAG,LEN)
      IF(ERR .NE. 0)GOTO 450
C
C*************************************************
C   FCN = 1 POPEN
C         2 PREAD
C         3 PWRIT
C         4 PCONT
C
C*************************************************
C
      GOTO(10,200,300,400)FCN
C
C
10    SPLBF(16)=TAG
C
C*************************************************
C   CREATE A TYPE 4 FILE, LENGTH 24 BLOCKS
C*************************************************
C
12    CALL CREAT(DCB,ERR,NAME,24,4,0,2HSP)
      IF(ERR .GE. 0)GOTO 20
      IF(ERR .NE. -2)GO TO 14
C
C  IF -2 WE CHANGE THE FILE NAME AND TRY AGAIN.  IF FILE ALREADY
C  EXISTS IT MEANS WE HAVE FILES ON DISK THAT HAVEN'T BEEN PRINTED
C  YET.
C
      NAME(3) = NAME(3) + 1
      GO TO 12
C
C  RETURN  FMGR ERROR FOR MASTER TO PRINT AT USER TERMINAL
C
14    TAG=2HFM
      TAG(2)=ERR
15    CALL REJCT(TAG,ERR,BUF)
      GOTO 450
C
C*************************************************************
C   SUCCESSFUL CREATE, CLOSE UP THE FILE AND CALL SPOPN FOR
C   A SPOOL LU.
C*************************************************************
C
20    CALL CLOSE(DCB)
      CALL SPOPN(SPLBF,ISLU)
      IF(ISLU .GT. 0)GOTO 30
C
C  IF WE DIDN'T GET A POSITIVE SPOOL LU, WE HAVE AN ERROR.
C  RETURN SPOOL ERROR TO MASTER TO PRINT
C
      TAG(1)=2HSP
      TAG(2)=ISLU
      GO TO 15
```

```
C
C*************************************************
C   WRITE OUT A FORM FEED
C*************************************************
C
30     ICNWD=IOR(ISLU,1100B)
C
C  WE NEED THE SUBFUNCTION BITS WHEN THEY ARE PASSED.
C  OR INTO THE SPOOL LU AND SAVE AS "LU".  WE CAN'T
C  MESS UP "ISLU" AS IT IS NEEDED IN COMMUNICATION WITH
C  SMP.
C
       LU=IOR(IAND(TAG,177700B),ISLU)
       CALL EXEC(3,ICNWD,-1)
C
C  TAG=0 MEANS NO ERROR TO MASTER, THE ACEPT RETURNS A SUCCESS
C  TO MASTER AND TRANSFERS THE DATA INTO OUR BUFFER
C
       TAG=0
       CALL ACEPT(TAG,ERR,BUF)
C
C*****************************************************
C   FROM NOW ON, WE ACCEPT ONLY PWRIT'S AND PCONT'S.
C   CAN'T ACCEPT ANY MORE POPEN'S OR WE'LL MESS UP.
C*****************************************************
C
 50    CALL GET(CLASS,ERR,FCN,TAG,LEN)
       IF(ERR .NE. 0)GOTO 450
       GOTO(60,200,300,400)FCN
C
C   REJECT POPEN'S WHILE WE'RE BUSY
C
60     TAG(1)=2HSV
       TAG(2)=10
       CALL REJCT(TAG,ERR)
       GOTO 50
C
C*************************************************
C    HANDLE PREAD (WE SIMPLY REJECT THEM)
C*************************************************
C
200    CALL REJCT(TAG,ERR)
       GO TO 5
C
C*************************************************
C    HANDLE PWRIT
C*************************************************
C
300    CALL ACEPT(TAG,ERR,BUF)
C
C   NOW LET'S UNPACK THE BUFFER AND OUTPUT THE RECORDS
C
       IX=3
       DO 320 I=1,BUF
       LEN = BUF(IX-1)
       CALL EXEC(2,LU,BUF(IX),LEN)
320    IX = IX + LEN + 1
C
C   NOW GO BACK AND WAIT FOR NEXT BUFFER OR PCONT.
C
       GOTO 50
```

```
C
C*************************************************
C   HANDLE ERROR EXIT'S
C*************************************************
C
450   CALL FINIS
      STOP 1
C
C*************************************************
C    HANDLE PCONT - CLOSE SPOOL LU AND PASS TO SMP
C*************************************************
C
400   CALL EXEC(3,ICNWD,-1)
      CALL EXEC(23,SMP,4,ISLU)
      CALL ACEPT(TAG,ERR,BUF)
      CALL FINIS
      END
```

REMOTE SYSTEM CONTROL VIA DS/1000

John A. Pezanno
Bill Reynolds
Howard Beyer/Holloman Air Force Base

INTRODUCTION

This article describes Holloman's experience providing data transfer capability to and from a remote site using the capabilities of the DS/1000 network software.

PROBLEM

The objective within our organization was to have the capability to transfer data tapes and disc files to and from another location 200 miles away for processing on CDC 6600/7600 computers. The tapes have multiple files of varying record lengths. The data has to be reliably transmitted or received. Partial tapes and files may also have to be sent. The two sites are connected by AT&T 209A modems through a noisy line. Each end of the line has a Hewlett-Packard HP 1000 system for the data transmission. At the remote end, the operators were totally unfamiliar with the HP system, and had neither the time nor the interest to learn how to use it. At the local end, the operators could be trained to operate the programs as necessary but it was desired to have as simple an operation as possible. The lack of remote handling meant that as much of the operation would have to be handled from the local end as possible, limiting the responsibilities of the remote operators to mounting and dismounting tapes. The local system is a disc-based RTE-IV system, but it is possible that if the disc crashed, the data transfer would still be necessary. The remote system is a memory based RTE-M3, with only a console and tape drives. Other disc-based remote systems will be added into the network in the future using the present local system as their connencting link to the remote.

APPROACH

The approach selected for this data transfer was to use DS/1000 modem capabilities using program-to-program communication. P-P was chosen over remote file access because of the versatility in error recovery, record length, and speed. At any time there is a possibility of problems at the remote end that might require local operator intervention. These problems include system failures, I/O device failure, and program errors. In order to provide complete local control of the remote system, and be able to obtain status information locally about the remote site, a number of utilities were required.

UTILITIES

A number of utilities were added to enable remote control. These are described below:

The program RBOOT will shut off the interrupt system and execute a 1060xx instruction, which causes the remote computer to execute its Remote Program Load (RPL) capability. RPL causes an automatic download of a new operating system. RBOOT is used when a change must be made to the system (e.g., replacement of the system software, or after an irrecoverable error.) Since RBOOT runs using the DS/1000 REMAT program, a complete system failure, or failure of DS/1000, would prevent the local operator from rebooting. In this case the remote operator must powerfail the computer by shutting it off, and then restart it manually.

The program SETHL is the startup program for the remote system. This program enables DS/1000 by scheduling the program LSTEN and passing it a class buffer to enable the communication link. SETHL then sends a message to the local console, stating that the remote system is up, and that the time is being set. This is followed by a DEXEC call to get the system time off of the local system, which is then formatted to ASCII. The message processor is called to set the remote clock. Since the program UPLIN is time scheduled when DS is enabled, the message processor is called to "ON,UPLIN,NOW". After this operation is completed, SETHL attempts to load the run-time programs. Up to five attempts to load each program will be made via calls to FLOAD. If there is a loader failure, or when the load is completed, an appropriate message is transmitted to the local node. SETHL uses the subroutine XMSG for printing all of its messages.

The program MSSEN runs at the local node. This program accepts four types of commands:

1. MSSEN will transmit any RTE command from the local system to the remote system using the DMESS routine provided by DS/1000. All messages returned as a result of these commands are printed on the local console. Examples of how this capability is useful include remote "UP"ing of devices, setting BREAK flags, etc.

2. MSSEN will also transmit any message to the remote node by typing "M,message" on the local terminal. The DMSG routine provided by the DS/1000 package is used to print the message on the remote console. The message will be preceded by the node number of the system from which the message was sent.

3. Any FMGR control command ("CN,p1,p2") can be tramsmitted via MSSEN. This enables remote tape drives to be positioned and EOF's written. In addition to all the standard CN commands available in File Manager, two have been added by MSSEN. They are "CN,lu,ST" and "CN,lu,DS" which get the status and dynamic status respectively of any device on the remote system. The format of the printout by these commands interprets the status information as if it were the status of a tape drive, showing whether the device is ON-LINE, READY, and if there is a WRITE RING regardless of what sort of device the LU actually is. In addition, the driver of the device is identified.

4. MSSEN will allow the last command issued to be reissued by typing "REDO" or "RE", saving the user the need for retyping the command.

MSSEN permits complete remote control of the tape units and permits remote RTE commands. Therefore, the only remaining requirement of the remote operator is that he must mount the tapes. A check can be made to see if the tape is mounted on the correct subchannel, and mounted correctly using the capabilities of MSSEN described above. Also, the tape can be repositioned, EOF's written, or the tape unit can be put off-line, all from the local node.

The subroutine XMSG is a utility subroutine designed to be used with the DS/1000 transmit and receive programs, but versatile enough to be usable with any DS/1000 program. XMSG has two entry points. The first, SETNL, is the entry point used to set the message node, message LU, and program name from which the message originated. The second, XMSG, is the message entry point. When XMSG is called, it prints the time in HH:MM:SS format with leading zeros printed, followed by the program name, the message, and optionally three labeled values and one unlabeled value. The labeled values are "FILE", "RECORD", and "ERROR". Negative values for the file or record will suppress these labels and values. Likewise, positive values for error will supress output of that label and value. This suppression feature permits inhibiting printing of items that occur prior to the ones intended to be printed. The unlabeled value will print positive decimal if positive, and positive octal if negative. The parameter following the message is the length of the message in words.

For example,

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

would print:

```
10:12:03 SETHL  FILE   5  ERR -4  ENCOUNTERED ON WRITE
```

The "RECORD" is suppressed since it is negative, as is the unlabeled value since it is not given. The message, although the first parameter in the call, is printed after the FILE, RECORD, and ERROR labels and values. The purpose of XMSG is to have the capability to print messages from a remote node to a local node without having the formatter in the system. Considerable space can be saved, which is especially important in the memory based system. In addition, all messages are printed in a consistent format, making them easier to read.

Our revised copy of the program WHZAT is a modified copy of the RTE-III WHZAT with the EXEC writes replaced by DEXEC calls. With this change, the WHZAT output can be directed to any node, while WHZAT runs in the remote node. Parameter P4 is the node (or −lu) number to which the output is directed, with the default value being the local node. This program permits the local operator to see the status of all programs and downed devices in the remote system. Using this version of WHZAT provides a much more complete system status than the REMAT STatus command.

Programs SLOAD and RLOAD respectively OFF (with an "OF,PROG,8") the send and receive programs, and reload them into the remote node using the FLOAD subroutine. They will continue to run until the load is successful, or until a BREAK is issued. Optionally, SLOAD and RLOAD can print the error message which is returned after each unsuccesful attempt.

The subroutine IUPIT is called when it is necessary to "UP" a device that is down. During a tape read, if a parity error occurs, the device will be downed. Any subsequent reads to the device will cause program suspension. Since the EQT, not the LU, of the "down" device must be "UPped", the subroutine finds the correct EQT, builds an UP, EQT command, and sends it to the remote site. In this way, the program calling IUPIT does not need to know the EQT , and IUPIT can be called with the LU of the down device.

PROGRAM DESIGN

With the design of these DS/1000 utilities complete, the programs to actually transfer data could be written. An important consideration in the writing of the programs was that each program be able to run at either a memory based or a disc based node without change in the code. To make the system easier for the operator to use, there had to be default logical units for all send and receive devices and destination nodes, however, these had to be easily changeable. The method chosen to implement the system was to have a number of different programs, as opposed to one large program for all functions. The main program establishes the options and configuration. There are separate programs for sending data and receiving data between tapes and disc files. It was realized that by using type 0 files, the difference between the tape and the disc could have been eliminated, but since the remote system has no file system whatsoever, this method was not possible. The method chosen allowed a stepped implementation of the software and permitted an operational tape to tape system in minimum time.

The program HLINK is the operator interface loaded into only the nodes at which we want to be able to initiate a transmission. The operator may run HLINK with the desired parameters in the run string, or interactively in which case he will be prompted for them. After defaults have been set or have been overridden, if a specific operation is requested by the operator (other than to reset the defaults) HLINK schedules the transmit program located at the transmit node, and passes to it the logical units, destination node, and message node. If parameters are not set, or if the operator desires to reset them, the current parameters are printed and the operator can then change them. The original or altered parameters are stored on a disc file. If the file does not exist, it is created, but if it cannot be created (because there is no file system) HLINK assumes the system is memory based and automatically stores the information internally. When the requested operation is completed, HLINK terminates.

The programs LUSEN and DISEN are scheduled without wait by HLINK at the node where the transmission is to occur from. After verifying that the transmitting device is ready, they establsh a communication link with the program LUREC for LU or file to LU transfers, or DIREC for LU to LU or file transfers. The operator may request special processing for the transfer, including repositioning of input and output devices, selective file and record copying, suppression of EOF's, and/or stacking of input tapes to a single output tape. The default condition for these programs is copying until a double EOF is found. Informatory and error messages are printed using the XMSG subroutine discussed earlier. All errors are handled by the programs and retried. In the case of unattended operation, printing of every DS error message that results from an unsuccessful transmission is suppressed. The first, second, fourth, eigth, etc. messages are the only ones printed. All messages go to the node which initiated the transmission (where HLINK was run).

The programs LUREC and DIREC write out the received records and send error messages to the scheduling node. They also reposition the output device if this option was requested.

PROBLEMS ENCOUNTERED

The main problems encountered in our system are due to noisy modem lines. The result is a considerable number of DS01 (data is not being recieved correctly), DS02 (communication line timeout), and DS05 (request timeout) errors. Another item we found to be a limitation were the DS/1000 error messages, which can only be sent to the system console. The capability to specify a system error device, particularly when there are a considerable number of errors would be helpful. A third, potentially critical problem, concerns the priority of the DS programs writing the error messages. If the system console is accidently shut off (as it was a few times at our remote site), the error messages cannot be printed. Ordinarily, on buffered devices, a program would be put in an I/O wait state when the buffer limits are reached. However, since the DS programs have a priority higher than

40, they will not suspend. Error messages continue to be stacked up in SAM until there is no SAM left. At that point it is impossible for the programs to operate any longer, or for the operator at the remote system to clear the system or even discover the problem. Since the system console must be an interactive terminal driver, its LU cannot be switched to the bit bucket. To eliminate error logging we wrote a dummy DVR00 which acts like the bit bucket, but which stores the most recent error message, internally overwriting old messages. Thus, the console can be switched to the dummy DVR00 when no error logging is desired. Eventually, we intend to write a program which will get the messages sent to the dummy DVR00 and transmit them to the local computer. This program will also be able to return messages to programs that do a read on the dummy driver.

THE FUTURE

We are presently working on integrating our HP 3000 through the local node so files can be sent between the HP 3000 and the remote node. Eventually, most of the local HP 1000 systems we have will be connected to the HP 3000 either directly, or through an intervening node. Our objective is to be able to transfer data from one node to any other node in our system via DS/1000, eliminating the need to physically carry tapes of data from data acquisition systems to data processing ones.

CONCLUSIONS

We have found DS/1000 to contain many versatile and useful features. We were able to get this complex system working in only a few days. The capability to completely control the remote system has considerably reduced the man-hours required to transmit data, and simplified our operation.

[*Editor's Note: Listings of the programs referenced above are included for use by any of our readers that would like to implement this system on their computer network. Please keep in mind that neither the Editor of the Communicator nor Hewlett-Packard can assume the responsibility of supporting these programs.*]

```
FTN4,L
      PROGRAM HLINK(3,60), MASTER SET DS1000  19 APRIL 1979 REV 1930 BMR
C*********************************************************************
C WRITTEN BY SGT BILLY M. REYNOLDS   APR 79   VERSION 1.0
C*********************************************************************
C
C THIS PROGRAM SETS OR CHANGES THE MASTER SETUP FOR THE DS/1000 LINK
C AT THIS END AND OPERATES THE DESIRED JOB BY RUNNING THE APPROPRIATE
C MASTER PROGRAM IN THE LINK
C
      DIMENSION IDCB(144)
      INTEGER QHL(6), CRTLU,IJOB,LNODE,NODE,IP(5),IKEEP,IARRY(5)
      INTEGER IRECLU,ITRNLU,DRECLU,DTRNLU,ITRANS,JRBUF(33),JBUFA(20)
      INTEGER SENDLU,RECVLU,KNODE
      EQUIVALENCE (IP(1),IJOB),(IP(2),ITRANS),(IP(3),SENDLU)
      EQUIVALENCE (IARRY(1),IRECLU),(IARRY(2),ITRNLU),(IARRY(3),DRECLU)
      EQUIVALENCE (IARRY(4),DTRNLU),(IARRY(5),NODE),(IP(4),RECVLU)
      EQUIVALENCE (IP(5),KNODE)
      DATA QHL/2H H,2HLI,2HNK,2H: ,2H ?,20137B/
      CALL RMPAR(IP)
   10 CRTLU=LOGLU(IDUM)
C
C SET UP OR OPEN THE MASTER SETUP FILE
C
   11 CALL OPEN(IDCB,IERR,6H!HLINK,0,2HHL)
      IF (IERR.GE.0) GO TO 13
      IF (IERR.EQ.-6)GO TO 15
      WRITE (CRTLU,12) IERR
   12 FORMAT ("HLINK: THE OPEN CALL FAILED DUE TO ERROR# ",I3)
      CALL CLOSE (IDCB)
      STOP 10
```

```
C
C IF SETUP FILE OPEN READ FILE, IF NOT ASK APPROPRIATE QUESTIONS TO
C SET UP THIS FILE
C
   13 CALL READF(IDCB,IERR,IARRY,5)
      IF(IERR.GE.0) GO TO 15
      WRITE(CRTLU,14)IERR
   14 FORMAT(/"HLINK: THE READF FAILED AND IS IN ERROR# ",I3)
      CALL CLOSE (IDCB)
      STOP 11
   15 IF (IJOB.NE.110) GO TO 100
C
C GIVE CURRENT RECEIVE LOGICAL UNIT OF LOCAL NODE AND ASK FOR CHANGE
C
   17 WRITE(CRTLU,20)IRECLU
   20 FORMAT(/" HLINK: THIS IS THE MASTER SETUP FOR THIS PROGRAM",/,
     *       "         IF YOU WISH TO CHANGE A LOGICAL UNIT,TYPE",/,
     *       "         IN THE NEW LOGICAL UNIT AFTER THE ? PROMPT",/,
     *       "         OR ELSE HIT THE SPACE BAR AND RETURN KEY ",/,
     *       "         TO KEEP THE CURRENT LOGICAL UNIT.",//,
     *       "         YOUR DEFAULT LOGICAL UNIT FOR RECEIVING",/,
     *       "         INFORMATION IS CURRENTLY LOGICAL UNIT ",I3)
0002  HLINK 10:45 AM  TUE.,  9  OCT., 1979
      CALL EXEC (2,CRTLU,QHL,6)
      READ (CRTLU,*)IRECLU
      WRITE(CRTLU,25)IRECLU
   25 FORMAT(/"HLINK: YOUR RECEIVE LOGICAL UNIT IS LU ",I3)
C
C GIVE CURRENT TRANSMIT LOGICAL UNIT FOR LOCAL NODE AND ASK FOR CHANGE
C
   30 WRITE(CRTLU,35)ITRNLU
   35 FORMAT(/" HLINK: YOUR DEFAULT LOGICAL UNIT FOR TRANSMITTING",/,
     *       "         INFORMATION IS CURRENTLY LOGICAL UNIT",I3)
      CALL EXEC (2,CRTLU,QHL,6)
      READ(CRTLU,*)ITRNLU
      WRITE(CRTLU,40)ITRNLU
   40 FORMAT(/"HLINK: YOUR RECEIVE LOGICAL UNIT IS LU ",I3)
C
C GIVE CURRENT RECEIVE LU FOR DESTINATION NODE AND ASK FOR CHANGE
C
   45 WRITE(CRTLU,50)DRECLU
   50 FORMAT(/" HLINK: YOUR DEFAULT LOGICAL UNIT FOR RECEIVING",/,
     *       "         INFORMATION AT THE DESTINATION NODE IS",/,
     *       "         CURRENTLY LOGICAL UNIT ",I3)
      CALL EXEC (2,CRTLU,QHL,6)
      READ(CRTLU,*)DRECLU
      WRITE(CRTLU,55)DRECLU
   55 FORMAT(/" HLINK: YOUR DESTINATION NODE RECEIVE LU# ",I3)
C
C GIVE CURRENT TRANSMIT LU FOR DESTINATION NODE AND ASK FOR CHANGE
C
   60 WRITE(CRTLU,65)DTRNLU
   65 FORMAT(/"HLINK: YOUR DEFAULT LOGICAL UNIT FOR TRANSMITTING",/,
     *       "         AT THE DESTINATION NODE IS LOGICAL UNIT ",I3)
      CALL EXEC (2,CRTLU,QHL,6)
      READ(CRTLU,*)DTRNLU
      WRITE(CRTLU,70)DTRNLU
   70 FORMAT(/" HLINK: YOUR DESTINATION NODE TRANSMIT LU# ",I3)
C
C GIVE CURRENT DESTINATION NODE NUMBER AND ASK FOR CHANGE
C
      WRITE (CRTLU,80)NODE
   80 FORMAT(/"HLINK: YOUR DESTINATION NODE NUMBER IS "I3)
      CALL EXEC (2,CRTLU,QHL,6)
      READ(CRTLU,*)NODE
      WRITE(CRTLU,81)NODE
   81 FORMAT(/"HLINK: YOUR DESTINATION NODE NUMBER IS ",I3)
```

```
C
C CREATE DISK FILE FOR MASTER SETUP IF ONE NOT ALREADY CREATED
C
      IF (IERR.GE.0) GO TO 83
      CALL CREAT(IDCB,IERR,6H!HLINK,1,1,2HHL)
      IF (IERR.GE.0) GO TO 84
      WRITE(CRTLU,82)IERR
   82 FORMAT("HLINK: CANNOT CREATE DISK FILE DUE TO ERROR# ",I3,/,
     *       "HLINK: SO INPUT WILL BE KEPT IN MEMORY INSTEAD")
      GO TO 100
   83 CALL RWNDF(IDCB)
0003  HLINK 10:45 AM  TUE.,  9  OCT., 1979
   84 CALL WRITF(IDCB,IERR,IARRY,5)
      IF(IERR.GE.0) GO TO 90
      WRITE(CRTLU,85)IERR
   85 FORMAT("HLINK: CANNOT PERFORM WRITF DUE TO ERROR# ",I3)
      CALL CLOSE (IDCB)
      STOP 12
   90 CALL CLOSE(IDCB)
C
C CHECK FOR OPENED DISK FILE, IF NOT GO BACK AND CREATE ONE
C
  100 IF(IRECLU.EQ.0)GO TO 17
      IF(SENDLU.EQ.0)GO TO 111
      ITRNLU=SENDLU
      DTRNLU=SENDLU
  111 IF(RECVLU.EQ.0)GO TO 112
      IRECLU=RECVLU
      DRECLU=RECVLU
  112 IF(KNODE.EQ.0)GO TO 113
      NODE=KNODE
  113 IF (IJOB.GT.110) GO TO 120
C
C CHECK FOR DESIRED JOB TO BE PERFORMED
C
  114 WRITE(CRTLU,115)
  115 FORMAT (/" HLINK: TYPE THE NUMBER CODE IN FOR THE JOB TYPE.",/,
     *        "JOB TYPES:        0 TO TERMINATE PROGRAM ",/,
     *        "                110 TO SET OR CHANGE DEFAULTS ",/,
     *        "                111 LU RECEIVE,LU TRANSMIT ",/,
     *        "                112 LU RECEIVE, DISK TRANSMIT",/,
     *        "                113 DISK RECEIVE, LU TRANSMIT",/,
     *        "                114 DISK RECEIVE, DISK TRANSMIT",/,
     *        "                141 LU TRANSMIT,LU RECEIVE ",/,
     *        "                142 LU TRANSMIT,DISK RECEIVE ",/,
     *        "                143 DISK TRANSMIT,LU RECEIVE ",/,
     *        "                144 DISK TRANSMIT, DISK RECEIVE")
  116 CALL EXEC (2,CRTLU,QHL,6)
      DO 117 I=1,20
      JBUFA(I)=2H
      JRBUF(I)=2H
  117 CONTINUE
  118 CALL EXEC(100001B,CRTLU+400B,JBUFA,-20)
      GO TO 185
  119 CALL ABREG(IA,LEN)
      IF(LEN.LT.1)GO TO 116
      CALL PARSE(JBUFA,LEN,JRBUF)
      IP(1)=JRBUF(2)
      IF(JRBUF(33).LT.2)GO TO 120
      IP(2)=JRBUF(6)
      IF (JRBUF(33).LT.3)GO TO 120
      IP(3)=JRBUF(10)
      IF (JRBUF(33).LT.4)GO TO 120
      IP(4)=JRBUF(14)
      IF(JRBUF(33).LT.5)GO TO 120
      IP(5)=JRBUF(18)
  120 IF (IJOB.EQ.0) GO TO 999
      IF(SENDLU.EQ.0)GO TO 122
      ITRNLU=SENDLU
      DTRNLU=SENDLU
  122 IF(RECVLU.EQ.0)GO TO 123
      IRECLU=RECVLU
      DRECLU=RECVLU
  123 IF(KNODE.EQ.0)GO TO 124
      NODE=KNODE
  124 CALL CLOSE (IDCB)
  125 CALL GNODE(LNODE)
```

```
C
C GO TO THE APPROPRIATE SECTION FOR EACH PARTICULAR JOB
C
  150 IKEEP=IJOB-139
      GO TO (180,250,350,450,550,114)IKEEP
  180 IKEEP=IJOB-108
      GO TO (114,11,200,300,400,500,114)IKEEP
  185 CALL ABREG(IA,IB)
      WRITE (CRTLU,190)IA,IB
  190 FORMAT (/"HLINK: CANNOT READ INFORMATION DUE TO ERROR ",2A2)
      GO TO 999
C
C JOB 111 TO BE PERFORMED, EXECUTE LUSEN PROGRAM AT DESTINATION NODE
C
  200 CALL DEXEC(NODE,10+100000B,6HLUSEN ,DTRNLU,IP(2),IRECLU,LNODE,
     1LNODE)
      GO TO 220
  210 STOP 13
  220 CALL ABREG(IA,IB)
      WRITE(CRTLU,230)NODE,IA,IB
  230 FORMAT(/"HLINK: LUSEND AT NODE "I2" IS UNSCHEDULED DUE TO ",/,
     *        "HLINK: ERROR ",2A2," ABORT ")
      GO TO 999
C
C JOB 141 TO BE PERFORMED, EXECUTE LUSEN PROGRAM AT LOCAL NODE
C
  250 CALL EXEC(10+100000B,6HLUSEN ,ITRNLU,IP(2),DRECLU,NODE,LNODE)
      GO TO 270
  260 STOP 14
  270 CALL ABREG(IA,IB)
      WRITE(CRTLU,280)IA,IB
  280 FORMAT(/"HLINK: LUSEND IS UNSCHEDULED DUE TO ERROR ",2A2,/,
     *        "HLINK: HLINK IS ABORTING")
      GO TO 999
C
C JOB 112 TO BE PERFORMED, EXECUTE DISEN PROGRAM AT DESTINATION NODE
C
  300 CALL DEXEC(NODE,10+100000B,6HDISEN ,2,IP(2),IRECLU,LNODE,LNODE)
      GO TO 320
  310 STOP 15
  320 CALL ABREG(IA,IB)
      WRITE(CRTLU,330)NODE,IA,IB
  330 FORMAT (/"HLINK: DISEND AT NODE "I2" IS UNSCHEDULED DUE TO",/,
     *        "HLINK: ERRORS ",2A2," HLINK IS ABORTING")
      GO TO 999
C
C JOB 142 TO BE PERFORMED, EXECUTE LUSEN PROGRAM AT LOCAL NODE
C
350   CALL EXEC(10+100000B,6HLUSEN ,ITRNLU,IP(2),2,NODE,LNODE)
      GO TO 370
  360 STOP 16
  370 CALL ABREG(IA,IB)
      WRITE(CRTLU,380)IA,IB
  380 FORMAT (/"HLINK: LUSEND IS UNSCHEDULED DUE TO ERROR ",2A2,/,
     *        "HLINK: HLINK IS ABORTING")
      GO TO 999
C
C JOB 113 TO BE PERFORMED, EXECUTE LUSEN PROGRAM AT DESTINATION NODE
C
400   CALL DEXEC(NODE,10+100000B,6HLUSEN ,DTRNLU,IP(2),2,LNODE,LNODE)
      GO TO 420
  410 STOP 17
  420 CALL ABREG(IA,IB)
      WRITE(CRTLU,430)NODE,IA,IB
  430 FORMAT(/"HLINK: LUSEND AT NODE "I2" IS UNSCHEDULED DUE TO",/,
     *        "HLINK: ERROR ",2A2," HLINK IS ABORTING")
      GO TO 999
```

```
C
C JOB 143 TO BE PERFORMED, EXECUTE DISEND PROGRAM AT LOCAL NODE
C
450   CALL EXEC(10+100000B,6HDISEN ,2,IP(2),DRECLU,NODE,LNODE)
      GO TO 470
  460 STOP 21
  470 CALL ABREG(IA,IB)
      WRITE(CRTLU,480)IA,IB
  480 FORMAT (/"HLINK: DISEND IS UNSCHEDULED DUE TO ERROR ",2A2,/,
     *        "HLINK: HLINK IS ABORTING")
      GO TO 999
C
C JOB 114 TO BE PERFORMED, EXECUTE DISEND PROGRAM AT DESTINATION NODE
C
500   CALL DEXEC(NODE,10+100000B,6HDISEN ,2,IP(2),2,LNODE,LNODE)
      GO TO 520
  510 STOP 22
  520 CALL ABREG(IA,IB)
      WRITE(CRTLU,530)NODE,IA,IB
  530 FORMAT(/"HLINK: DISEND AT NODE "I2" IS UNSCHEDULED DUE TO ",/,
     *        "HLINK: ERROR ",2A2," HLINK IS ABORTING")
      GO TO 999
C
C JOB 144 TO BE PERFORMED, EXECUTE DISEND PROGRAM AT LOCAL NODE
C
550   CALL EXEC(10+100000B,6HDISEN ,2,IP(2),2,NODE,LNODE)
      GO TO 570
  560 STOP 23
  570 CALL ABREG(IA,IB)
      WRITE(CRTLU,580)IA,IB
  580 FORMAT(/"HLINK: DISEND IS UNSCHEDULED DUE TO ERROR ",2A2,/,
     *        "HLINK: HLINK IS NOW ABORTING")
  999 END
      END$
```

```
ASMB,L
      NAM XMSG,7 REV 1933 FORMATS & SENDS MESSAGES FOR HLINK
      ENT XMSG,SETNL
      EXT EXEC,DEXEC,.MVW,.ENTR,KCVT,.DFER,CNUMD,CNUMO

*     THIS SUBROUTINE PRINTS MESSAGES FOR THE DATA LINK UTILITY
*     WITHOUT USING FORMATTER.  MESSAGES ARE IN THE FORMAT:
*
*     HH:MM:SS YYYYYY  FILE XXX  REC WWWW  ERR-VV MESSAGE UUUU
*
*     WHERE:
*     HH              IS HOUR (LOCAL CPU TIME)
*     MM              IS MINUTE
*     SS              IS SECOND
*     YYYYYY          IS SIX CHARACTER PROGRAM NAME
*     XXX             IS FILE #
*     WWWW            IS RECORD #
*     VV              IS NEGATIVE ERROR #
*     UUUUU           IS OPTIONAL #
*                      POSITIVE FOR DECIMAL
*                      NEGATIVE FOR OCTAL (CHANGED TO +)
*
*     TO CALL:
*
*     CALL SETNL (NODE,LU,6HPROGNM)
*
*     TO INITIALIZE NUMBERS. IF SETNL IS NOT CALLED FIRST, THE
*     DEFAULTS ARE USED. SETNL MAY BE RECALLED AT ANY TIME TO
*     CHANGE ANY VALUES
*
*     PARAMETERS ARE
*
*     NODE            IS NODE WHERE MESSAGE IS WRITTEN. DEFAULT LOCAL (-1)
*     LU              IS LOGICAL UNIT FOR MESSAGE. DEFAULT IS 1
*     PROGNM          IS 6 CHARACTER ARRAY. DEFAULT IS BINARY 0 (NULLS)
```


```
*
*     TO CALL THE MESSAGE TRANSMITTER
*
*     CALL XMSG (ARRAY,LENGTH,FILE,RECORD,ERROR,OPTION)
*
*     WHERE
*
*     ARRAY          CONTAINS MESSAGE TO BE PRINTED
*     LENGTH         IS ARRAY LENGTH (MUST BE IN WORDS)
*     FILE           IS FILE # TO BE PRINTED (SUPPRESSED IF NEGATIVE)
*     RECORD         IS RECORD # (SUPPRESSED IF NEGATIVE)
*     ERROR          IS 2 DIGIT ERROR CODE (SUPPRESSED IF POSITIVE)
*     OPTION          OR OCTAL IF MINUS (CONVERTED TO POSITIVE BEFORE PRINTING)
*     THE ONLY REQUIRED PARAMETERS ARE ARRAY AND LENGTH. ALL OTHERS ARE
*     OPTIONAL. IF THEY ARE SET TO NEGATIVE (FOR RECORD & FILE) OR
*     POSITIVE (FOR ERROR), THAT PORTION OF MESSAGE IS INHIBITED.
*     TRAILING PARAMETERS MAY BE LEFT OFF.
*
*     NOTE: IF THE LAST CHARACTER IN THE MESSAGE ARRAY IS A "_"
*     (SUPPRESS RETURN/LINE FEED CHARACTER) AND THERE IS NO
*     OPTIONAL PARAMETER, THE RETURN/LINE FEED WILL BE SUPPRESSED.
*
* SETNL INITIALIZES FILE, REC, ERROR, NUMBR ADDRESSES TO ZERO.
* USED "OCT 0" INSTEAD OF "BSS 1" FOR ABOVE.
*
*     SET "NO ABORT" BIT ON DEXEC WRITE. CONTENTS OF "A"
*     AND "B" REGISTERS CAN BE OBTAINED BY CALL TO "ABREG"
*     TO FIND IF CALL WAS SUCESSFUL.
 NODE  DEC -1
 LU    DEC 1
 PROG  DEF PROGN
 SETNL NOP
       JSB .ENTR     GET PARAMETERS
       DEF NODE       AND STORE
       LDA NODE,I    GET PARAMETER
       LDB LU,I            "
       DST NODE       AND STORE LOCALLY
       JSB .DFER     MOVE PROGRAM NAME TO
       DEF PROGN      LOCAL BUFFER
       DEF PROG,I
       CLA
       STA FILE
       STA REC
       STA ERROR
       STA NUMBR
       JMP SETNL,I    THEN EXIT
 ARRAY BSS 1          ADDRESS OF MESSAGE ARRAY
 LNGTH BSS 1          LENGTH OF ABOVE ARRAY
 FILE  OCT 0          FILE COUNT TO BE OUTPUT
 REC   OCT 0          RECORD COUNT TO BE OUTPUT
 ERROR OCT 0          ERROR TO BE PRINTED
 NUMBR OCT 0          ADDITIONAL NUMBER DESIRED
 XMSG  NOP
       JSB .ENTR     GET ADDRESS OF PARAMETERS
       DEF ARRAY
       JSB EXEC
       DEF *+3
       DEF D11
       DEF OUT
*ADD 100 TO MINUTES AND SECONDS SO LEADING ZERO
*WILL BE PRINTED, SINCE WE ONLY USE LAST 2
*NUMBERS ANYWAY.
       LDA OUT+1
       ADA D100
       STA OUT+1
       LDA OUT+2
       ADA D100
       STA OUT+2
```

```
*FORMAT TIME
       JSB KCVT       GET HOURS
       DEF *+2
       DEF OUT+3
       STA OUT         AND STORE
       JSB KCVT       GET SECONDS
       DEF *+2
       DEF OUT+1
       STA OUT+3       AND STORE
       JSB KCVT       GET MINUTES
       DEF *+2
       DEF OUT+2
       LDB A::        GET COLONS TO SURROUND
       RRR 8           THE MINUTES
       DST OUT+1       AND STORE
*INITIALIZE COUNT
       LDA D8         SET # OF WORDS TO 8
       STA KOUNT
       ADA OUTP       SET BUFFER POINTER
       STA OUTPR       TO NEXT WORD
       LDA BLANK      BLANK OUT WORDS
       STA OUT+4       FIVE TO END
       LDA OUTP5       TO CLEAR BUFFER
       STA B
       INB
       JSB .MVW
       DEF D35
       NOP
       JSB .DFER      MOVE PROGRAM NAME INTO
       DEF OUT+5       WORDS 6-8
       DEF PROGN
*GET FILE
       LDA FILE       GET FILE ADDRESS
       SZA,RSS         WAS IT GIVEN?
       JMP MSG         NO! LAST PARAMETER
       LDA A,I        GET VALUE
       SSA            NEGATIVE?
       JMP GETR       YES! GO TO RECORD HANDLER
       STA OUT+39     NO! STORE TEMPORARILY
       JSB CNUMD      MAKE ASCII
       DEF *+3
       DEF OUT+39      FROM
       DEF OUT+30      AND STORE TEMPORARILY
       DLD FIL        GET ASCII "FILE"
       DST OUT+9
       DLD OUT+31     GET LOW 4 BYTES OF FILE #
       DST OUT+11
       LDA KOUNT      GET WORD COUNT
       ADA D5         ADD 5 MORE
       STA KOUNT       AND STORE
       ADA OUTP       SET POINTER
       STA OUTPR
 GETR  CLA            ZERO OUT FILE LOCATION IN
       STA FILE        CASE ITS NOT GIVEN NEXT TIME
*RECORD HANDLER
       LDA REC         SAME AS FILE ABOVE
       SZA,RSS
       JMP MSG
       LDA A,I
       SSA
       JMP GETE
       STA OUT+39
       JSB CNUMD
       DEF *+3
       DEF OUT+39
       DEF OUT+30
       JSB .DFER      MOVE 6 NUMBERS
       DEF OUT+15
       DEF OUT+30
       DLD RECD       MOVE "REC" AND
       DST OUT+14      OVERLAY TOP 2 NUMBERS
       LDA *-1        MOVE TEMPORARY BUFFER TO
       LDB OUTPR       PROPER LOCATION
       INB
       JSB .MVW
```

```
        DEF D6
        NOP
        LDA KOUNT       BUMP COUNTER
        ADA D5
        STA KOUNT
        ADA OUTP         AND POINTER
        STA OUTPR
 GETE   CLA
        STA REC
*ERROR HANDLER
        LDA ERROR       SAME AS FILE HANDLER
        SZA,RSS
        JMP MSG
        LDA A,I
        SSA,RSS          EXCEPT DON'T WRITE IF POSITIVE
        JMP MSG
        CMA,INA         MAKE POSITIVE
        STA OUT+39
        JSB .DFER
        DEF OUT+18
        DEF ERR
        JSB KCVT
        DEF *+2
        DEF OUT+39
        STA OUT+21
        LDB OUTPR
        LDA D18P
        JSB .MVW
        DEF D4
        NOP
        LDA KOUNT
        ADA D4
        STA KOUNT
        CLA
        STA ERROR
*MESSAGE HANDLER
 MSG    LDA ARRAY
        LDB LNGTH,I
        STB LNGTH
        LDB KOUNT
        ADB OUTP
        INB
        JSB .MVW
        DEF LNGTH
        NOP
        LDA KOUNT
        ADA LNGTH
        INA
        STA KOUNT
*EXTRA NUMBER
        LDA NUMBR
        SZA,RSS
        JMP WRIT
        LDA A,I
        SSA             NEGATIVE?
        JMP OCTAL       YES! TREAT AS POSITIVE OCTAL
        STA OUT+39
        JSB CNUMD
        DEF *+3
        DEF OUT+39
        DEF OUT+36
        JMP FIXN
 OCTAL  CMA,INA
        STA OUT+39
        JSB CNUMO
        DEF *+3
        DEF OUT+39
        DEF OUT+36
 FIXN   LDA KOUNT
        ADA OUTP
        STA ADD
        LDA KOUNT
        ADA D3
        STA KOUNT
        JSB .DFER
```

```
 ADD    DEF A,I
        DEF OUT+36
        CLA
        STA NUMBR
*WRITE  OUTPUT ARRAY
 WRIT   JSB DEXEC
        DEF *+6
        DEF NODE
        DEF WRITI       WRITE COMMAND
        DEF LU          OUTPUT LU
        DEF OUT         OUTPUT ARRAY
        DEF KOUNT       LENGTH IN WORDS
        NOP
        JMP XMSG,I
*CONSTANTS AND STORAGE
        SUP PRESS
 A      EQU 0
 B      EQU 1
 WRITI  OCT 100002      WRITE W/ NO ABORT
 D3     DEC 3
 D4     DEC 4
 D5     DEC 5
 D6     DEC 6
 D8     DEC 8
 D11    DEC 11
 D35    DEC 35
 D100   DEC 100
 A::    ASC 1,::        USED FOR TIME
 BLANK  ASC 1,          USED TO BLANK FILL BUFFER
 RECD   ASC 2,REC       PRINT "REC "
 FIL    ASC 2,FILE      PRINT "FILE"
 ERR    ASC 3,  ERR-    PRINT "ERR" AND MINUS SIGN
 PROGN  BSS 3           ADDRESS WHERE PROGRAM NAME IS STORED
 KOUNT  BSS 1           RUNNING COUNT OF OUTPUT BUFFER LENGTH
 OUTP   DEF OUT         PERMANENT POINTER TO START OF BUFFER
 OUTP5  DEF OUT+4       PERMANENT POINTER TO WORD 5
 018P   DEF OUT+18      PERMANENT POINTER TO WORD 18
 OUTPR  BSS 1           TEMPORARY POINTER TO CURRENT BUFFER LO
 OUT    BSS 40          BUFFER
        END XMSG
```

```
FTN4,L
      PROGRAM MSSEN(3,60), SENDS COMMANDS DS 1000 REV 1935 05SEP79
C
C         RUNNING MESSN
C
C         TO RUN, MESSN, TYPE "RU,MESSN" AND ANSWER QUESTIONS.
C OPTIONALLY, TYPE "RU,MSSEN,NODE".
C
C         REMOTE RTE COMANDS
C
C  REMOTE RTE COMMANDS ARE THE SAME AS LOCAL COMMANDS. NO "*" OR
C  "LU>" PROMPT IS NECESSARY. ANY RETURN MESSAGES, WILL BE RETURNED
C  TO YOU. ANY VALID RTE COMMAND CAN BE GIVEN.
C
C         MESSAGES TO REMOTE SYSTEM CONSOLE
C
C  TO SEND A MESSAGE TO THE REMOTE SYSTEM CONSOLE, TYPE:
C
C  "M,MESSAGE
```

```
C
C  THE SYSTEM CONSOLE AT THE OTHER END WILL SHOW:
C
C        =N   NN:MESSAGE
C
C  WHERE NN IS YOUR NODE
C
C        FMGR CONTROL COMMANDS
C
C  FILE MANAGER CONTROL COMMANDS TO REWIND, FORWAD SPACE, ETC A
C  DEVICE CAN BE GIVEN. THE FMGR COMMANDS ARE GIVEN AS FOLLOWS:
C
C                      CN,LU,CM
C
C  WHERE LU IS LOGICAL UNIT AND CM IS COMMAND
C
C  THE FOLLOWING ARE LEGAL COMMANDS:
C
C  RW OR 4             REWIND DEVICE
C  5                   REWIND AND SET TO STANDBY
C  EO OR 1             WRITE EOF ON DEVICE
C  DS OR 6             READ DYNAMIC STATUS OF DEVICE
C  ST                  GIVES LAST STATUS OF DEVICE
C  FF OR 13            MOVES DEVICE FORWARD 1 FILE
C  BF OR 14            MOVES DEVICE BACK 1 FILE
C  FR OR 3             MOVES DEVICE FORWARD 1 RECORD
C  BR OR 2             MOVES DEVICE BACK 1 RECORD
C  TO                  SETS DEVICE AT TOP OF FORM
C  NOTE: DYNAMIC STATUS REQUEST WILL READ THE CURRENT STATUS DIRECTLY
C  FROM THE DEVICE. HOWEVER, IF THE DEVICE IS DOWN OR LOCKED, THE
C  REQUEST CANNOT BE HONORED. THE STATUS REQUEST WILL GET THE STATUS
C  OF THE DEVICE OBTAINED FROM THE LAST REQUEST TO IT.
C
C                   REDO COMMAND
C
C THE PREVIOUS COMMAND CAN BE RETRIED BY TYPING "RE" OR "REDO". THE
C PREVIOUS COMMAND WILL BE ECHOED TO THE INPUT DEVICE.
C
C
C                      ERROR MESSAGES
C
C  DS ERRORS
C  DS01, DS02, DS05       MEAN PROBLEMS WITH THE LINE. RETRY. IF THE
C                         ERROR CONTINUES, GET HELP.
C  DS04                   ILLEGAL NODE IN RESPONSE TO FIRST QUESTION.
C                         EXIT PROGRAM AND RESTART GIVING GOOD NODE.
C  DS08                   DS LINE PROBLEMS OR DEVICE IS LOCKED BY
C                         REMOTE PROGRAM. CAN OCCUR ON CN,LU,DS COMMAND
C  IOXX                   SEE DS1000 MANUAL FOR ERROR
C  "ILLEGAL CODE"         YOU ENTERED A NON-EXISTANT CONTROL COMMAND. RETRY.
C
C THIS PROGRAM HAS THE CAPACITY TO CONTROL DS/1000 OPERATIONS BY
C SENDING COMMANDS TO THE DESTINATION NODE IN PERFORMANCE OF CERTAIN
C OPERATIONS.
C
      DIMENSION IBUF(20),IBUF2(20)
      INTEGER IX(2)
      INTEGER PARM(5)
      EQUIVALENCE (IX,X)
      INTEGER ILEN,INODE,IPBUF(33),LUIN(3),IPRAM,CRTLU
      EQUIVALENCE (IPBUF(6),LUIN,LU),(IPBUF(14),IPRAM)
      DATA INODE/-1/
      CALL RMPAR (PARM)
      CRTLU=LOGLU(IDUM)
```

```
C
C DETERMINE THE DESTINATION NODE AND COMMAND TO BE TRANSMITTED
C
      WRITE(CRTLU,20)
   20 FORMAT (" MSSEN REV 1935")
      INODE=PARM
      IF ((PARM.NE.CRTLU).AND.(PARM.NE.0)) GO TO 39
      INODE=-1
      WRITE (CRTLU,38)
38    FORMAT(" WHAT IS YOUR DESTINATION NODE ? ")
      READ(CRTLU,*)INODE
39    WRITE(CRTLU,40)
   40 FORMAT(" TYPE COMMAND TO BE TRANSMITTED ?")
70    IPBUF=IBUF
      IPBUF(2)=IBUF(2)
      CALL EXEC (1,CRTLU+400B,IBUF,20)
      IOLD=I
      CALL ABREG (IA,I)
      IF ((IBUF.EQ.2H  ).OR.(I.EQ.0)) GO TO 860
      IF (IBUF.NE.2HRE) GO TO 45
      IBUF=IPBUF
      IBUF(2)=IPBUF(2)
      I=IOLD
      CALL EXEC (2,CRTLU,IBUF,I)
45    ILEN=I*2
C
C     COPY MESSAGE TO XMIT BUFFER
C
      DO 47 J=1,20
47    IBUF2(J)=IBUF(J)
C
C CHECK FOR TYPE OF COMMAND. IF CONTROL COMMAND USE EXEC CALLS.
C IF MESSAGE, PASS TO REMOTE
C
      IF(IBUF.EQ.2HCN)GO TO 340
      IF (IBUF.EQ.2HM,) GO TO 800
      CALL DMESS(INODE,IBUF2,ILEN)
      CALL ABREG(IA,IB)
      IF(IA.GE.0) GO TO 220
      IF(IB.EQ.-1)GO TO 180
      IA=-IA
      IA=(IA+1.0) / 2.0
C
C PRINT TO TERMINAL ANY RETURNED MESSAGES FROM DESTINATION NODE
C
      WRITE(CRTLU,160)(IBUF2(K),K=1,IA)
  160 FORMAT (" RETURNED MESSAGE: ",20A2)
      GO TO 220
  180 WRITE(CRTLU,200)(IBUF2(K),K=1,2)
  200 FORMAT(" TRANSMIT COMMAND FAILED DUE TO ERROR ",2A2)
C
C REQUEST NEXT COMMAND IF ANY OR TERMINATE PROGRAM
C
220   WRITE(CRTLU,260)
  260 FORMAT(" TYPE NEXT COMMAND OR HIT SPACE BAR AND RETURN KEY ",/,
     *       " FINISH ?")
      GO TO 70
C
C DETERMINE THE TYPE OF CONTROL COMMAND
C
  340 CALL PARSE(IBUF2,ILEN,IPBUF)
C
C CALL TO REWIND TAPE AT DESTINATION NODE
C
      IF(IPBUF(10).NE.2HRW) GO TO 380
      CALL DEXEC(INODE,100003B,LUIN+400B)
      GO TO 820
  360 GO TO 220
```

```
C
C CALL TO MAKE END OF FILE ON TAPE AT DESTINATION NODE
C
  380 IF(IPBUF(10).NE.2HEO)GO TO 420
      CALL DEXEC(INODE,100003B,LUIN+100B)
      GO TO 820
  400 GO TO 220
C
C     DYNAMIC STATUS CALL
C
  420 IF ((IPBUF(10).NE.2HDS).AND.(IPBUF(10).NE.6)) GO TO 520
      CALL DEXEC (INODE,100003B,LUIN+600B)
      GO TO 820
  440 CALL ABREG (IA,IB)
441   X=0.0
      IF (IAND(IA,100B).EQ.0) X=4HNOT
      WRITE (CRTLU,460) LU,IX
  460 FORMAT (I3," IS ",2A2,"AT LOAD POINT")
      X=0.0
      IF (IAND(IA,4).NE.0) X=4HNOT
      WRITE (CRTLU,480) LU,IX
  480 FORMAT (I3," DOES ",2A2,"HAVE WRITE RING")
      X=0.0
      IF (IAND(IA,1).NE.0) X=4HNOT
      WRITE (CRTLU,500) LU,IX
  500 FORMAT (I3," IS ",2A2,"ON LINE")
      IA=IAND (IA,37400B)/256
      WRITE (CRTLU,123) LU,IA
123   FORMAT (" LOGICAL UNIT ",I2," USES DV.",O2)
      GO TO 220
C
C     NON-DYNAMIC STATUS CALL
C
520   IF (IPBUF(10).NE.2HST) GO TO 561
      CALL DEXEC (INODE,100015B,LUIN,IA)
      GO TO 820
567   GO TO 441
C
C CALL TO SHIFT TO TOP OF PAGE ON LINE PRINTER AT DESTINATION NODE
C
  561 IF(IPBUF(10).NE.2HTO)GO TO 560
      CALL DEXEC(INODE,100003B,LUIN+1100B,IPRAM)
      GO TO 820
  540 GO TO 220
C
C CALL TO FORWARD FILE ON TAPE AT DESTINATION NODE
C
  560 IF(IPBUF(10).NE.2HFF)GO TO 600
      CALL DEXEC(INODE,100003B,LUIN+1300B,IPRAM)
      GO TO 820
  580 GO TO 220
C
C CALL TO BACKSPACE FILE ON TAPE AT DESTINATION NODE
C
  600 IF(IPBUF(10).NE.2HBF)GO TO 640
      CALL DEXEC(INODE,100003B,LUIN+1400B,IPRAM)
      GO TO 820
  620 GO TO 220
C
C CALL TO FORWARD RECORD ON TAPE AT DESTINATION NODE
C
  640 IF(IPBUF(10).NE.2HFR)GO TO 680
      CALL DEXEC(INODE,100003B,LUIN+300B,IPRAM)
      GO TO 820
  660 GO TO 220
C
C CALL TO BACKSPACE RECORD ON TAPE AT DESTINATION NODE
C
  680 IF(IPBUF(10).NE.2HBR)GO TO 720
      CALL DEXEC(INODE,100003B,LUIN+200B,IPRAM)
      GO TO 820
  700 GO TO 220
```

```
C
C     CALL TO CONTROL TAPE AT DESTINATION NODE WITH NUMERIC CODE
C
  720 IF (IPBUF(10).GT.27) GO TO 760
      CALL DEXEC (INODE,100003B,LUIN+IPBUF(10)*100B,IPRAM)
      GO TO 820
  740 GO TO 220
C
C     IMPROPER CALL
C
  760 WRITE(CRTLU,780)IPBUF(10)
  780 FORMAT(" FUNTION CODE ",A2," IS AN ILLEGAL CODE. ")
      GO TO 220
C
C     MESSAGE TRANSMITTER
C
  800 CALL DMESG (INODE,IBUF2(2),I-1)
      GO TO 220
  820 CALL ABREG(IA,IB)
      WRITE(CRTLU,840)IA,IB
  840 FORMAT(" COULD NOT PERFORM EXEC CALL DUE TO ERROR ",2A2)
      GO TO 220
  860 END
      END$


FTN4,L
      PROGRAM RLOAD (3,65), LOAD LUREC TO NODE 30 REV 1932 2 AUG 79
C
      INTEGER IOFF(10), PARM(5)
      DATA IOFF/2HOF,2H,L,2HUR,2HEC,2H,8/
C
      CALL RMPAR(PARM)
C
      ICRT = LOGLU (ID)
      WRITE (ICRT,10)
   10 FORMAT(" RLOAD   REV 1932")
C
    40 CALL DMESS (30,IOFF,10)
C
    60 ITRY = 0
    80 ITRY = ITRY + 1
      CALL FLOAD (6HLURECD,48,10,30,IERR,2HHL,1,7)
      IF (IERR.EQ.0) GO TO 90
      IF (PARM(2).NE.0) WRITE (ICRT,85) IERR, ITRY
    85 FORMAT(" RLOAD: FAIL. IERR =",I5,"  TRY =",I3)
      IF (IFBRK(ID)) 120,80
C
   90 WRITE (ICRT,100) ITRY
  100 FORMAT(" RLOAD: LUREC LOADED. TRYS = ",I5)
  120 END
      END$

ASMB,L
    NAM IUPIT,7 REV 1924 - UPS LU'S
    ENT IUPIT
    EXT .ENTR,KCVT,MESSS
*     THIS ROUTINE WILL CALCULATE THE EQT FOR ANY LU IN THE SYSTEM
*     AND THEN ISSUE A CALL TO THE MESSAGE PROCESSOR TO "UP" THE
*     EQT. IF THE ROUTINE IS CALLED AS A FUNCTION, THE RETURNED
*     VALUE WILL CONTAIN A 0 IF THE COMMAND WAS SUCCESSSFUL. ANY
*     NON-ZERO VALUE INDICATES THAT THE SYSTEM TRIED TO RETURN A
*     MESSAGE WHICH MEANS THAT THERE WAS AN ERROR.
*
*     TO CALL
*
*                   CALL IUPIT(LU)
*
*     WHERE LU IS THE LOGICAL UNIT WHOSE EQT IS TO BE UPPED.
*
*
*     TO CALL AS FUNCTION:
*
*                   I=IUPIT(LU)
*
```

```
*     WHERE THE VALUE RETURNED IN I IS 0 FOR SUCCESSFUL UP, OR
*     NON-ZERO FOR FAILURE.
*
*
*     NOTE: NO CHECK IS MADE TO SEE IF THE EQT WAS IN FACT DOWN.
*     THE DEVICE REFERENCE TABLE IS SEARCHED FOR THE EQT LINKED TO
*     THE REQUESTED LU. THE COMMAND IS CONFIGURED AND ISSUED. THE
*     VALUE RETURNED FROM MESSS IS PASSED TO THE CALLER IN THE "A"
*     REGISTER.
 LU     BSS 1
 IUPIT  NOP
        JSB .ENTR
        DEF LU
        CCA             GET A MINUS 1
        ADA DRT          AND ADD TO FW OF DRT
        ADA LU,I         AND ADD TO LOGICAL UNIT
        LDA A,I          GET WORD 1 OF LU
        AND B77         MASK EQT#
        STA M+2          AND STORE TEMPORARILY
        JSB KCVT        GET ASCII EQUIVALENT
        DEF *+2          OF EQT#
        DEF M+2
        STA M+2         STORE IN MESSAGE ARRAY
        DLD UP          GET FIRST PART OF MESSAGE
        DST M           NOW HAVE "UP,EQT#"
        JSB MESSS       CALL MESSAGE PROCESSOR
        DEF *+3
        DEF M           ADDRESS
        DEF D6          # OF CHARACTERS
        JMP IUPIT,I     EXIT W/RETURN FROM MESSS IN "A"
 DRT    EQU 1652B       ADDRESS OF FW OF DRT
 A      EQU 0
 UP     ASC 2,UP,       UP MESSAGE
 M      BSS 6           ALLOW 6 WORDS IN CASE OF RETURN MESSAGE
 D6     DEC 6
 B77    OCT 77
        END
```

```
ASMB,L
      NAM DVD00  RTE DUMMY CONSOLE DRIVER 30CT79 REV 1935
      ENT ID00, ID01, CD01
ID00  NOP              ENTRY POINT TO WRITE FOR CRT
      LDA EQT6,I     GET FUNCTION
      SLA            READ OR CONTROL?
      JMP BAD0       YES! SEND IMMEDIATE COMPLETION
      LDA EQT8,I      NEGATIVE LENGTH (CHARACTERS)
      SSA,RSS
      JMP C.1         NO
      CMA,INA        MAKE LENGTH POSITIVE
      SLA            ODD #?
      INA            YES. INCREMENT BY ONE
      ARS            DIVIDE BY 2
C.1   STA #WORD
      ADA =D-41       CHECK ON LENGTH OF WRITE
      SSA
      JMP C.2         LENGTH < 40, OK
      LDA =D40
      STA #WORD
C.2   LDA EQT7,I      MOVE MESSAGE FROM USER BUFFER
      LDB ABUF          TO DRIVER BUFFER
      MVW #WORD
      ISZ #WORD       ADD 1 WORD FOR MISS COUNTER
      LDA EQT        GET TIMEOUT OF DVD01
      SZA,RSS         IS IT THERE?
      JMP NOTIM      NO!
      CCA            SET TO -1
      STA EQT,I
      CLA            CLEAR ADDRESS TO SHOW DONE
      STA EQT
NOTIM ISZ COUNT       KEEP TRACK OF HOW MANY TIME DRIVER CALL
BAD0  CLB             RETURN ZERO LENGTH RECORD ON READ
      LDA =B4
      JMP ID00,I      DONE
```

```
ID01  NOP             ENTRY POINT TO READ FROM CRT
      LDA EQT6,I      GET FUNCTION
      ARS             GET BIT1 (0 = READ. 1 = WRITE/CNTL)
      SLA             READ?
      JMP BAD1        NO! EXIT
      LDA COUNT        SEE IF DRIVER HAS BUFFER TO WRITE
      SSA
      JMP WAIT         NO
      LDA ACOUN       MOVE BUFFER FROM DRIVER
      LDB EQT7,I       TO USER
      MVW #WORD
      CCA              SET MISS BUFFER TO NONE
      STA COUNT
      LDA =B4
      LDB #WORD
      JMP ID01,I
WAIT  LDA =B10000     SET DRIVER TO HANDLE TIME OUTS
      IOR EQT4,I
      STA EQT4,I
      LDA EQT15       GET ADDRESS OF TIMEOUT COUNTER
      STA EQT          AND STORE LOCALLY
      CLA
      STA EQT15,I
      JMP ID01,I
BAD1  CLB
      LDA =B4
      JMP ID01,I
CD01  NOP
      LDA EQT1,I      CHECK IF REQUEST WAS PENDING
      SZA,RSS
      JMP EXIT        NO!
      LDA ACOUN        MOVE BUFFER FROM DRIVER TO USER
      LDB EQT7,I
      MVW #WORD
      LDB #WORD
EXIT  CCA              SET MISS BUFFER TO NONE
      STA COUNT
      CLA
      JMP CD01,I
*
*   CONSTANT AND STORAGE AREA
*
ABUF  DEF BUF          ADDRESS OF DRIVER BUFFER
ACOUN DEF COUNT
COUNT DEC 0            COUNTER OF MISS BUFFERS
BUF   BSS 40           DRIVER BUFFER
#WORD BSS 1            NUMBER OF WORDS STORED IN BUFFER
EQT   DEC 0           RESERVED FOR EQT15 OF READ
*
*    BASE PAGE AND COMMUNICATION AREA DEFINITION
*
.     EQU 1650B
EQT1  EQU .+8
EQT4  EQU .+11
EQT6  EQU .+13
EQT7  EQU .+14
EQT8  EQU .+15
EQT9  EQU .+16
EQT14 EQU .+84
EQT15 EQU .+85
      END
```

```
ASMB,L
      NAM RBOOT,3 CAUSES REBOOT IN 21MX-E W/ RPL
      ENT RBOOT
      EXT $LIBR
*     THIS PROGRAM EXECUTES A "106022" HALT TO CAUSE A
*     REBOOT IN ANT 21MXE WITH RPL ENABLED.
RBOOT NOP
      JSB $LIBR
      NOP
      OCT 106022
      END RBOOT
```

AN INTRODUCTION TO OPERATING SYSTEMS FUNDAMENTALS

Gary McCarney/HP Rockville MD

INTRODUCTION

Users of mini-computer systems are often assumed to have a complete understanding of the concepts of multiprogramming, swapping, priorities, file management, what the operating system does for the user, etc. Quite often these users don't understand these concepts and their subsequent utilization of the system suffers. This article will explain these concepts, beginning on a very elementary level, and gradually building to describe fully the operations of a disc-based operating system. The reader must tolerate a certain amount of poetic license at the beginning, such as programs getting into memory by some "magic".

SINGLE PROGRAM SYSTEM

Let's begin by considering what the computer's memory "looks like" and how it is used. Figure 1 is a representation of the computer's memory (note that memory is always drawn as a rectangle). Memory locations are referenced by sequential octal numbers called addresses. Let us assume that we have a computer connected to a terminal device. In the lower addresses of memory there is an operating system. The purpose of our operating system is to allow the user at the terminal to interact with the computer using English-like commands. Using this computer and operating system, we choose to develop a computer program which will read the temperature of an oven — such as an integrated circuit oven — and if the result is not within the preprogrammed limits, send a control signal to the oven to make the necessary adjustments. By using "magic", we install this program into our computer's memory, as shown in Figure 2. To execute our program, we simply indicate to the system to RUN our program. The system knows where in memory programs begin, location 34000_8 for example, and control is transferred to begin executing the program. Each time we type RUN, the program measures the temperature and, if necessary, makes a correction.

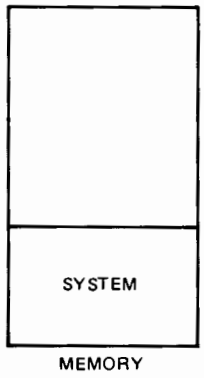


Figure 1

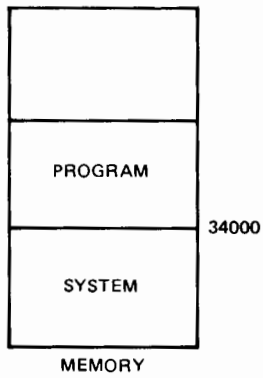


Figure 2

MULTIPLE PROGRAM SYSTEM

With the temperature monitoring program working, perhaps the need develops for another program that collects data from an experiment or production line. We wish to install this program into computer memory and since there is room in memory above the temperature program, we can have both programs in memory at the same time. However, there is a problem whenever we wish to execute the second program since the system will always automatically transfer control to memory location 34000_8 each time we type RUN. Some way is required to indicate to the system which of the two programs to execute. Since one of our first

premises was to allow the user to interact with the computer using English-like commands, we need to assign a name to each program. Let's call the temperature monitoring program TEMP and the data collection program COLEK. Typing RUN, COLEK should execute our data collection program. In order for the system to know what we are trying to do, we need to create space within the system area where the names of the programs currently in memory are stored, along with their starting addresses. Since this table contains unique identifying information about a particular program, let's call this the ID Table. Both TEMP and COLEK would need a segment in the ID Table and as more and more programs are added, the number of segments would increase accordingly. (See Figure 3.) Let's call this table then the ID Segment Table.

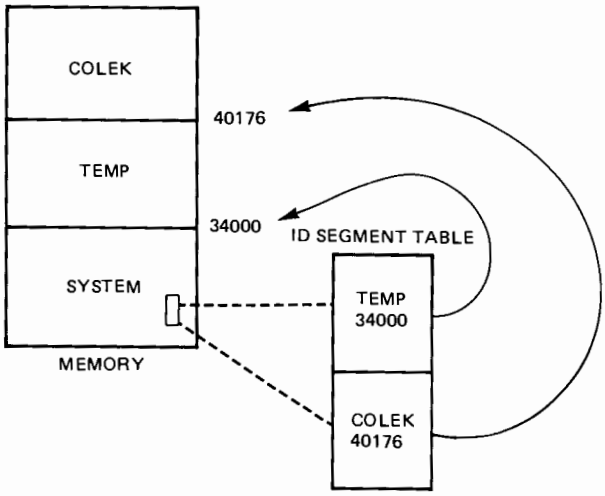


Figure 3

PRIORITIES

If we assume that the program COLEK normally executes for 10 minutes and has been running for about 2 minutes when you type RUN, TEMP, what happens? Since COLEK has control of the machine, TEMP simply waits for COLEK to complete before TEMP executes. Fine. What could go wrong in a critical temperature environment in 8 minutes? If the temperature we are reading is that of a boiler, the worst that can happen is that the boiler gets angry and blows up! Some way is needed to indicate that some programs should get preferential treatment, a form of priority. We obviously want the boiler program to run quickly, but what happens to the two minutes of data that has been collected by COLEK? Since COLEK and TEMP reside in different parts of memory, the data should not be affected within one area by the execution of a program in another area of memory. Our only concern is with the working registers within the computer itself; e.g., the A register. If the contents of these registers is saved for COLEK, then execution can be resumed later where it left off by simply restoring the registers. If the number of memory locations for each ID Segment were increased by the number of working registers, the system could copy the contents of these registers into these locations before control is passed to a higher priority program. A program's priority could also be kept within the program's ID Segment. Let's increase the intelligence of our system by having it check for a program's priority each time someone types RUN into the terminal. After a higher priority program completes, the system could restore the registers from the ID Segment and transfer control to the proper location of memory to continue execution.

Suppose a high priority program is waiting for some action to occur; e.g., a user to type some answer. While this high priority program is waiting for the response, the entire computer is idle. Why not let the next highest priority program which is waiting its turn execute while this waiting is happening. In this way we do not waste any of the capabilities of the computer system. With our scheme of saving the registers within the program's ID Segment, we only encounter the minimal overhead of copying the registers. Both of these examples illustrate the concept known as Multiprogramming; i.e., sharing usage of the computer resources between programs.

With the importance of the temperature program, we obviously will be running it often whenever the oven or boiler is operating. It becomes somewhat ridiculous to imagine sitting at the terminal typing RUN, TEMP every minute to keep our operation functioning normally. What is needed is some way to have TEMP run automatically at some interval — in this case, once a minute. If we add a clock to our system, we could store the next time TEMP is supposed to run into part of the ID Segment. The system could check the next time for execution by examining the ID Segment, comparing it to the clock, and if the times match, automatically execute the TEMP program.

SWAPPING

Any computer installation seems to generate more and more opportunities for programmatic solutions to problems. Now that our system is handling the two tasks outlined above, we have a need for a program that collects some data, reduces the data, and produces a final report. We write the program and when we try to put it into memory, there simply is not enough room. Now what? To install this new program, call it program PRODT, we must overwrite one or more existing programs in memory. If PRODT, can get enough memory space by overwriting COLEK, then only one program needs to be reinstalled later. But if we assume that the "magic" we have been going through to install a program in memory is time-consuming, then we might want to find some short-cut way to reinstall our program to get the maximum usefullness from our computer system. The disc can be used to store programs that are ready to execute (known as absolute or executable programs), and we can increase the smarts in our system to allow storing these programs onto the disc. The trade-off we must make will be between letting a program reside permanently in memory and letting it reside on the disc until it is to be executed. When a disc resident program must execute, it first must be copied from the disc into memory before control is transferred to begin execution.

Since there is a typical delay of between 25 and 50 milliseconds to access a particular area of the disc, we would only put programs on the disc that would not suffer from this delay. Perhaps our temperature monitoring program might be more helpful if it was always in memory — especially if if we decided to run it at a very high rate, say every 10 milliseconds, but we don't mind taking some extra time to move PRODT and COLEK back and forth between memory and disc.

With the addition of the disc, both programs COLEK and PRODT would have their executable form of the program stored on the disc and more information would be added to the ID Segment to contain the disc address for the program. See Figure 4. Now

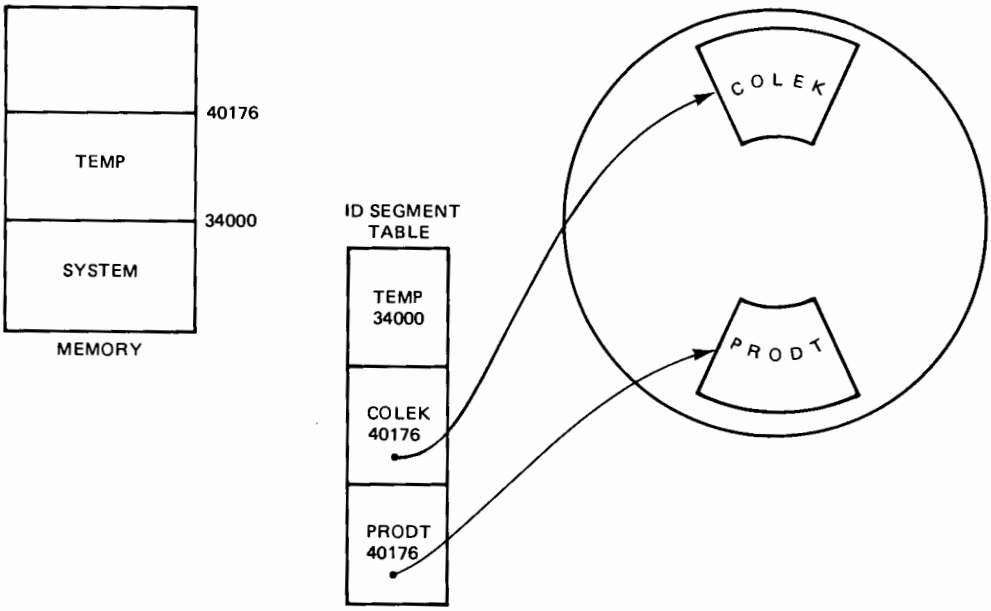


Figure 4

when we type RUN, PRODT the system first checks the ID Segment Table for a match of the name. After PRODT is found, the system must check to see if it is in memory or disc resident. If disc resident, its location on the disc must be found. The executable code must be copied from the disc into the proper portion of memory and then control transferred to the starting address found in the ID Segment. See Figure 5. If program PRODT is executing and normally runs for 20 minutes with a low priority, what happens if you type RUN, COLEK, and COLEK has a higher priority than PRODT? This case is significantly different than the previous example, since for COLEK to execute it would have to be read from the disc into memory, overwriting PRODT, and destroying the data that PRODT has collected or calculated so far. This is not acceptable.

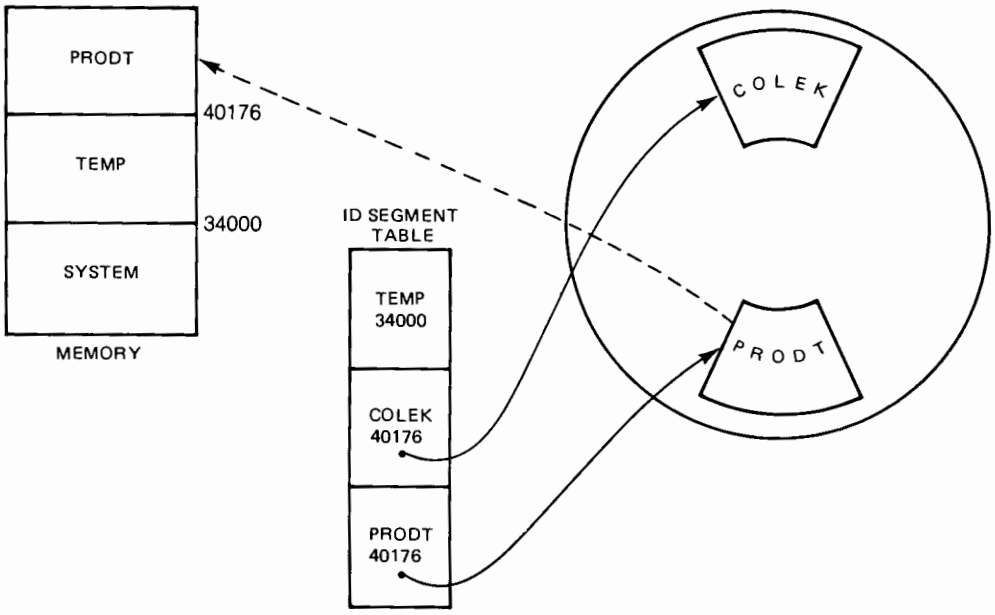


Figure 5

The system must first save the working registers into PRODT's ID Segment and then temporarily copy all of PRODT onto the disc before we can bring COLEK into memory. This means that at some moment in time there will be two copies of PRODT on the disc. One is the original and the other is in some partially executed state. The last thing the system must do is put some reminder into the ID Segment for PRODT that a partially executed copy is saved on the disc, and also record where it is saved. See Figure 6. Only after PRODT is copied to the disc and the registers have been saved can COLEK be read in from the disc and begin to execute. See Figure 7. When COLEK completes its execution there is no need anymore to save a partially executed copy of PRODT and COLEK can be overwritten in memory. The system now brings the partially executed copy of PRODT back into memory from the disc, releases the space used on the disc for the temporary storage and removes the information about the partial copy (registers, etc.) from the ID Segment. See Figure 8. This is the concept of SWAPPING. There is one more element of bookkeeping that must be done by the operating system. Space on the disc must be controlled to prevent overwriting original or partially executed programs. Within the system we add another table known as the Track Assignment Table (TAT) which is used by the system to maintain control over the space on the disc. Since disc space is divided into concentric circles, and you could think of one of these circles as a track, depending on the information stored within the TAT, it is easy to see if a track is either free or in use.

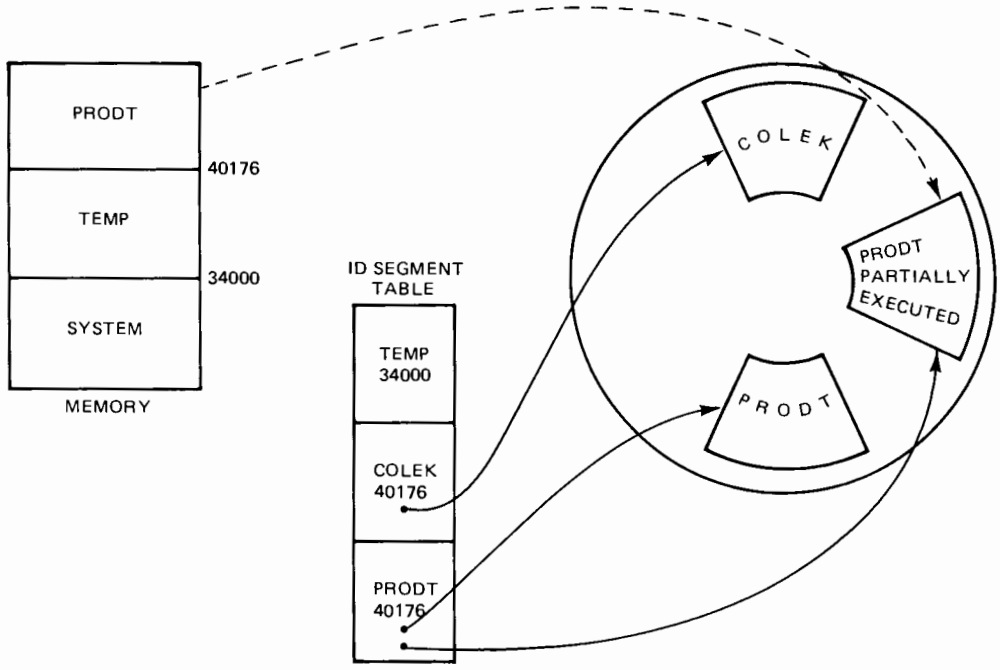


Figure 6

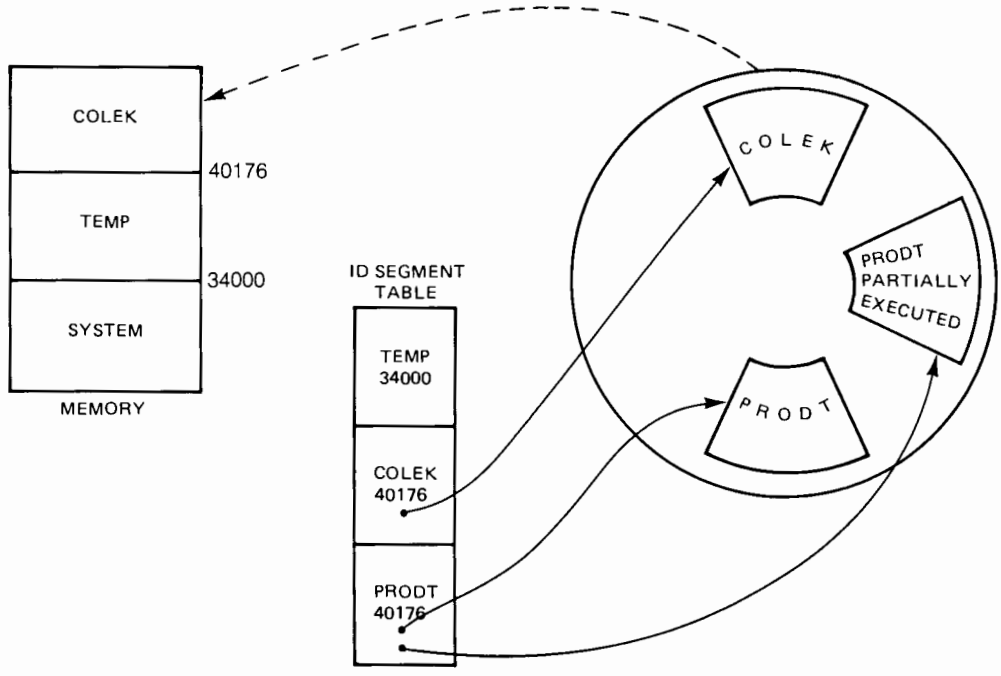


Figure 7

SHARING DATA

There will be times when you may wish to pass some of the data collected or reduced by one program, to another program. If both of these programs are permitted access to an area of memory that is common to both, then data can be exchanged. In many systems, there is an area set aside within the memory that is denoted as a System Common Area and is used for this purpose.

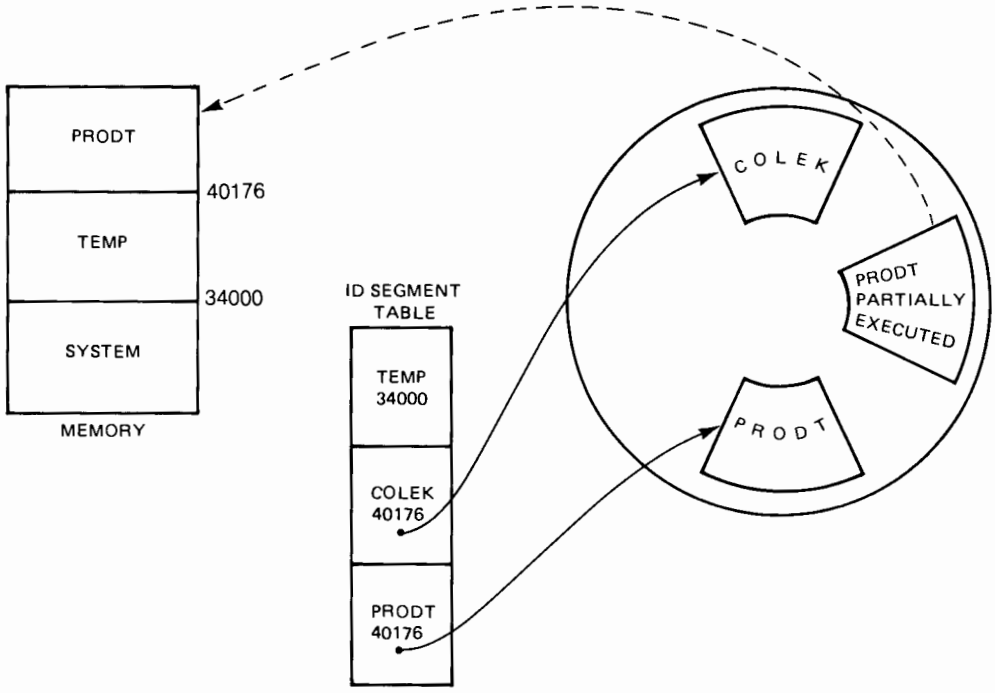


Figure 8

COPY OF SYSTEM ON DISC

Now that we have a computer disc connected to our system, let's look for other uses that we might make of all that space. As you are probably aware, the memory that is used for today's computers is designed such that if the electrical power is removed — such as turning the computer off for the weekend — all of the contents of memory are lost. That means on Monday morning you must first install the complete operating system and all memory resident programs into memory before you can resume your normal operations. This could consume a lot of time and resources. With this computer disc we have attached, why not store the complete operating system and all memory resident programs onto the disc and design an easy method of bringing the operating system into memory? On the 1000 series computer you can have a very small program stored onto an integrated circuit chip that can be read into memory by pressing one of the front panel switches — the IBL. This small program is known formally as the binary loader and informally as the "boot". The main purpose of the boot is to read another small program, known as the bootstrap from the disc. The bootstrap is slightly larger than the boot and has more smarts. It, in turn, loads the operating system and memory resident programs from their tracks on the disc into memory, and transfers control to the operating system for normal operation. This all takes place in a matter of seconds.

Our computer's disc storage is now used for the operating system and memory resident programs, as well as copies of both the original and partially executed disc resident programs. What remains to be added is a simple method for adding programs to our system. Let's first add an Interactive Editor that permits us to type in our original program, called the source program. In order for the editor to be capable of both short and long source programs, some area other than memory should be used. If the editor program requests some work space on the disc from the system, all that the system needs to do is search the TAT for an available track, or tracks, assign it to the editor to use, and return the track address to the editor.

FILE MANAGED AREA

When we finish entering our source program, we would like to save this source where it can be easily retrieved for updating. The easiest way to do this is to add to our system a file management utility, which permits the user to save program code by simply specifying some name to be associated with the code. This file management utility is simply another disc resident program in our operating system. Let's call the utility FMGR and assign some portion of the disc for its use. Since the system resides on the outer most tracks, let's use the innermost section, say 100 tracks, for FMGR to store our source programs according to the names we have given them. Recall that the TAT must be made aware of this space that has just been reserved. Figure 9 illustrates this disc configuration.

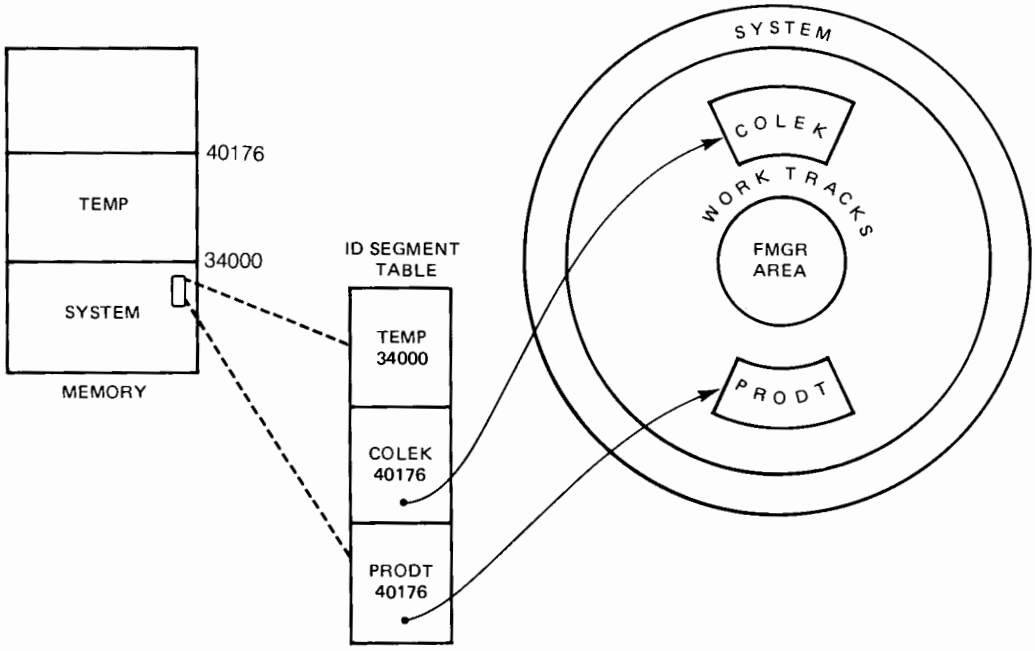


Figure 9

CREATING EXECUTABLE CODE

Now that the source code is saved in the FMGR area, the next step is to compile the source (assuming the source code is in FORTRAN) and create relocatable code. Relocatable code is a translation from a high level language (such as FORTRAN) to a simple pattern of 1's and 0's that the computer can understand. This relocatable code, which is generated by the compiler, FTN4, could also be stored into the FMGR area under some new name. The final step is converting the relocatable into executable code using a utility program designed for this purpose, let's call it LOADR.

THE LOADR program does several things. First, the program relocatable code is converted into executable code, complete with all of the proper memory addresses. Next, the loader adds various system and user supplied library programs to create the fully executable version of our program. Finally, an ID Segment is created with as much of the information described above as possible, including a pointer to the location in the work space area of the disc where this new executable program was created.

CONCLUSION

This article has followed the evolution of an operating system from a single program, single purpose computer system to a simplified RTE multiprogramming disc operating system. The need for ID Segments, program priorities, swapping, memory and disc resident programs, as well as utilities, was developed and explained. The reader should now be able to better understand more complex operating systems, such as RTE-IVB, which extend the concepts covered here to include mapping, memory partitioning, session monitor, and time-slicing, to name just a few. •

EASY FORMS FOR THE 2645A

Todd Field/HP, Woodbury, N.Y.

Introduction

This article describes a program to read a form, complete with escape sequences, from a 264x terminal into a file.

Frank Slootweg's article, "Block Mode Input with 264x Series Terminals", from Volume III issue 2 of the COMMUNICATOR/1000 shows a sample program to write a form onto a 264x terminal and read the data from the terminal in block mode. Following his directions, a programmer can easily make use of the protected/unprotected fields, block mode, and many of the other features of the 264x series terminals. The one step remaining to be performed is to create the form and access the form from a program.

There are several different methods of creating the form itself. Each method has its own advantages and drawbacks. One method is to hard code the escape sequences and ASCII characters which make up the form, directly into the FORTRAN program, as Frank Slootweg does in his examples. This method is the easiest for simple forms, but grows rapidly in complexity with the size of the form. Alternately, one could use an intermediate language and some translation subroutines. Alex Swartz, in his article "Software for the 2645 Terminal" in Volume II issue 5 of the COMMUNICATOR/1000, uses this approach. Alex Swartz points out the many advantages of this method. The disadvantages are that one needs to keep in mind the relation between the intermediate language and the escape codes for the terminal, and that if a new escape sequence is ever introduced, changes must be made in the intermediate language. Perhaps the most straightforward method of making a form is to simply create it on the 264x. By using the softkeys to define the unprotected fields, the definition process can be shortened to a matter of minutes. The only remaining step is to store the form somehow from the terminal to the HP/1000 in such a manner that it can be read by a program and written back out to the terminal. If the screen could be "dumped" to a file, this problem would be overcome.

Right now there is no easy method of storing a form from the terminal into a file. A form on the terminal could be recorded off-line to a cassette and read in using FMGR commands, but there are problems with this approach. The FMGR :ST and :DU commands could be used to read the cassette, but blank lines would be deleted, upsetting the spacing. A program to read the form directly off the screen would have to fiddle with the block mode strappings and this would be a little tricky to write. A bonus in a programmatic approach is that additional escape sequences can be put in the file, releasing the programmer from some busy work. In addition, as the form would not be in the program itself, a correspondingly larger program could be written. Figure 1 is a short program, F2645, to do just this.

```
0001  FTN4,L
0002        PROGRAM F2645(3,99),2645A FORM READ PROGRAM  TF  790822
0003  C
0004  C     PROGRAM TO READ A FORM FROM A 2645A TERMINAL
0005  C
0006  C     WRITTEN BY  TODD FIELD, HP/WOODBURY  8/22/79
0007  C
0008  C     (LINES WITH ESCAPE SEQUENCES ARE BRACKETED BY ESCY ESCZ)
0009  C
0010        INTEGER INBUF(80),IDCB(144),IPBUF(10),BLINES,LINENO,TLOG,
0011       +        PREBUF(10),POSTBF(4),PRELEN,POSTLN
0012  C$Y %
0013        DATA PREBUF/2H$c,2H$X,2H$m,2H$h,2H$J,2H$&, 2H=0,2HD$,2H&k,2H1B/%
0014       +     ,POSTBF/2H$W,2H$b,2H $,2Hh_/%
0015       +     ,PRELEN,POSTLN/10,4/%
0016  C$Z
0017        LU=LOGLU(IDUM)
```

```
0018  C
0019  C      READ IN FILE NAME
0020  C
0021   10    WRITE(LU,100)
0022   100   FORMAT(/," FILE NAME? _")
0023         READ(LU,200) INBUF
0024   200   FORMAT(80A2)
0025         IF (INBUF(1).EQ.2H/E)  GOTO 9900
0026         I=1
0027         CALL NAMR(IPBUF,INBUF,20,I)
0028         IF (IAND(IPBUF(4),3).NE.3)   GOTO 10
0029  C
0030  C      ATTEMPT TO OPEN FILE
0031  C
0032         CALL OPEN(IDCB,IERR,IPBUF,0,IPBUF(5),IPBUF(6))
0033         IF (IERR.EQ.-6)  GOTO 300
0034            IF (IERR.GE.0)  GOTO 400
0035               WRITE(LU,9999) IERR
0036               GOTO 10
0037  C
0038  C         CREATE FILE
0039  C
0040   300      CALL CREAT(IDCB,IERR,IPBUF,24,30,IPBUF(5),IPBUF(6))
0041            IF (IERR.GE.0)  GOTO 400
0042               WRITE (LU,9999) IERR
0043               GOTO 10
0044  C
0045  C      FILE OPEN.  DUMP MESSAGE AND SOFTKEYS TO TERMINAL.
0046  C      ALSO, DUMP DISABLE, FORMAT OFF, MEM LOCK OFF, HOME, CLEAR,
0047  C                  LINE MODE, BLOCK MODE TO FILE.
0048  C
0049   400   CALL WRITF(IDCB,IERR,PREBUF,PRELEN)
0050         IF (IERR.NE.0)  GOTO 9000
0051         WRITE(LU,410)
0052  C%Y%
+0053    410  FORMAT("%h%J%&f1a1k 7L%&dB[%[", %
+0054         +            "%&f1a2k 7L%]]%&d@", %
+0055         +            "%&f1a3k 6L%&dB%[", %
+0056         +            "%&f1a4k 6L%]%&d@", %
+0057         +       //,"F1 - START UNPROTECTED FIELD WITH '['", %
+0058         +       //,"F2 - STOP UNPROTECTED FIELD WITH ']'", %
+0059         +       //,"F3 - START UNPROTECTED FIELD WITHOUT '['", %
+0060         +       //,"F4 - STOP UNPROTECTED FIELD WITHOUT ']'")%
+0061   C%Z
0062         CALL EXEC(12,0,2,0,-4)
0063  C
0064  C      BLOCK/LINE, BLOCK MODE, NO TIMEOUT, UPDATE TERMINAL CONFIGURATION
0065  C
0066         WRITE (LU,420)
0067  C%Y %
+0068    420  FORMAT("%h%J%&s0D%&k1B_") %
+0069   C%Z
0070         CALL REIO(3,LU,22B,0)
0071         CALL REIO(3,LU,25B,0)
0072  C
0073  C      WAIT FOR USER TO PRESS ENTER KEY
0074  C
0075         CALL REIO(1,LU,INBUF,1)
```

```
0076  C
0077  C      USER HAS PRESSED ENTER.  DISABLE KEYBOARD AND PREPARE TO LOOP
0078  C
0079         WRITE (LU,510)
0080  C$Y %
$0081    510  FORMAT("$c$h_") %
$0082  C$Z
0083         BLINES=0
0084         LINENO=-1
0085  C
0086  C      LOOP WHILE BLANKLINES < 5
0087  C
0088   600   CONTINUE
0089            LINENO=LINENO+1
0090  C
0091  C         READ A LINE
0092  C
0093            DO 605,I=6,80
0094    605        INBUF(I)=2H
0095            WRITE(LU,610) LINENO
0096  C$Y %
$0097    610     FORMAT("$&a",I2,"r0C$d_")%
$0098  C$Z
0099            CALL REIO(1,LU+1000B,INBUF(6),75)
0100            CALL ABREG(IA,TLOG)
0101            IF ((TLOG.GT.0).AND.(.NOT.((TLOG.EQ.1).AND.(INBUF(6).EQ.2H
0102           +     ))))  GOTO 700
0103  C
0104  C            BLANK LINE
0105  C
0106               BLINES=BLINES+1
0107               GOTO 1000
0108    700    CONTINUE
0109  C
0110  C            NON-BLANK LINE
0111  C
0112               BLINES=0
0113  C
0114  C            INSERT POSITIONING AND WRITE TO FILE
0115  C
0116               CALL CODE
0117               WRITE(INBUF,710) LINENO
0118  C$Y %
$0119    710      FORMAT("$&a ",I2,"r00C")%
$0120  C$Z
0121               CALL WRITF(IDCB,IERR,INBUF,TLOG+5)
0122               IF (IERR.NE.0)  GOTO 9000
0123    1000 IF (BLINES.LT.5)  GOTO 600
0124  C
0125         GOTO 9100
0126  C
0127  C      ERROR
0128  C
0129    9000 WRITE (LU,9999) IERR
0130         GOTO 9105
0131  C
0132  C      ALL DONE.  WRITE HOME, FORMAT ON, ENABLE KEYBOARD TO FILE.
0133  C      CLOSE FILE. PUT TERMINAL IN PAGE MODE AND CHARACTER MODE.
0134  C      ENABLE KEYBOARD FROM TERMINAL AND UPDATE CONFIGURATION.
0135  C
```

```
0057*
0058  00047 076006R MOVE  STB BSTRT     WE FOUND A UNIT SEPERATOR SO
0059  00050 060001        LDA B          SAVE THE ADDRESS AND
0060  00051 003000        CMA INA          FIGURE OUT HOW MANY
0061  00052 042002R       ADA TEMPA          WORDS WE WANT TO MOVE
0062  00053 072010R       STA BCNT             AND SAVE THIS
0063  00054 060001        LDA B          B IS DESTINATION AND
0064  00055 002004        INA             A IS SOURCE AND
0065  00056 105765        MBT BCNT         MOVE BYTES!
      00057 000010R
      00060 000000
0066  00061 026032R       JMP CHECK      GO BACK AND TRY IT AGAIN
0067                      END
```

Figure 5.

The Subroutine

FFORM is a simplified version of Frank Slootweg's program A2645. Now, all that is necessary for the programmer to do to display a form, read in the data, reset the terminal or any combination thereof, is to call FFORM as follows:

```
CALL FFORM(INAME,ICR,INBUF,INLEN,IMODE,LU)
```

where:

INAME is the file name the form is in
ICR is the file's cartridge reference number
INBUF will contain the data input by the user
INLEN is the length of INBUF and is returned as the length of the data read.
IMODE is the functions FFORM is to perform, depending on which bits are set, the following actions will occur:
- bit 2: display form on screen
- bit 1: read data from screen
- bit 0: reset terminal

LU is the logical unit where the form will be displayed and the data read.

To understand what IMODE is used for, consider the following scenarios:

1. A form is to be displayed, the data is read and the terminal reset.IMODE would be 7 for this application.

2. A form is to be displayed and the form is to be filled out many times. IMODE would be set to 4 to display the form, 2 to read the data in, and finally set to 1 to reset the terminal.

Remember that a unit seperator will be inserted by the 2645 terminal between each unprotected field. FFORM calls a subroutine USTRP (figure 5) to strip the unit seperators out of the input, but the programmer must pass FFORM a buffer large enough to take this into consideration.

Figure 6 contains a program to illustrate how FFORM works.

```
0001  FTN4,L
0002        PROGRAM TOAD
0003  C
0004  C     TEST FOR FFORM
0005  C
0006        INTEGER INBUF(200)
0007  C
0008        LU=LOGLU(IDUM)
0009  C
0010  C     WRITE FORM TO TERMINAL
0011  C
0012        IERR=0
0013        CALL FFORM(6HTOADF ,0,INBUF,IERR,4,LU)
0014        IF (IERR.LT.0)  GOTO 9000
0015  C
0016  C     READ DATA
0017  C
0018        ILEN=200
0019        CALL FFORM(0,0,INBUF,ILEN,2,LU)
0020        WRITE(6,10) (INBUF(I),I=1,ILEN)
0021   10   FORMAT(" INPUT BUFFER WAS: &,4(/,60A2))
0022  C
0023  C     READ DATA AGAIN
0024  C
0025        ILEN=200
0026        CALL FFORM(0,0,INBUF,ILEN,2,LU)
0027        WRITE (6,10) (INBUF(I),I=1,ILEN)
0028  C
0029  C     RESET TERMINAL
0030  C
0031        CALL FFORM(0,0,0,0,1,LU)
0032  C
0033  C     DO A COMBINED WRITE/READ/RESET
0034  C
0035        ILEN=200
0036        CALL FFORM(6HTOADF ,0,INBUF,ILEN,7,LU)
0037        IF (ILEN.LT.0)  GOTO 9000
0038        WRITE (6,10) (INBUF(I),I=1,ILEN)
0039        STOP 0
0049  C
0041   9000 WRITE (6,9001) IERR
0042   9001 FORMAT(" IERR RETURNED WAS: ",I6)
0043        CALL FFORM(0,0,0,0,1,LU)
0044        STOP 1
0045        END
```

Figure 6

Conclusion

A programmer can now create a data entry form directly on the 264x terminal and use F2645 to store it to a file. The subroutine FFORM can then be used to display the form on a 264x and read the data from the 264x. It is left to the reader as an exercise to modify F2645 and FFORM to run on a 2640B and/or to work over a multipoint line.

PERFORMANCE STUDY FOR DATACAP 1000

Ben Heilbronn
Steven Richard/HP Data Systems Division

INTRODUCTION TO DATACAP/1000

Briefly, DATACAP/1000 is Hewlett-Packard's application software tool designed to use with HP1000 computers, RTE-IVB operating system, HP3075 series data capture terminals and IMAGE/1000 data base management software. DATACAP/1000 allows dramatic increases in programmer productivity in the development of factory data collection applications. Most aspects of application development can be handled without source code programming, but if necessary, user subroutines written in FORTRAN can be used to extend the product's built-in capabilities.

For a more detailed explanation of DATACAP/1000, refer to the DATACAP/1000 brochure, 5953-4224 or reference manual (92080-90001).

The major objective of this brief is to define three parameters of DATACAP/1000 performance: response time, throughput, and CPU utilization. Hewlett-Packard does not intend to represent a customer's application in these pages. Instead, we are making a modest attempt to define what DATACAP/1000 can do in a "typical" environment.

MEASURING PERFORMANCE

The measurements presented here are produced with the hardware configuration recommended in the DATACAP/1000 Configuration Guide (92080-90003) in conjunction with a terminal simulator. While no guarantee is made, these figures are conservative in their statement of the relationship of the parameters involved.

It should be noted that these measurements were made using the "Production" Transaction Monitor Program (TMP) rather than the "Development" TMP (TMPD). TMPD uses an inter-module communications method that places a heavy burden on the CPU. For this reason, the use of TMPD is recommended for low volume and testing purposes only.

SYSTEM CONFIGURATION

The operating system was an RTE-IVB, generated with the Multi-Terminal Monitor. DATACAP/1000 was installed on an HP 1000 system utilizing an E-series processor and one megabyte of high- performance fault control memory. Peripherals included an HP 2645A console, HP 7920A disc drive, HP 7970B mag tape drive, and two HP 12790A Multipoint Terminal Interfaces. The operating system was generated to accept up to 56 HP 3075 series data capture terminals. Memory was divided into the partitions recommended in the DATACAP Configuration Guide to allow concurrent residence of all DATACAP modules in the physical memory available. Although one megabyte of memory was avalable, only the required amount for the terminal count under test was utilized.

THE TERMINAL SIMULATOR

Performance testing was accomplished using SIMUL, a set of software and hardware which can be programmed to simulate inputs on an HP 3075/6 terminal running under control of the Mulitpoint Terminal Interface. Representation of the physical connections is shown in Figure 1.

SIMUL emulates not only the terminals involved but also the terminal operator's behavior in terms of "think time", the interval between transactions, and a script of answers that are to be provided to the questions asked by DATACAP/1000. Each transaction is defined with the aid of the SIMUL Transaction Generator Program (TRANS) in terms of sequence number, think time, answer text, and input method (keyboard at 10cps typing speed or card/badge reader at 600ms per entry).

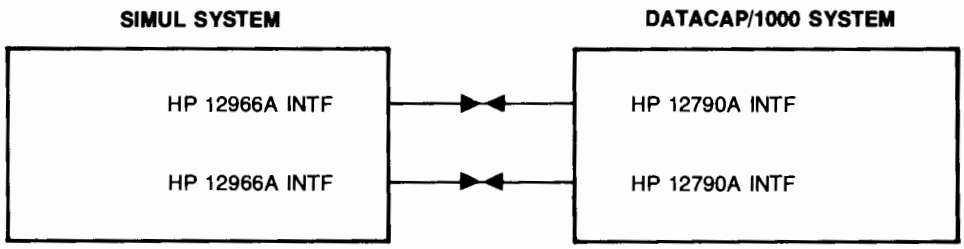


Figure 1.

Each time a SIMUL "terminal" is polled by the system under test, SIMUL responds with one answer from the set of entries pre-defined in the transaction, starting with entry number one and proceeding through the script until the last entry has been transmitted. After transmission of the last entry, SIMUL begins the same transaction over again. Therefore, the SIMUL system can be used to simulate a network of data capture terminals using the following procedures:

1. Interactively design appropriate transactions and responses using DATACAP/1000's TGP and SIMUL's TRANS programs taking the necessary transaction intervals and user think times into account.
2. Interactively define the configuration of data capture terminals, interface lines, and transactions to be run on each line using DATACAP/1000's TMPGN and SIMUL's CONF programs.
3. Run the simulation with that particular configuration.

Data from the simulation run is collected from two sources:

- SIMUL reports the average response time at the answer level and the number of times the answer has been given.
- A program in the DATACAP/1000 host system is used to determine the CPU utilization during the simulation run.

THE SCENARIOS

It was appropriate to measure DATACAP/1000 under a wide variety of application conditions in order to allow this brief to help customers see how their actual application might behave under DATACAP/1000 control. To that end, testing was done under the following sets of conditions:

- One basic transaction was used which included entry of five different data items as listed below:

EMPLOYEE IDENTIFICATION BADGE	(TEN ASCII CHARACTERS)
WORK ORDER NUMBER	(TEN ASCII CHARACTERS)
PART NUMBER	(TEN ASCII CHARACTERS)
QUANTITY COMPLETED	(INTEGER VALUE)
HOURS WORKED	(REAL NUMBER)

- This basic transaction was implemented with three different levels of system interaction as shown in Figure 2.

DATACAP/1000 FEATURE	TRANSACTION #1	TRANSACTION #2	TRANSACTION #3
SYSTEM PROVIDED DATA	YES	YES	YES
ASCII STRING MASK	YES	YES	YES
INTEGER VALUE RANGE	YES	YES	YES
REAL NUMBER RANGE	YES	YES	YES
DISC FILE STORAGE	YES	NO	YES
DATA BASE VALIDATION[1]	NO	YES	YES
DATA BASE UPDATE[1]	NO	NO	YES
DATA BASE ADD[1]	NO	NO	YES
USER VALIDATION	NO	NO	YES
MAG TAPE LOGGING	NO	YES	YES

[1]See Appendix I for details of the data base structure.

Figure 2.

Transaction #1 passed all the data collected plus the system provided information (transaction number, terminal number, time of day, and date) to a disc file using masks and range checking only.

Transaction #2 performed the same mask and range checking validations but also validated the work order against a master data set and used mag tape logging as its only means of storage.

Transaction #3 combined the functions of transactions 1 and 2 but also added an update against the work order master and added one record to both of the detail data sets. Transaction 3 also included a user written validation subroutine.

- Each of the three transaction implementations was tested on the terminal configurations shown below:

Terminal Count	Multipoint Interfaces
8	1
16	1
32	1 and 2
48	2
56	1 and 2

Figure 3

- The user think time for each answer was held at a constant two seconds with the exception of the first question which "selected" the desired transaction. The think time for this answer was varied to act as the independent variable. This simulated a level distribution of transactions occuring as individuals used the terminals with the regularity specified by this "interval time".

RESPONSE TIME

It should be noted that only the AVERAGE response time has been considered in this study. This average is made up of the responses to seven different events in the transaction:

1. transaction selection
2. question 1
3. question 2
4. question 3
5. question 4
6. question 5
7. transaction completion

The response time after the "Transaction Complete" special function key has been pressed may be about twice as long as the average, with the remaining responses being somewhat quicker, thus bringing the average down. Also, if no other terminal has already selected that transaction, it may take about twice the average response time to bring a transaction into activity.

CPU UTILIZATION

The second dependent variable considered by this study was CPU utilization. The percent of CPU time used by DATACAP/1000 is important in two ways. First, the system should be run at some fraction of fully loaded (we have regarded fully loaded as 80%) to allow for variations from the planned 'level' load. In practice, these variations will occur because transactions will tend to bunch around some event in the day whether it be coffee break or check-out time. Second, the use of any other subsystem, for instance DS or RJE, concurrently with DATACAP will have an approximately additive effect on CPU utilization, and, depending on the relative priorities, can have a detrimental effect on response time and throughput. Reserve CPU capacity or careful scheduling should allow for concurrent subsystem operation with a minimal impact on the data capture terminal user.

THROUGHPUT

Throughput is a useful independent variable to use in a study of CPU utilization and response time. Throughput is calculated by using the number of complete transactions that occur. The transaction used in this study was comprised of the seven events referenced in the previous discussion of Response Time. A complete transaction is the sum of several time intervals associated with each event. These time intervals and their sequence in the simulation are described below as Figures 4a and 4b.

TIME FACTORS

1. TRANSACTION SELECTION	$T_{INITIAL_1} + T_{INPUT_1} + T_{RESPONSE_1}$
2. QUESTION #1	$T_{ACTION_2} + T_{INPUT_2} + T_{RESPONSE_2}$
3. QUESTION #2	$T_{ACTION_3} + T_{INPUT_3} + T_{RESPONSE_3}$
4. QUESTION #3	$T_{ACTION_4} + T_{INPUT_4} + T_{RESPONSE_4}$
5. QUESTION #4	$T_{ACTION_5} + T_{INPUT_5} + T_{RESPONSE_5}$
6. QUESTION #5	$T_{ACTION_6} + T_{INPUT_6} + T_{RESPONSE_6}$
7. TRANSACTION COMPLETION	$T_{ACTION_7} + T_{INPUT_7} + T_{RESPONSE_7}$

$T_{INITIAL_1} = T_{ACTION_1}$ = TIME INTERVAL BETWEEN TRANSACTIONS

T_{ACTION_n} = USER THINK TIME

T_{INPUT_n} = KEYBOARD OR CARD/BADGE READER DATA ENTRY TIME

$T_{RESPONSE_n}$ = TERMINAL RESPONSE TIME

Figure 4A.

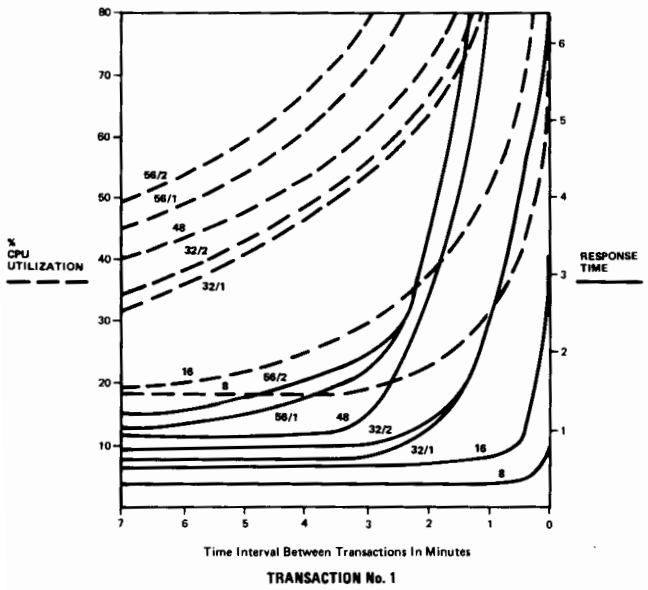
As the bands for both CPU utilization and response time show, there are some factors which offer slight deviations within the basic shapes. For instance, spreading the terminals between two interfaces causes a small increase in overhead, but has the advantage of a slight reduction in response time. In the case of the curves which describe response time, the vertical width of the band is due to the terminal count.

To cite two specific examples, the response time for eight terminals running transaction 1 every two minutes was 0.35 seconds, whereas the response time for 16 terminals running the same transaction every four minutes (the same system throughput) was 0.49 seconds.

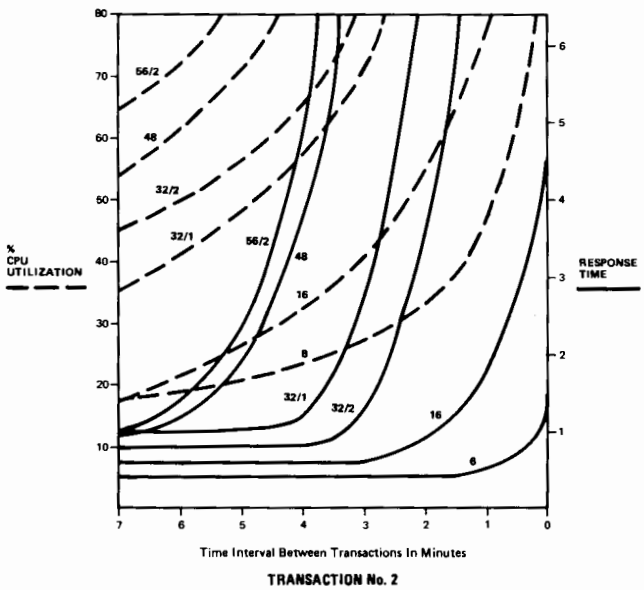
Comparing the results with one and two multipoint interfaces and a system throughput of 490 transactions per hour for transaction 2, showed a response time of 0.90 seconds at 58.8% CPU utilizations for one interface (point A on Figure 5-2) and a response time of 0.68 seconds at 63.2% CPU utilization (point B) for two interfaces. Both tests utilized 32 terminals.

An actual customer case was tested which required 360 transactions/hour, and compared with the graphical data to see where it fit in. The transaction had a combination of attributes from transactions one and two, but with 2.5 instead of 5 entries per transaction. The expected response time indicated graphically on Figure 5-2 lies around .75 seconds with 48.55% CPU usage. The actual tests showed respective figures of .81 seconds and 44.3% CPU usage. Attributing the lower CPU figures to a shorter transaction, these figures are very much as expected, and consistent with the data obtained.

Looking now at Figure 6, which represents the same experimental points plot with the interval between transactions as the independent variable, we see distinct CPU utilization and response time functions for each of the terminal counts tested. Once again, a knee in each response time curve can be seen where the CPU utilization curve reaches 80%.



TRANSACTION No. 1



TRANSACTION No. 2

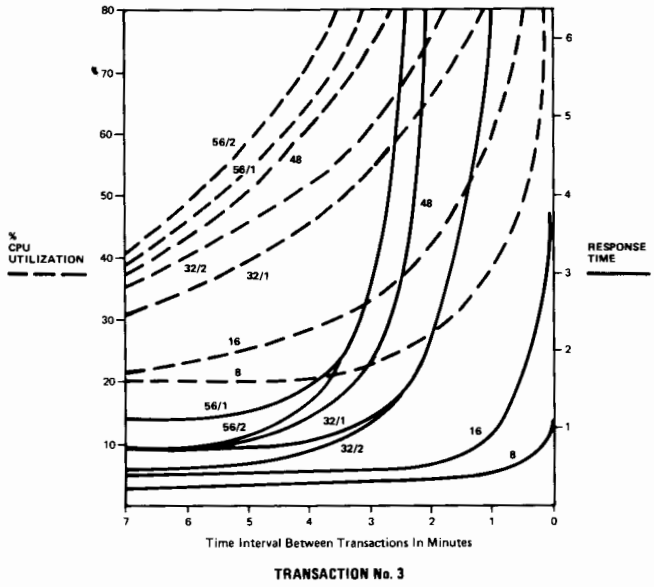
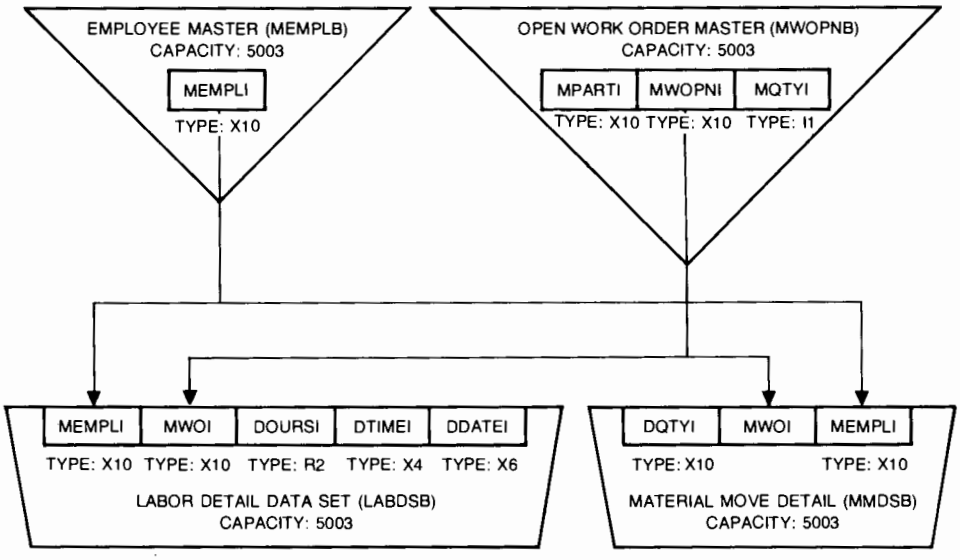


Figure 6

CONCLUSION

This testing has definitely shown that DATACAP provides a viable performance capability other than for the most stringent time and attendance applications which can be supported with very simple custom programming.

BN2DB DATA BASE



Appendix

NEW SOFTWARE PRODUCT CATALOG TO REPLACE SOFTWARE NUMBERING CATALOG

Bill Bohler/HP Data Systems Division

A new and expanded version of the Software Numbering Catalog (SNC) will be making its appearance when the first of the year rolls around. Appropriately named the Software Product Catalog (I told you it had been expanded), it will replace the old SNC as the standard reference to part numbers, media, and manuals for DSD software products. The biggest advantage to the new catalog is that it incorporates information previously scattered throughout various product documentation into one *file* which is shipped with the software on the same distribution medium. Distributing the product catalog as a file on the product distribution medium is just one new feature, however.

The new product catalog contains all the information the old numbering catalog contained and may be employed in any applications where the old catalog saw usage. In addition, the new catalog contains a product manual listing, a media availability listing, and a section containing listings of relocatable and library files, expanded to list each entry point. Refer to figure 2 for a look at a sample Software Product Catalog.

The available media listing contains a description and part number for each medium on which the product is available. These part numbers take the form nnnnn-13xxx where the left hand field constitutes the product part number such as 92068A for RTE-IVB and the right hand field the particular medium number. For example, if product #12345 requires two 7902 flexible discs, they might be numbered 12345-13401 and 12345-13402. More on part numbering schemes later.

The first section of the catalog is titled "MANUALS" and contains a list of the product manuals. Each entry in this section consists of the manual part number, manual title, print date, change number, and revision date code. The manuals are listed in order of ascending part numbers. The print date reflects the date the current edition of the manual was printed. The change number indicates the latest change issued against the current edition of the manual and the revision date code indicates the revision level of the manual as of its latest revision or change notice. Note that the revision date code of the manual reflects the revision level of the manual only and does not in itself imply anything about the revision of the software.

To determine what revision level software is described in a manual, simply refer to the catalog and cross-reference the revision levels of the manual and the revision levels of the software discussed in that catalog. Once the new catalog is in use, the user will be able to determine manual/software compatibility even for outdated software and manuals simply by refering to backdated catalogs. For this reason, it is suggested that users retain backdated copies of the Software Product Catalog.

Manual/software compatibility is also noted by means of the "Manual/Software Compatibility Notice" printed in each manual. Previously, manuals "tracked" software by revision date code, i.e. a REV 1940 manual described REV 1940 software. This required issuing a manual change notice against a manual every time its associated software changed, even if no change occurred in the manual itself. The reason for the change notice was simply to update the revision date code of the manual to agree with that of the software. Now, however, a change in software will cause the issuance of a revised Software Product Catalog but no manual change notice will be issued unless actual changes to the manuals occur.

Regardless of any software considerations, an up-to-date manual consists of the current edition of the manual and the last change notice issued against it. Change notices are cumulative, i.e., the latest change notice incorporates all previous change notices. When a manual is revised, all change notices will be integrated into the manual, the change number cleared, and new print and revision dates established.

Section 2 of the catalog is titled "SOFTWARE MODULES" and contains the same basic information as the old Software Numbering Catalog. Additionally, the files are cross referenced by media and revision date codes are included. Each entry consists of the file name, part number, revision date code, a short description, and media part number if applicable. Thus, if product #12345 resides on discs with media part numbers 12345-13401 and 12345-13402 and contains file %WATME on the first disc and file %WORRY on the second, the entry for %WATME will include media part number 12345-13401 and the entry for %WORRY will contain 12345-13402. Files are listed by type (relocatable,library, etc.) and alphabetically within each type. The following list indicates some generally adhered to file naming conventions:

&xxxxx — source program files

*xxxxx — command files, answer files, file manager transfer files

"xxxxx — ASCII information files, help files

%xxxxx — non-library relocatables

$xxxxx — searchable, relocatable libraries

!xxxxx — absolute binary, bootable, executable files

Section 3 is the "SOFTWARE SUBMODULES" section. In this section, each relocatable and library file that contains sub-modules is expanded and each entry point listed. The file names are listed in alphabetical order. Each sub-module entry consists of the entry point name, part number, revision date code, and an optional description. The entry point names are listed in the order they appear in the file.

The naming convention for the file containing the Software Product Catalog is the product number with its suffix letter transposed to be a prefix. For example, for product 92070A, the file is named A92070, for 12345W, W12345. The location of the file on the distribution medium is medium dependent. When the distribution medium is flexible or hard disc, the location of the catalog file is unimportant since the discs have directories and the file may be accessed by name. On mini-cassettes, the file is the last file on the first cassette.

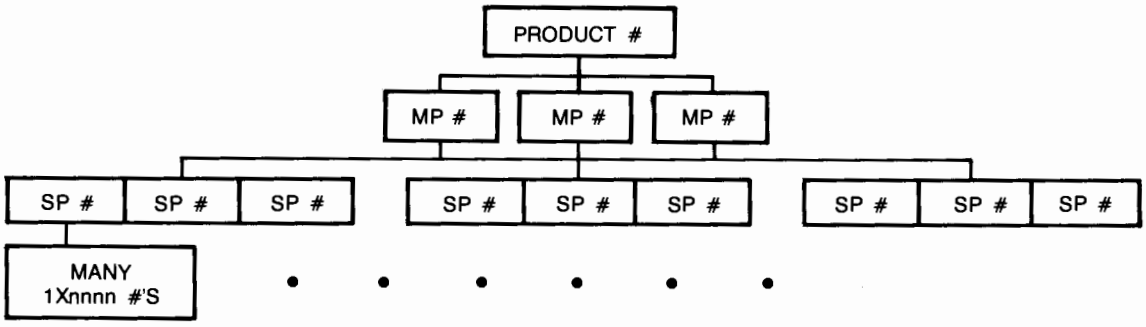
Part numbering is arranged such that several levels of part numbers are formed. Figure 1 describes the relation of these levels and some of the numbering conventions used are listed below. The first two digits of the right hand field of the service part number supply the following information:

xxxxx-1Xnnn — indicates sub-file contained in a file

xxxxx-12nnn — indicates file with sub-files

xxxxx-16nnn — indicates single binary relocatable file

xxxxx-18nnn — indicates source file



PRODUCT # — (nnnnn-xxx where xxx = media option #)
MP # — media part number (nnnnn-13xxx #'s)
SP # — service part number (nnnnn-12,16,18nnn #'s)

Figure 1.

As described in the media availability section, each disc or tape on which a product is distributed is assigned a media part number of the form nnnnn-13xxx. Each file is assigned a service part number. Since a software product may require several discs/tapes, and since each disc/tape is assigned a media part number, each product may contain several media part numbers. In turn, each disc/tape will contain several files, and therefore, each media part number may have several service part numbers associated with it. Finally, if a service part number indicates a file containing sub-files, that file will contain many individual modules and thus the service part number may have many 1Xnnnn or sub-assembly part numbers associated with it. Below is a sample Software Product Catalog.

```
*********************************************************************
*                                                                   *
*   SOFTWARE PRODUCT CATALOG          11:10 AM  MON., 10  SEPT., 1979   *
* 92070A-041         RTE SYSTEM SOFTWARE ON 7902 FLEXIBLE DISC      *
*                                                                   *
*********************************************************************
*                                                                   *
* THIS FILE PROVIDES A LIST OF THE SOFTWARE MODULES AND MANUALS     *
* THAT COMPRISE THE PRODUCT. THE LIST IS ORGANIZED INTO THREE       *
* PARTS AS FOLLOWS:                                                 *
*                                                                   *
*   A.  MANUALS - THE ENTRY FOR EACH MANUAL INCLUDES THE TITLE      *
*       OF THE MANUAL, MANUAL PART NUMBER, PRINT DATE, AND IF       *
*       APPLICABLE, THE LATEST MANUAL CHANGE NUMBER AND DATE.       *
*                                                                   *
*   B.  SOFTWARE MODULES - EACH SERVICEABLE SOFTWARE MODULE IS      *
*       LISTED IN ALPHABETICAL ORDER WITHIN FILE TYPES.             *
*                                                                   *
*   C.  SUBFILES - SUBFILES TO SOFTWARE MODULES, WHERE APPLICABLE.  *
*       INCLUDES CONTENTS OF LIBRARIES, SUBROUTINE NAMES, AND ANY   *
*       FURTHER BREAKDOWN OF THE SOFTWARE MODULE THAT MAY BE        *
*       USEFUL TO IDENTIFY A PARTICULAR PIECE OF SOFTWARE.          *
*                                                                   *
*                                                                   *
* THIS PRODUCT IS DELIVERABLE ON THE FOLLOWING MEDIA; THE SPECIFIC  *
* MEDIUM IS SPECIFIED AT TIME OF ORDER OF THE PRODUCT.              *
*                                                                   *
*      92070-13400          7902 FLEXIBLE DISC                     *
*      92070-1350X          800 BPI MAG TAPE                       *
*      92070-1350Y          1600 BPI MAG TAPE                      *
*                                                                   *
*                                                                   *
* SECTION 1: MANUALS                                                *
*                                                                   *
*   PART          DESCRIPTION                    PRINT  CHANGE  DATE  *
*   NUMBER                                       DATE     #     CODE  *
*   -----------  ------------------------------ -----  ------  ----  *
*   92070-90000  RTE GENERAL INFORMATION MAN.    1/80           2001  *
*   92070-90002  RTE OPERATOR'S GUIDE            1/80           2001  *
*   59310-90064  HP-IB USER'S MANUAL             1/80           2001  *
*   92067-90003  RTE IV ASSEMBLER REFER. MAN.    4/79           2001  *
*                                                                   *
*                                                                   *
* SECTION 2: SOFTWARE MODULES                                       *
*                                                                   *
*                                                                   *
* FILE    SERVICE       REV                                         *
* NAME    PART          CODE  DESCRIPTION                 MEDIA     *
* ------  --------      ----  -------------------------   ----------- *
(ASC) GENERAL ASCII FILES

(REL) RELOCATABLE BINARY FILES

  %AUTOR  92070-16252   1941  AUTO RESTART PROGRAM
  %CLASS  92070-16093   1941  CLASS I/O
                .
                .
                .
  %XREF   92070-16012   1941  ASSEMBLER CROSS REF.

(LIB) SUBROUTINE LIBRARY FILES

  $CMDLB  92070-12004   1941  OP. CMD ACTION LIB.
  $FMP    92070-12003   1941  FILE MGR. LIBRARY
```

```
*                                                                    *
*                                                                    *
* SECTION 3: SOFTWARE SUBMODULES                                     *
*                                                                    *
*                                                                    *
* FILE    SERVICE       REV                                          *
* NAME    PART          CODE  DESCRIPTION             MEDIA          *
* ------  -------       ----  ----------------------  -------------  *

(REL) RELOCATABLE BINARY FILES

  %AUTOR
   AUTOR   92070-16252   1941
  %CLASS
   CLASS   92070-16093   1941
     .
     .
     .
  %TIME
   TIME    92070-16139   1941
  %XREF
   XREF    92067-16012   1913

(LIB) SUBROUTINE LIBRARY FILES

  $CMDLB
   BL..    92070-1X098   1941
   CN..    92070-1X099   1941
    .
    .
    .
   SY..    92070-1X034   1941
   CNOPT   92070-1X277   1941
  $FMP
   FMP     92070-12003   1941
   $BMON   92070-1X244   1941
   APOSN   92070-1X038   1941
   CRETS   92070-1X041   1941
   CREAT   92070-1X040   1941
     .
     .
     .
   NAMR    92070-1X263   1941
   INAMR   92070-1X264   1941
```

Figure 2. Sample Software Product Guide

NOTE: This catalog has been edited to comply with space restrictions. It is intended as a sample only and information contained herein is not guaranteed correct.

JOIN AN HP 1000 USER GROUP!

Ever wonder how other HP users have used the HP 1000 in application areas similar to your own? Have a special program or driver you'd like to share with other users? Interested in hearing about new developments in HP 1000 hardware and software?

If your answer to any one of these questions is YES, then an HP 1000 user group might be just the thing for you. These and other similar activities are carried on regularly as part of the function of the many HP 1000 user groups that exist around the world. There's a good chance that there's a user group right near your location! To get in on the action, just check the list below, and contact the group nearest you to find out when and where the next meeting will be held.

Or, if there isn't a group near you, why not start one? The Communicator/1000 can help you out by announcing the creation of new groups. Just send a letter — c/o Editor, HP 1000 Communicator, — with the name of your new group and the means by which other users can join. We'll add your group to our list and publish it in the next issue of the Communicator.

Here are the groups that we know of as of October, 1979. (If your group is missing, send the Communicator/1000 editor all of the appropriate information, and we'll update our list.)

NORTH AMERICAN HP 1000 USER GROUPS

Area	User Group Contact
Boston	LEXUS P.O. Box 1000 Norwood, Mass. 02062
Chicago	Dave Olson Institute of Gas Technology 1846 W. Eddy Street Chicago, Illinois 60657
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

NORTH AMERICAN HP 1000 USER GROUPS (CONTINUED)

Area	User Group Contact
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
Washington/Baltimore	Paul Toltavull Hewlett-Packard Co. 2 Choke Cherry Rd. Rockville, MD. 20850
General Electric Co. (GE employees only)	Stu Troupe 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, Wokingham Berkshire, RG11 5AR England (734) 784 774
Amsterdam	Mr. Van Puten Institute of Public Health Bilthoven Anthony Van Leeuwenhoeklaan 9 The Netherlands
South Africa	Andrew Penny Hewlett-Packard South Africa Pty. private bag Wendywood Sandton, 2144 South Africa

INTERNATIONAL CUSTOMER TRAINING SCHEDULES 79/80

	AUSTRIA	AUSTRALIA	BELGIUM	ENGLAND	FINLAND	FRANCE	GERMANY	ITALY	JAPAN	NETHERLANDS	SPAIN	SWEDEN	SWITZERLAND
22941A 21MX/XE Maint. 5 days/$500						Oct 22 (G)							
22943A 7970B/E Maint. 5 days/$500													
22945A 7905/06 Maint. 5 days/$500						Oct 29 (G)							
22951B Intro to HP 1000 4 days/$400	Sep 03			Sept 19 (W)			Sept 24	Sept 10 (R) Oct 22 (M)		Aug 27 Nov 19 Feb 11 May 05		Oct 08	
22951B-H01 FORTRAN IV 5 days							Oct 08				Oct 15		
22952B 1000 ASMB 5 days/$500	Sep 24	Oct 08 (P) Nov 12 (B)	Sept 10	Oct 29 (W)	Nov 11	Oct 08 (O)	Sep 03	Oct 15 (R) Dec 17 (M)	Jun 11 (T)	Dec 17 Apr 21	Oct 22	Sep 10 Nov 12	
22961B DS/1000 Theory of Operation 4 days/$500							Oct 01			Jan 28			
22962B DS/1000 to HP 3000 Theory of Operation 1 day/$100							Oct 05			Sep 28			
[redacted]													
[redacted]													
[redacted]													
22980C HP-IB Interface With HP 1000 4 days/$400										Aug 20 Feb 04			
22983B 21MX/E Microprogramming 5 days/$500						Oct 01 (O)							
22984A 7920 Maint. 5 days/$5000													
[redacted]													
22987A DS/1000 User's Course 5 days/$500		Oct 22 (P)		Oct 22 (W)			Sep 17			Nov 05 Jun 23			
22990A RTE Driver Writing 3 days/$300		Aug 06 (B) Oct 01 (P) Nov 19 (B)								Oct 01 Dec 10 Apr 28			
[redacted]													

■ Mature Product Courses

INTERNATIONAL CUSTOMER TRAINING SCHEDULES 79/80 Continued

	AUSTRIA	AUSTRALIA	BELGIUM	ENGLAND	FINLAND	FRANCE	GERMANY	ITALY	JAPAN	NETHERLANDS	SPAIN	SWEDEN	SWITZERLAND
22992A HP 1000 Memory RTE 10 days/$1000							Sept 10						
22993A IMAGE 5 days/$500													
22994A Session Monitor User 10 days/$1000					Sept 23 Nov 25								
22995A System Manager 5 days/$500													
22996A RTE-IVA-IVB Upgrade 2 days/$325													
22997A Advanced RTE 5 days/$800													
40270A Intro to HP Computers 5 days						Nov 05 (O)							
91302A 2645 Maint. 3 days/$300													

■ Mature Product Courses

INTERNATIONAL TRAINING CENTER ADDRESSES

AUSTRIA
(Vienna)
Handelskai 52
Postfach 7
A 1205 Wien
Tel: (0222) 35 16 21-32
Telex: 75923
Cable: Hewpack Wien

AUSTRALIA
(Blackburn) B
CUSTOMER TRAINING CENTER
31-41 Joseph Street
Blackburn, Victoria, Australia
(Pymble) P
CUSTOMER TRAINING CENTER
31 Bridge Street
Pymble, New South Wales, Australia

BELGIUM
(Brussels)
Avenue du Col Vert, 1
Groenkraaglaan
B-1170
Brussels, Belgium
Tel: (02) 672 22 40

ENGLAND
(Altrincham) A
Navigation Road
Altrincham
Cheshire WA14 1NU
(Winnersh) W
King Street Lane
Winnersh, Workingham
Berkshire RG11 5 AR
Tel: Workingham 784774
Cable: Hewpie London
Telex: 8471789

FINLAND
(Helsinki)
Nahkahousuntie 5
00211 Helsinki 21
Tel: 90-692 30 31

FRANCE
(Grenoble) G
5, avenue Raymond-Chanas
38320 Eybens
Tel: (76) 25-81-41
Telex: 980124
(Orsay) O
Quartier de Courtaboeuf
Boite Postale No. 6
F-91401-Orsay
Tel: (01) 907 7825

GERMANY
(Boeblingen)
Kundenschulung
Herrenbergerstrasse 110
D-7030 Boeblingen, Wurttemberg
Tel: (07031) 667-1
Telex: 07265739
Cable: HEPAG

ITALY
(Milan)
Via G. Di Vittorio, 9
Tel: 02-903691
20063 Cernusco S/N (MI)

JAPAN
(Osaka) O
Chuo Building
5-4-20 Nishinakajima
Yodogawa-Ku, Osaki-shi
Osaka, 532 Japan
Tel: 06-304-6021
Telex: 523-3624 YHP OSA
(Tokyo) T
2205 Takaido Higashi 3-chome
Suginami-Ku, Tokyo 168
Tel: 03-33-8111
Telex: 232-2024 YHP MARKET TOK

NETHERLANDS
(Amsterdam)
Van Heuven Goedhartlaan 121
Amstelveen 1134
Netherlands
Tel: 020 472021

SPAIN
(Madrid)
Jerez No. 3
E-Madrid 16
Tel: (1) 458 26 00
Telex: 23515 hpe

SWEDEN
(Stockholm)
Enighetsvagen 1-3, Fack
S-161 20 Bromma 20
Tel: (08) 730 05 50
Cable: MEASUREMENTS
Telex: 10721

SWITZERLAND
(Zurich)
19 Chemin Chateau — Bloc
1219 Le Lignon — Geneve
Tel: 022/96 03 22

For course prerequisites and registration information contact one of the HP training centers listed above.

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

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

Printed in U.S.A. 11/79

Part No. 5951-6111