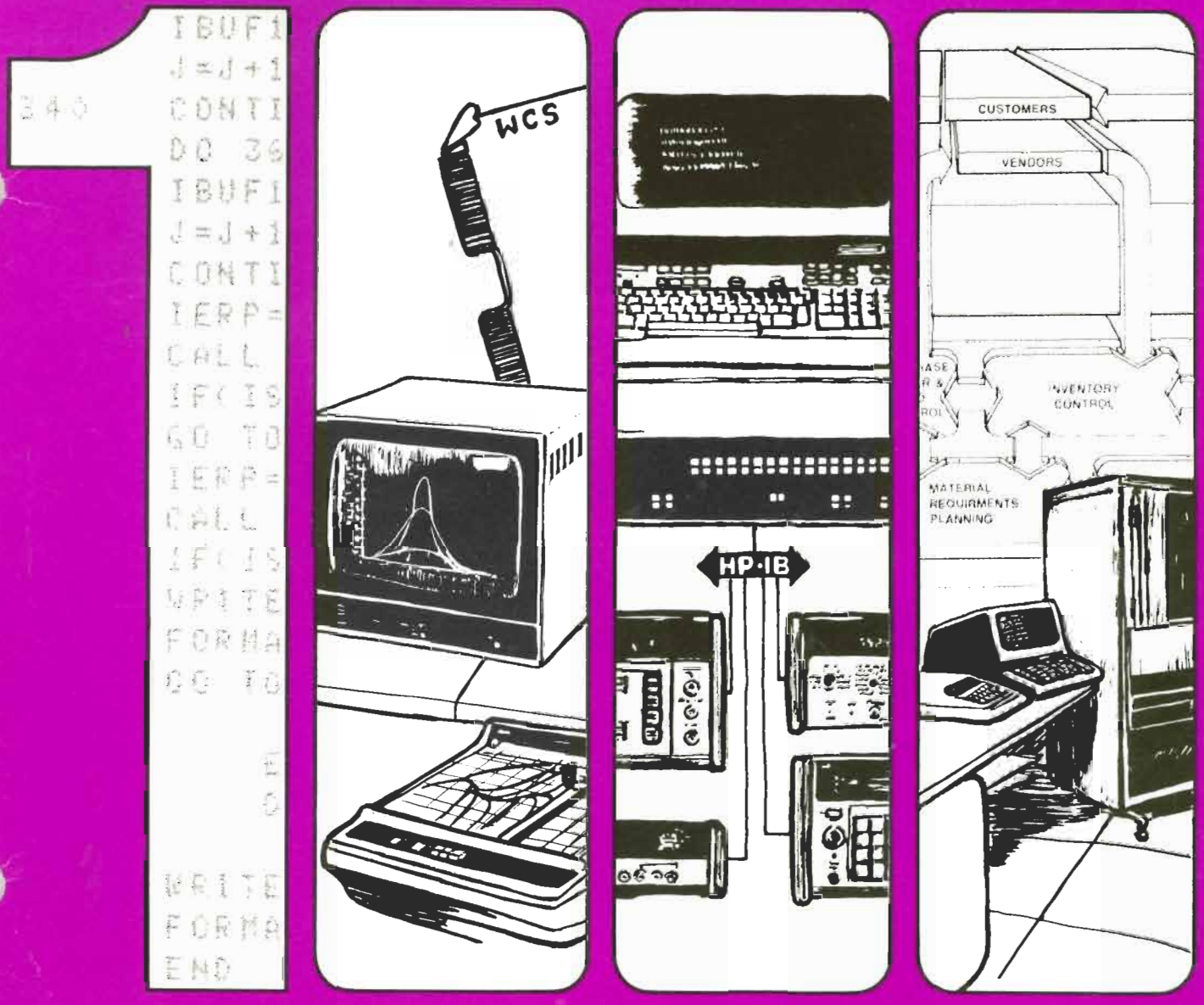
HEWLETT PACKARD

Hewlett-Packard Computer Systems

COMMUNICATOR



1980

Volume IV Issue 6

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

Received from D. Underwood

HEWLETT-PACKARD
COMPUTER SYSTEMS

Volume IV
Issue 6

COMMUNICATOR/1000

Feature Articles

Departments

EDITOR'S DESK

ABOUT THIS ISSUE

This issue of the Communicator brings us to the end of 1980, and the end of Volume IV. I hope that all of our readers have found the Communicator to be helpful and informative during the last year, and will also find the same qualities in this issue.

Issue 6 brings together four superlative feature articles dealing with Data Communications, Operations Management, and Operating Systems.

The set is opened with an article by Peter D. Cox of Computer & Engineering Services in Brisbane Australia. Peter's article, entitled "Scheduling Programs From Type 6 Files", gives an excellent description of some routines provided in RTE systems to programmatically RP ,"clone", and release ID segments for type 6 files. I am certain that Peter's article, and the program that he has included can be used by many of our readers in a variety of applications. I am also certain that creative programmers will be able to find numerous additional applications for the routines that Peter has described.

The second feature article presented in issue 6 is in the Data Communications category. Lyle Weiman, in Data Systems Division's software lab has contributed an article that I am sure will be of interest to any of our readers that have purchased the new DS/1000-IV product. This new distributed systems offering from HP includes a very powerful feature which allows users to effectively "log-on" to a remote system. Lyle has written, and contributed to the User's Group, two programs that the user with absolutely no DS know-how can run, to quickly and easily log on to a remote computer. Lyle's programs present an efficient and friendly way to create "Remote Interactive Session Access".

The third feature article in issue 6 is the continuation of Robert Niland's Links/1000 series that has appeared in the last several Communicators. In this issue, Bob explains the code that the Fortran compiler generates to call subroutines and functions. Bob goes on to describe some common pitfalls that programmers can encounter when calling Fortran subroutines or functions, and also discusses what to watch out for when passing parameters between them.

The final feature article in this issue is from Joel Dubois in HP's Grenoble Division. His article, "How to Read IBM Flexible Discs on an RTE-L", provides the reader with the tricks that he will need to know in order to use his L-Series system to read flexible discs that have been prepared on an IBM system. Joel's contribution adds a new dimension to the Communicator. His is the first feature article dealing with HP's L-Series computer that has ever been published in the Communicator.

As a token of our appreciation for a job well done, an HP 32-E calculator is awarded to the best article in each of the three categories, customers, HP field personnel, and employees of any HP division. For more information about the calculator awards, see the article on the following page entitled, "Become a Published Author in the Communicator/1000". For more information on the calculator awards made for this issue, keep reading!

Best Feature Article From a Customer	**Scheduling Programs From Type 6 Files** Peter D. Cox/Computer&Engineering Services
Best Feature Article From HP Division Personnel	**Remote Interactive Session Access** Lyle Weiman/HP Data Systems Division

As always, many thanks to all of the authors that spent so much time, and expended so much effort, to share with us what they have learned about, or the software that they have developed. We all appreciate your generosity.

All of us here at Data Systems Division are hoping that we get the same outstanding support from our readers in 1981 that we got in 1980. After all, without the help of our readers, the Communicator would just be 50 blank pages. We hope to hear more from those of you that have sent letters and articles to us in the past, and we hope to also hear from a large number of you that we have never heard from before.

Sincerely,

The Editor

EDITOR'S DESK

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

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

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

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

WIN AN HP-32E CALCULATOR!

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

1. HP 1000 Customers;

2. HP field employees;

3. HP division employees 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

EDITOR'S DESK

LETTER TO THE EDITOR:

Dear Editor,

I enjoyed the article by David Liu in Volume 4 Issue 3 of the Communicator/1000 ["Put Your 264X Terminal Display on Paper"]. Unfortunately, either something is missing from the article or was contained in a previous issue of the Communicator. Specifically, the subroutine SMOVE which is referred to on page 17 is the one that I can not find. Perhaps you would be kind enough to point me in the right direction to find this subroutine so I can try out Mr. Liu's program.

Sincerely,

Robert L. Miller
Associate Professor
Michigan Molecular Institute

Dear Mr. Miller,

The SMOVE routine that you are referring to is included in what we call the Character String Library, %DECAR. This library module is included with the RTE-IVB Grandfather software.

It sounds like your system does not have %DECAR generated in as a library, and therefore, you will have to first get the module on some disc cartridge, and then have the LOADR "SE,%DECAR" to satisfy the external SMOVE. Good luck.

The Editor

RETURNING TO THE MAIN FROM A SEGMENT

C. S. Hopman/South Pacific Commission

[Editor's note: I recently received a letter from Mr. Hopman suggesting some enhancements to information in the RTE-IVB Programmers's Reference Manual. After reading the letter I became convinced that Mr. Hopman had documented a feature of our software that could be very useful to many of our readers. Eventhough he really sent a "Letter to the Editor", it seemed much more appropriate to publish his letter in the "Bit Bucket" after considering its content. Thanks for your letter Mr. Hopman.]

Dear Sirs,

The HP 92068A HP 1000 RTE-IVB Programmer's Reference Manual contains (page 4.3) the following statements: "Chaining of segments is unidirectional... A segment may, in turn call another segment, but a segment written in FORTRAN cannot easily return to the main program"

This information is misleading and may cause users to underestimate seriously the computing power that HP 1000 FORTRAN makes available to them. I would suggest that the paragraph be amended as follows:

"Return from a segment to the main or root program is effected through assigned instructions which may be passed, like other data, through COMMON or parameter passage.

Example:

```
        PROGRAM MAIN(4)
        COMMON INSTN
              .
              .
              .
        ASSIGN 100 TO INSTN
        CALL EXEC (8,6HSEGM01)
C
C   THE NEXT INSTRUCTION, WITH LINE NUMBER 100, IS THE
C   LINE TO WHICH THE SEGMENT WILL RETURN
C
100     CONTINUE
              .
              .
              .
        END

        PROGRAM SEGM01 (5)
        COMMON INSTN
              .
              .
              .
        GO TO INSTN
        END
```

Since it is possible to place several return addresses in COMMON, segments may return to any desired locations in MAIN, and MAIN may call the same segments as often as desired. By making judicious use of segmentation, EMA and workfiles (whose definitions may be shared in COMMON), it is possible to run extremely large and complex programs in relatively small partitions."

A listing of an example has been provided which you may try out. I have used this technique to run in 56K a program which required over 300K running under IBM OS VS1. My version is interactive whereas theirs presumes card input, and all those questions take up lots of room.

Yours sincerely,

C. S. Hopman

```
FTN4
      PROGRAM SDFG(4)
      COMMON INSTN
      DO 100 IX=1,50
      WRITE (1,8230)
8230  FORMAT('HI! THIS IS THE MAIN!')
      ASSIGN 10 TO INSTN
      CALL EXEC (8,6HSDFG1 )
10    WRITE (1,8250)
8250  FORMAT ('SEGM1 DONE!')
      ASSIGN 20 TO INSTN
      CALL EXEC (8,6HSDFG2 )
20    WRITE (1,8270)
8270  FORMAT ('SEGM2 DONE!')
100   CONTINUE
      END


FTN4
      PROGRAM SDFG1(5)
      COMMON INSTN
      WRITE (1,8230)
8230  FORMAT (' HI! THIS IS SEGM1!')
      GO TO INSTN
      END


FTN4
      PROGRAM SDFG2 (5)
      COMMON INSTN
      WRITE (1,8230)
8230  FORMAT ('HI! THIS IS SEGM2!')
      GO TO INSTN
      END
```

THE ZAPME TECHNIQUE FOR PROGRAM DEBUGGING

Larry W. Smith/HP Fullerton Ca.

It never ceases to amaze me how much I learn from simply chatting to HP RTE users. While I was attending the first HP 1000 International User's Conference this past August, I was talking to Robin Brocino who is an HP 1000 system manager and she was commenting on a method she uses to debug assembly language programs. Her comments stunned me because I had never thought of debugging a program quite that way. But, nevertheless, her method is ingenious. Robin remarked, "I just have my programmers, whenever they want to abort or dump out the registers, simply jump to the A register and let the system print out all the information." For example, the following program with the resulting terminal output illustrates the method:

```
                    ASMB,R,L
00000                 NAM ZAPME
00000 102501  START   LIA 1
00001 006404          CLB,INB
00002 105745          LDX D2
00003 000007R
00004 105755          LDY D3
00005 000010R
00006 024000          JMP 0
00007 000002  D2      DEC 2
00010 000003  D3      DEC 3
                      END START
```

Memory Protect Occurs

Resulting CRT Output

```
MP INST =  24000
ABE        0       1 0
XYO        2       3 0
MP     ZAPME 42050
ZAPME ABORTED
```

The memory protect logic of the HP 1000 series computers will not allow a branch to a location below the memory protect fence address, which address 0 will always be whenever a program is in execution. The B register (loaction 1) could also be used.

Although this is a clever technique for dumping the working registers of the computer, it is not the intent of this article to suggest nor recommend that it serve as a replacement for DBUGR. Maybe you can add this to your bag of tricks. Oh, for you FORTRAN users, here's the same trick:

```
JUMP=0
   .
   .
   .
GO TO JUMP
```

In closing, if you try this technique and your program does not abort, then it is time to get reacquainted with your HP Customer Engineer or ask your system manager to explain the situation to you.

All in all, I hope that you thought Robin's technique was as clever as I did.

JULIAN CALENDAR
1981

JANUARY

SUNDAY	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY
				1(1)	2(2)	3(3)
4(4)	5(5)	6(6)	7(7)	8(8)	9(9)	10(10)
11(11)	12(12)	13(13)	14(14)	15(15)	16(16)	17(17)
18(18)	19(19)	20(20)	21(21)	22(22)	23(23)	24(24)
25(25)	26(26)	27(27)	28(28)	29(29)	30(30)	31(31)

FEBRUARY

SUNDAY	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY
1(32)	2(33)	3(34)	4(35)	5(36)	6(37)	7(38)
8(39)	9(40)	10(41)	11(42)	12(43)	13(44)	14(45)
15(46)	16(47)	17(48)	18(49)	19(50)	20(51)	21(52)
22(53)	23(54)	24(55)	25(56)	26(57)	27(58)	28(59)

MARCH

SUNDAY	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY
1(60)	2(61)	3(62)	4(63)	5(64)	6(65)	7(66)
8(67)	9(68)	10(69)	11(70)	12(71)	13(72)	14(73)
15(74)	16(75)	17(76)	18(77)	19(78)	20(79)	21(80)
22(81)	23(82)	24(83)	25(84)	26(85)	27(86)	28(87)
29(88)	30(89)	31(90)				

APRIL

SUNDAY	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY
			1(91)	2(92)	3(93)	4(94)
5(95)	6(96)	7(97)	8(98)	9(99)	10(100)	11(101)
12(102)	13(103)	14(104)	15(105)	16(106)	17(107)	18(108)
19(109)	20(110)	21(111)	22(112)	23(113)	24(114)	25(115)
26(116)	27(117)	28(118)	29(119)	30(120)		

MAY

SUNDAY	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY
					1(121)	2(122)
3(123)	4(124)	5(125)	6(126)	7(127)	8(128)	9(129)
10(130)	11(131)	12(132)	13(133)	14(134)	15(135)	16(136)
17(137)	18(138)	19(139)	20(140)	21(141)	22(142)	23(143)
24(144)	25(145)	26(146)	27(147)	28(148)	29(149)	30(150)
31(151)						

JUNE

SUNDAY	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY
	1(152)	2(153)	3(154)	4(155)	5(156)	6(157)
7(158)	8(159)	9(160)	10(161)	11(162)	12(163)	13(164)
14(165)	15(166)	16(167)	17(168)	18(169)	19(170)	20(171)
21(172)	22(173)	23(174)	24(175)	25(176)	26(177)	27(178)
28(179)	29(180)	30(181)				

JULY

SUNDAY	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY
			1(182)	2(183)	3(184)	4(185)
5(186)	6(187)	7(188)	8(189)	9(190)	10(191)	11(192)
12(193)	13(194)	14(195)	15(196)	16(197)	17(198)	18(199)
19(200)	20(201)	21(202)	22(203)	23(204)	24(205)	25(206)
26(207)	27(208)	28(209)	29(210)	30(211)	31(212)	

AUGUST

SUNDAY	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY
						1(213)
2(214)	3(215)	4(216)	5(217)	6(218)	7(219)	8(220)
9(221)	10(222)	11(223)	12(224)	13(225)	14(226)	15(227)
16(228)	17(229)	18(230)	19(231)	20(232)	21(233)	22(234)
23(235)	24(236)	25(237)	26(238)	27(239)	28(240)	29(241)
30(242)	31(243)					

SEPTEMBER

SUNDAY	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY
		1(244)	2(245)	3(246)	4(247)	5(248)
6(249)	7(250)	8(251)	9(252)	10(253)	11(254)	12(255)
13(256)	14(257)	15(258)	16(259)	17(260)	18(261)	19(262)
20(263)	21(264)	22(265)	23(266)	24(267)	25(268)	26(269)
27(270)	28(271)	29(272)	30(273)			

OCTOBER

SUNDAY	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY
				1(274)	2(275)	3(276)
4(277)	5(278)	6(279)	7(280)	8(281)	9(282)	10(283)
11(284)	12(285)	13(286)	14(287)	15(288)	16(289)	17(290)
18(291)	19(292)	20(293)	21(294)	22(295)	23(296)	24(297)
25(298)	26(299)	27(300)	28(301)	29(302)	30(303)	31(304)

NOVEMBER

SUNDAY	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY
1(305)	2(306)	3(307)	4(308)	5(309)	6(310)	7(311)
8(312)	9(313)	10(314)	11(315)	12(316)	13(317)	14(318)
15(319)	16(320)	17(321)	18(322)	19(323)	20(324)	21(325)
22(326)	23(327)	24(328)	25(329)	26(330)	27(331)	28(332)
29(333)	30(334)					

DECEMBER

SUNDAY	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY
		1(335)	2(336)	3(337)	4(338)	5(339)
6(340)	7(341)	8(342)	9(343)	10(344)	11(345)	12(346)
13(347)	14(348)	15(349)	16(350)	17(351)	18(352)	19(353)
20(354)	21(355)	22(356)	23(357)	24(358)	25(359)	26(360)
27(361)	28(362)	29(363)	30(364)	31(365)		

COMMUNICATOR INDEX BY CATEGORY

COMMUNICATOR INDEX BY CATEGORY (Continued)

COMMUNICATOR INDEX BY ISSUE

COMMUNICATOR INDEX BY ISSUE (Continued)

Documentation Enhancement

Charles G. Fugee/Dynalectron Corp.

Prompted by Alan Tibbetts' Bit Bucket article on supressing CRLF (which we use with great success) we suggest the following for source code and documentation enhancements.

For expanded and compressed print:

Under program control write the following octal codes for the associated print mode.

```
33B,23153B,30523B         Expanded
15446B,65460B,51400B      Normal
33B,23153B,31123B         Compressed
```

In the editor, the codes to enter are:

```
ESC & k1S    Expanded
ESC & k0S    Normal
ESC & k2S    Compressed
```

Top of page and bottom of page

Under program control:

```
33B,23154B,30526B         Top of Page
33B,23154B,31126B         Bottom of Page
```

In the editor,

```
ESC & 11V    Top of Page
ESC & 12V    Bottom of Page
```

Underline on and underline off

Under Program control:

```
15446B,62104B       Underline On
15446B,62101B       Underline Off
```

In the editor

```
ESC & dD     Underline On
ESC & dA     Underline Off
```

These codes may be kept in files created by the editor and either merged into your current workfile, or dumped to your printer. Note that these commands work on the HP 263X printers and on the HP 2608 printers but not on the HP 2767, or other actual 'line' printers.

As an example of the power of this type of tool, look what it can do with a mathematical formula in a documentation report.

```
for Pi = 3.141592654
                                                   2
                                              -1/2x
        f ( x )   =          1               e
                      (2 Pi) ** 0.5
```

A FORTRAN demonstration program follows. Be sure to define your printer's logical unit number when it is requested by the program.

Remember that once a print mode is turned on it must be turned off, or it will remain in that mode, i.e., the underline will stay on until turned off.

It's turned on now and will stay that way until I issue the command to the editor to turn it off. Now I'll turn it off. Ah! That's better!

```
FTN4,B,L,F
          PROGRAM ENHNC(3,1000), ENHANCEMENT DEMO PROGRAM, CGF, 9/26/80
C                              C.G.FUGEE, DYNALECTRON CORP., RBD
C
C SEE  WHAT  ENHANCING  DOES?
C
                INTEGER EXPND(3),NORM(3),CMPRS(3),TOP(3),UNDRON(2)
                INTEGER UNDROF(2),PAGE(3),BOT(3)
                DATA EXPND,NORM/33B,23153B,30523B,15446B,65460B,51400B/
                DATA CMPRS,TOP / 33B,23153B,31123B, 33B,23154B,30526B /
                DATA BOT / 33B,23154B,31126B /
                DATA UNDRON,UNDROF / 15446B,62104B, 15446B,62101B /
C
C
C
                      LP=0
 5              WRITE(1,10)
                READ(1,*) LP
                IF(LP.LE.0) GO TO 5
C
C BEGIN  THE  DEMONSTRATION
C
                WRITE(LP,12) EXPND,NORM
                WRITE(LP,13) CMPRS,NORM
                WRITE(LP,14) UNDRON,UNDROF
                WRITE(LP,15) BOT,TOP
C
C FORMAT  STATEMENTS
C
10     FORMAT(5X,"ENTER THE LOGICAL UNIT NUMBER OF YOUR PRINTER _")
12     FORMAT(//5X,3A2,"THIS IS EXPANDED"3A2,1X,"    THIS IS NORMAL ")
13     FORMAT(//5X,3A2,"THIS IS COMPRESSED"3A2,22X,"    THIS IS NORMAL ")
14     FORMAT(//5X,2A2,"UNDERLINE ON"2A2,25X,"UNDERLINE OFF")
15     FORMAT(//5X,3A2,20X"BOTTOM OF PAGE",3A2,20X"TOP OF PAGE")
C
C              BYE  BYE !
C
                STOP
                END
```

TOP OF PAGE

T H I S I S E X P A N D E D THIS IS NORMAL

THIS IS COMPRESSED THIS IS NORMAL

UNDERLINE ON UNDERLINE OFF

BOTTOM OF PAGE

SCHEDULING PROGRAMS FROM TYPE 6 FILES

Peter D. Cox/Computer & Engineering Services Brisbane Australia

All RTE III,IV, or IVB users have at some time been faced with the age old problem of scheduling programs that exist as type 6 files, either with or without creating a re-named copy.

If this scenario sounds familiar, then consider the requirements set out below. These were our criteria for development of the simple subroutine listed in Appendix A for use with RTE IVA or IVB.

REQUIREMENTS:

Our target system has to meet the objective of multiple users running the same or different data entry programs using one Image data base.

We require:

- Multiple copies of any program
- All programs are to exist in Type 6 files
- No program is to occupy an ID segment unless scheduled
- Multiple 'MENU' programs are to be selected according to User and/or terminal LU number
- Only HP supported software is to be used for ease of upgrade
- No special generation requirements are permitted so that modifications may be made without doing another system generation
- No terminal is to be tied exclusively to one task (as would occur if a special program were attached in the interrupt table) allowing all terminals to be used by system or group managers etc.
- There must be no special FMGR transfer files to maintain

BACKGROUND:

Recently there have been three articles in the Communicator which specifically satisfy some of the requirements stated above. They are briefly described below:

- Loading Program Segments from FMP Files — Larry W. Smith — Communicator Vol. III Issue 4

  This is a solution to a slightly different problem and requires special software. This flexible of an approach was not necessary in our application because all of our programs fit into 512 tracks (minus the system tracks) on LU's 2 and 3.

- Multiple Terminal Scheduler and ID Segment Manager — Michael P. Wingham — Communicator Vol. IV Issue 2.

  Our application made it necessary to use fewer transfer files and FMGR copies.

- Scheduling Data Acquisition Programs Without Providing General Terminal Access — Frank K. Fulton — Communicator Vol. III Issue 3.

  This article provides the basis for our solution, but we have enhanced Mr. Fulton's methods by removing the limitation of generating a special program into the system, and have further explored the FMGR routines that he uses in his solution.

PRESENT HP TOOLS:

The tools documented by HP (and the key word is ' documented ') provide the following solution:

1. Schedule a menu program from the user 'HELLO' file ('HI' file in IVA)

2. Use a transfer file to schedule the desired program

UNDOCUMENTED ROUTINES:

If one thinks for a minute about what FMGR is capable of doing with type 6 files and the re-naming of programs, it is obvious that there is another solution besides that described above.

This alternative solution comes in the form of three subroutines used in FMGR to carry out the processes that we users have been getting ulcers over for years.

The routines referenced above were brought to the surface by Mr. Fulton in the third mentioned article, but we took his suggestions a little further, and with a little experimentation, were able to utilize a few additional features of these routines.

- IDRPL Performs a FMGR 'RP' on a Type 6 file

  The calling sequence:

  ```
  CALL IDRPL(IDCB,IER,NAME)
  ```

  IDCB — The same data control block used in the 'CALL OPEN' statement that opens the file.
  IER — The FMP error return parameter.
  NAME — The program name required. This may or may not be the file name, in general it will not. Generally it will be the modified name for this copy of the program.

The 'NAME' parameter is the key to meeting the objectives of our design as it specifies the program name to be used in the new ID segment, and need not be the same as the program name coded into the type 6 file.

- IDRPD Releases an ID segment, and is equivalent to a FMGR :OF,prog,8 command.

  The calling sequence:

  ```
  CALL IDRPD(NAME,IER)
  ```

  NAME — ID segment name — must be the same as the 'NAME' parameter specified in IDRPL above.

  IER — FMP error return parameter.

- IDDUP Duplicates an existing ID segment giving it a new name

  The calling sequence:

  CALL IDDUP(NAM2,NAME,IER)

  NAM2 — The name of the new ID segment.
  NAME — The name of the existing ID segment.
  IER — FMP error return parameter.

Note that in this application IDDUP is not used since permanent ID segments are not used. The known error returns (IER parameter) are given in Table 1 of Appendix C.

THE SOLUTION:

Session monitor's 'PRMPT' program is enabled on interrupt (due to the entry in the interrupt table in the system generation) and each user 'Logs On'. So far nothing is different from a "vanilla" RTE-IVB system.

At this point the user's 'HELLO' file is activated and a decision is made regarding access based on the LU of the terminal ID use. If the decision is to schedule a menu select program, this is done by the HELLO file, and the operator is prompted by the menu program to select a function that he would like to perform. His choice is converted to an actual Type 6 file name (which program to run) and a subroutine called 'EXTY6' is entered. (The 'HI' file can perform the same function in IVA.) A sample 'HELLO' file is shown in Appendix D.

The subroutine EXTY6 now carries out the following operations:

- We pass the file name and session identifier (Global 8P or terminal LU number in IVA) to EXTY6
- EXTY6 constructs a new program name as follows-

  Required file is — PROGA
  Session identifier is — XX
  New name becomes — PRXXA
  e.g. PROGA on terminal 27 becomes PR27A

  NOTE

  The file manager convention 'PROXX' cannot be used since this format is is reserved for 'Sons' of FMGXX and will return an error -15 (illegal name) if used.

- EXTY6 opens the Type 6 file
- EXTY6 calls IDRPL to 'RP' the program with its new name PRXXA
- EXTY6 closes the Type 6 file
- EXTY6 schedules the program PRXXA with wait, passing it any parameters that have been specified. (In the example these are the terminal & printer LU numbers and the program's actual name.)
- EXTY6 waits until the program terminates and then calls IDRPD to release the ID segment.
- EXTY6 then returns to the caller.

SEGMENTED PROGRAMS:

EXTY6 satifies our design objectives if the main does all of the short ID segment management for its segments, and EXTY6 is used to schedule the main. The listing in Appendix B illustrates the flow for segmented programs.

The program in Appendix B uses the routine SEGLD which in turn uses T5IDM for short ID segment management. SEGLD is documented in the RTE-IVB Programmer's reference manual.

CONCLUSION:

We have a solition which is –

- Totally HP supported
- Uses the minimim number of ID segments
- Requires no special considerations in a system generation
- Can be nested as deep as we like e.g. AMAIN → PRXXA → PRXXB → PRXXC → etc.
- Is callable by any FTN4 or Assembler Main, Segment or Subroutine
- Uses no special routines
- Does not access system areas (such as ID segment words) directly

The only thing left is to puzzle at why H.P. has not documented these three routines that could save us all some lost sleep and frustration when our old system dies after an H.P. software upgrade because some piece of information in an ID segment word we were using has shifted or disappeared.

[Editor's notes: The three subroutines which Peter Cox describes in this article are documented in the new "RTE-IVB Technical Specifications" manual, HP part number 92068-90013. I highly recommend this manual for any sophisticated user that needs to know information about RTE internal workings.

Note that in the programs SEG1 and SEG2 below, a common block has been declared. In order for these programs to load successfully, appropriate parameters will have to be passed to the LOADR to allocate space for this common block.]

APPENDIX A:

Below is listed a Fortran subroutine to rename a Type 6 file, create an ID segment for it, and schedule the new program with wait.

```
FTN4,L
      SUBROUTINE EXTY6(NAME,IU,LU,IER,ISES), REV 03  -  RUN TYPE 6
     +FILES - 080180
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                              C
C  SUBROUTINE TO SCHEDULE A PROGRAM SP'D INTO A TYPE 6 FILE ON LU 2            C
C   OR LU 3 . THE PROGRAM IS RE-NAMED BY THE IDRPL ROUTINE AND THIS            C
C   ACTUAL NAME IS PASSED TO THE PROGRAM VIA SCHEDULE CALL PARAMETERS          C
C   3,4 AND 5 .                                                                C
C                                                                              C
C  RE-NAME SYNTAX  -  NEW NAME IS COMPOSITE OF ORIGINAL NAME AND               C
C   CALLING TERMINAL NUMBER WHERE CHARACTERS 1&2 ARE ORIGINAL NAME             C
C   3&4 ARE THE SESSION IDENTIFIER 'ISES' AND 5&6 ARE ORIGINAL.                C
C                                                                              C
C  THE TEMPORARY ID SEGMENT IS CLEARED BEFORE EXIT .                           C
C                                                                              C
C  PARAMETERS - NAME = FILE NAME OF REQUIRED PROGRAM                           C
C               IU   = TERMINAL LOGICAL UNIT NUMBER                            C
C               ISES = SESSION IDENTIFIER (TERMINAL NUMBER)                    C
C               IER  = ERROR RETURN VALUE                                      C
C                                                                              C
C  ERRORS  -  THE FOLLOWING ARE TESTED AND RETURNED IN IER                     C
C              0  - SUCCESSFUL CALL AND RUN                                    C
C              -6 - FMP ERROR - FILE NOT FOUND                                 C
C              14 - NO BLANK ID                                                C
C              16 - FILE NOT ON LU 2 OR LU 3                                   C
C              9  - ID NOT FOUND                                               C
C              18 - PROGRAM NOT DORMANT                                        C
C                                                                              C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
      DIMENSION NAME(3),NAM2(3),IDCB(144)
C
C  SET IER TO 0
C
      IER=0
C
C  COMPOSITE NEW NAME FROM NAME SUPPLIED AND TERMINAL LU
C
      NAM2(1)=NAME(1)
      NAM2(2)=NAME(2)
      NAM2(3)=NAME(3)
C
C  MOVE THE INTEGER VALUE ISES INTO THE NAME STRING
C
      CALL ISTRG(ISES,NAM2,3,4,3)
C
C  IF LESS THAN 10 REPLACE THE BLANK WITH ZERO
C
      IF(ISES.LT.10) CALL SFILL(NAM2,3,3,00060B)
```

```
C
C  OPEN THE TYPE 6 FILE WITH FMP OPEN - EXCLUSIVE ACCESS
C
      CALL OPEN(IDCB,IER,NAME)
      IF(IER.NE.6) GO TO 900
C
C  BUILD AND RP ID SEGMENT WITH THE NAME SET TO 'NAM2'
C
      CALL IDRPL(IDCB,IER,NAM2)
      IF(IER.NE.0) GO TO 900
C
C  CLOSE THE TYPE 6 FILE
C
      CALL CLOSE(IDCB,IER)
      IF(IER.LT.0) GO TO 900
C
C  SCHEDULE THE NEW PROGRAM WITH WAIT
C
      CALL EXEC(23,NAM2,IU,LU,NAM2(1),NAM2(2),NAM2(3))
C
C  RELEASE THE ID SEGMENT
C
      CALL IDRPD(NAM2,IER)
      IF(IER.EQ.0) GO TO 1000
C
C  CHECK ERRORS AND REPORT
C
 900  ASSIGN 60 TO IEF
      IF(IER.EQ.-6) ASSIGN 10 TO IEF
      IF(IER.EQ.14) ASSIGN 20 TO IEF
      IF(IER.EQ.16) ASSIGN 30 TO IEF
      IFTIER.EQ.18) ASSIGN 40 TO IEF
      IF(IER.EQ.9) ASSIGN 50 TO IEF
      WRITE(IU,IEF)IER,03440B
 10   FORMAT(/,"**** FILE NOT FOUND ****   ERROR ",I4,A2)
 20   FORMAT(/,"**** NO BLANK ID SEGMENT ****   ERROR ",I4,A2)
 30   FORMAT(/,"**** FILE NOT LU 2 OR LU 3 ****   ERROR ",I4,A2)
 40   FORMAT(/,"**** PROGRAM NOT DORMANT ****   ERROR ",I4,A2)
 50   FORMAT(/,"**** ID SEGMENT NOT FOUND ****   ERROR ",I4,A2)
 60   FORMAT(/,"**** ERROR ",I4," DETECTED ****")
 60   FORMAT(/,"**** ERROR ",I4," DETECTED ****")
C
C  RETURN TO MAIN PROGRAM WITH ERROR SET IN IER
C
 1000 RETURN
      END
```

The Fortran subroutine 'ISTRG' is called by EXTY6 to load the integer value of the session identifier into the 'NAME' string.

```
FTN4,L
      SUBROUTINE ISTRG(INO,ISTG,IBEG,IEND,ILEN),  REV 02 - INTEGER TO
     +STRING  010979 -  PDC
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                                                      C
C  SUBPROGRAM TO MOVE AN INTEGER VARIABLE INTO A DECIMAL               C
C  STRING IN A2 FORMAT.                                                C
C   NEG NUMBERS CONVERTED IN ACCORDANCE WITH THE                       C
C  DECIMAL ARITHMETIC ROUTINE.                                         C
C                                                                      C
C    PARAMETERS - INO - INTEGER NUMBER FOR CONVERSION                  C
C                 ISTG - RETURN STRING CONTAINING STRING               C
C                          CONVERSION OF "NUM".                        C
C                 IBEG - INTEGER GIVING START POSITION IN              C
C                          "ISTG" OF CONVERTED INTEGER                 C
C                 IEND - INTEGER GIVING FINISH POSITION                C
C                          (OUTPUT IS RIGHT JUSTIFIED IN "ISTG")       C
C                 ILEN - INTEGER GIVING LENGTH OF "ISTG"               C
C                                                                      C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
      DIMENSION ISTG(ILEN),INEG(5),KSTG(6),NSTG(1)
      LOGICAL NNEG
      DATA INEG/2H-J,2HKL,2HMN,2HOP,2HQR/
      NUM=INO
      NNEG=.FALSE.
      IF (NUM .GE. 0.0) GO TO 10
      NUM=NUM*(-1)
      NNEG=.TRUE.
 10   CALL CODE
      WRITE(KSTG,20)NUM
 20   FORMAT(6X,I6)
      IST=12-IEND+IBEG
      IF (IST .LT. 1) IST=1
      CALL SMOVE(KSTG,IST,12,ISTG,IBEG)
      IF (NNEG .NE. .TRUE.) GO TO 40
      CALL SMOVE(KSTG,12,12,NSTG,1)
      CALL CODE
      READ(NSTG,30)ITST
 30   FORMAT(I1)
      ITST=ITST+1
      CALL SMOVE(INEG,ITST,ITST,ISTG,IEND)
 40   CONTINUE
      RETURN
      END
```

APPENDIX B:

Below is an example of a segmented program that may be scheduled by EXTY6.

```
FTN4,L
      PROGRAM SGTST,3,2000
C
C  TYPE 5 SEGMENT LOAD TEST
C
      COMMON ISEG(3),ILAB,ISTAT,IPRAM(5)
      ISEG(1)=2HSE
      ISEG(2)=2HG1
      ISEG(3)=2H
C
C  RECOVER THE INPUT AND LIST LU NUNBERS
C
      CALL RMPAR(IPRAM)
      IU=IPRAM(1)
C
C  ASSIGN SEGMENT RETURN POINT
C
      ASSIGN 20 TO ILAB
C
C  SET PASS COUNTER ISTAT
C
      ISTAT=1
 20   WRITE(IU,30)ISTAT
 30   FORMAT(/,"I AM THE MAIN - PASS ",I2)
C
C  END AFTER 5 PASSES
C
      IF(ISTAT.GE.5) GO TO 50
C
C  LOAD THE SEGMENT WHO'S NAME APPEARS IN ISEG
C
      CALL SEGLD(ISEG,IERR)
C
C  OUTPUT ANY LOAD ERRORS
C
      WRITE(IU,40)IERR
 40   FORMAT("LOADR ERROR ",I4)
C
C  END OF RUN
C
 50   ISEG(2)=2HG1
C
C  RELEASE SHORT ID SEGMENTS BELONGING TO THE PROGRAMS SEGMENTS
C
      CALL IDRPD(ISEG,IERR)
      ISEG(2)=2HG2
      CALL IDRPD(ISEG,IERR)
C
C END
C
      END
```

```
C
C  SEGMENT NO. 1
C
      PROGRAM SEG1,5
      COMMON ISEG(3),ILAB,ISTAT,IPRAM(5)
      IU=IPRAM(1)
C
C  PRESET SEGMENT 2 AS NEXT SEGMENT
C
      ISEG(2)=2HG2
C
C  OUTPUT SEGMENT 1 MESSAGE
C
 10   WRITE(IU,20)
 20   FORMAT(/,"I AM SEGMENT 1 ")
C
C  INCREMENT PASS COUNTER
C
      ISTAT=ISTAT+1
C
C  RETURN TO MAIN
C
      GO TO ILAB
      END



C
C  SEGMENT NO. 2
C
      PROGRAM SEG2,5
      COMMON ISEG(3),ILAB,ISTAT,IPRAM(5)
      IU=IPRAM(1)
C
C  PRESET SEGMENT 1 AS NEXT SEGMENT
C
      ISEG(2)=2HG1
C
C  OUTPUT SEGMENT 2 MESSAGE
C
 10   WRITE(IU,20)
 20   FORMAT(/,"I AM SEGMENT 2")
C
C  INCREMENT PASS COUNTER
C
      ISTAT=ISTAT+1
C
C  RETURN TO MAIN
C
      GO TO ILAB
      END
      END$
```

As can be seen, all control is through one section of code which is the main, and this code releases the short ID segments before exiting.

APPENDIX C:

Table 1

PROGRAM	ERROR	MEANING
IDDUP(duplicate ID)	0	Successful completion
	15	ID not found
	16	ID not type 2 or 3
	17	ID not 'RP' ed
	23	ID copy already exists
	–15	ID copy name illegal
IDRPL('RP' Type 6 file)	0	Successful completion
	–11	File DCB not open
	14	No blank ID available
	–15	ID name illegal
	16	File not LU 2 or 3
	19	Type 6 file not 'SP' ed on current system
	23	ID name already exists
IDRPD(release ID)	0	Successful completion
	9	ID segment not found
	17	ID segment not an 'RP'
	18	Program not dormant

APPENDIX D:

The following is an example of a User 'HELLO' file used with EXTY6.

```
:SV,4
:IF,8P,EQ,18,7
:IF,8P,EQ,1,6
:IF,8P,EQ,7,5
:IF,8P,EQ,9,6
:IF,8P,EQ,11,5
:DP :DP,''INVALID LOG ON ADRESS''
::
:RU,STX45,1,6,8P,1
::
:RU,STX21,1,6,8P,1
::
```

Programs 'STX45' & 'STX21' are the menu programs for this user when logged on to a 2645 or 2621 terminal respectively.

The run parameters for STX45 and STX21 as used above are:

1 = terminal LU (always 1 in IVB)
6 = printer LU
8P = session identifier
1 = the users level of access within our menu program (STX45 or STX21)

REMOTE INTERACTIVE SESSION ACCESS

Lyle Weiman/HP Data Systems Division

[The additional software described in this article has been tested by the author and works as described. However, like other software discussed in the Communicator/1000, the programs discussed here must be totally supported and maintained by the user. Keep in mind that the routines described here may not be appropriate to your particular application, and future changes to the supported DS/1000 software may or may not be compatible with them. This article is offered solely to present one possible application of LU-mapping to extend a capability of the supported software.]

PURPOSE

If you've used DS/1000, even a little, you've probably wished that there was some way to gain access to a copy of File Manager on systems other than the one you were on. What is needed is a kind of "virtual terminal" capability whereby your own terminal can act like it is connected directly to any computer in the network. DS/1000-IV provides this "virtual terminal" capability directly, via LU-mapping, if you're willing to devote an LU to every potential remote user. LU's being in such short supply these days, a more dynamic method is needed, one which can assign these LU's only as required. This article describes how to combine DS/1000-IV software, with its LU-mapping capability, and some additional software newly contributed to the User's Library to attain this goal. The end result can be considered true Remote Interactive Session Access.

The HP-supported DS/1000-IV software provides remote session access, but it is not interactive. You can access files on specific private or group cartridges belonging to a specific account, and execute system-level commands, but you are limited to the capabilities provided by REMAT and the programmatic access methods such as RFA and PTOP. You are not talking to a File Manager, hence (from a remote terminal) you can't do such useful things as spooling compiler output, associating a spool file and an LU, modifying the Session Switch Table (SST), running a program saved as a type 6 file, loading programs which require some dialog with the LOADR without having to build a "batch" file, sending messages to operators or "mail" to various accounts, etc., etc. Most importantly, the remote job entry capability is severely limited. The supported software does provide enough "hooks" to allow you to do this however, with a little extra software from the User's Library.

Since LU mapping is a new capability, it may be useful to some readers to briefly describe what it is, what it provides, and how it works. Those familiar with this capability may skip to the section that deals with the "Theory of Operation".

BASICS OF LU MAPPING

LU mapping is just what the name implies: the ability to change ordinary I/O requests into I/O performed on devices which are in a node (nodes) other than the one in which the program is executing. At one stroke, it frees the user to access programs at any node in the network from any terminal in the network, without modifying any code in the programs.

The operating systems involved may be any RTE (memory-based or disc-based), and the topology of the network imposes no restrictions (other than that a path must exist along DS/1000-supported links between the two computers).

LU mapping is, by the way, a more general case of the "virtual terminal" capability mentioned above, because it operates with any unit-record device (terminals, printers, CTU's, punches, mag tapes).

The perceptive reader may have noticed that another way to create an LU mapping/virtual terminal environment would be to modify all programs to use DEXEC calls instead of EXEC or REIO calls for all non-disc I/O. This is true of course, but the concept of an "I/O map" allows the same generality without any of the reprogramming effort that would be necessary to modify an existing, working program, to allow passage of an additional parameter to it.

What is an "I/O map"? It is the transformation of a local LU number into a remote node and LU number. In practice, a certain set of LU's at a given node are set aside as "mappable" LU's. They "point" to an EQT (DVT in RTE-L) which "points" to a special driver. This driver is the heart of the software-implemented mapping process. The actual "map" is a two-word entry in the EQT/DVT (EQT's may have more than one "map" in them; the subchannel of the LU being accessed determines which entry in the map will be used). In conjunction with two other support programs (LUQUE and LUMAP), the driver is able to transform an ordinary EXEC I/O call (read, write, control, buffered, unbuffered, standard, re-entrant or class-I/O) into a remote (DEXEC) I/O call. For further details concerning I/O mapping or remote I/O, see the explanation in the DS/1000-IV Network Manager's Manual.

The concept of "LU mapping" is conceptually similar to the I/O mapping performed by RTE-IVB Session Monitor, but it's not the same. The "mapping" provided by Session Monitor switches I/O directed to one LU to another. Both LU's are in the same node, and the "map" to be used is taken from the particular session under which the calling program (doing the I/O) is executing. The term, "LU mapping", as used in this article, will be reserved exclusively to mean, the transformation of I/O directed from one LU at the "local" node, to a specific device at a particular remote node. "True" (system) LU's are always used to set up the maps, so there needs to be no concern as to what transformation Session Monitor might try to perform, the I/O bypasses this mapping.

There are many ways to accomplish remote interactive session access using LU mapping directly. The simplest of course, is to set up a unique I/O map at the session node for each possible remote terminal. When a user needs to log onto the session node, he would simply use REMAT to schedule a "clone" of the File Manager, specifying the proper LU at the session node which is mapped to his/her terminal. Note that the term "session node" is used to describe the remote node on which an interactive session is to be set up. This method of setting up a remote interactive session has several disadvantages:

1. There is a possibility for error if the wrong LU is typed.

2. Use of FMGR is limited to 20 minutes in this case because of the arbitrary limit placed on remote-schedule-with-wait requests. When the time limit expires, the FMGR clone is summarily aborted by DS/1000.

3. It is difficult to send a "break" to the remote FMGR, or any of its "sons".

4. The system generation must allocate a fixed set of LU's for the maximum number of remote users which can ever be logged on. LU's are such precious commodities these days, even in RTE-IVB nodes, and the potential number of remote users so large, as to make this last objection quite serious.

5. If alternatively, "temporary" LU mappings, made up from a pool of unallocated map entries, are used, how can they be returned back to the pool?

To overcome these objections, the programs RLOG and RLOGX were invented. One uses RLOG to initiate the remote log-on process. RLOG runs in the local node. RLOG calls upon RLOGX (at the Session node) to find an appropriate map entry, or to set one up.

THEORY OF OPERATION

The contributed software package consists of the programs RLOG, RLOGX, and the subroutines NMSC and IDUTL. RLOG is a small program which calls upon the program RLOGX, asking it to find an LU at the session node which is mapped to the node and terminal where the user is.

To start the process:

```
:RU,RLOG[,<terminal LU>[,<remote node #>[,<flag>]]]
```

The first parameter is the user's local terminal LU, which does not need to be specified, because FMGR and the MTM software will default it. The remote node number may be specified in the parameter string, which is a helpful feature if the user wants to set up softkeys for remote log-on, but it's not necessary (to make life simpler for those who have trouble remembering what parameters go with which programs in the vast pantheon of RTE programs). RLOG will ask you for the node number, if it was not specified in the run-string:

```
REMOTE NODE #?
```

As always, a "flag" parameter is provided for those cases where the remote node number is zero (a situation which is to be discouraged at every opportunity).

RLOG sends a request to RLOGX to obtain the LU mapped to the user's node and terminal, and awaits the reply. If RLOGX can't find a map set up for the user's node and terminal, it will attempt to set one up. Bit 8 in the map is set in this case to indicate a temporary map set-up. RLOGX returns the LU to be used to RLOG. It is advisable to have a "pool" of mappable LU's from which IOMAP can choose (RLOGX uses IOMAP to allocate an LU), and these must all have been assigned to subchannel zero. If IOMAP can't find an available "pool" entry, RLOGX returns a negative number instead of a valid LU to RLOG, which causes RLOG to print an error message.

If a suitable LU can be found, the map is set up for the user's terminal, and the program SYSAT (part of the supported software) is scheduled (RLOG and SYSAT must exist at your local node), passing to it the session node number and mapped-LU number. This causes SYSAT to generate a request for system attention at the session node, just as if a key had been struck at an enabled terminal local to that node. If the message is delivered without error, SYSAT prints "MESSAGE DELIVERED". If an error occurs, SYSAT calls DSERR and prints the message that DSERR returns.

PRMPT is scheduled as a result of the "interrupt" that was received from SYSAT, by the LU-mapping driver. From here, operation is the same as for the terminals physically connected to the session node: a request to LOGON is generated, the user is asked for the account name and password. All of this dialog is mapped automatically to the user's terminal on a remote system. If the account name and password typed in is acceptable, a session is then created. The user is now talking to a FMGR "clone", just as if he/she logged onto a local terminal, and can enter any commands supported by FMGR within the constraints of the account and the capabilty level.

Meanwhile, RLOG doesn't terminate, so the local FMGR is held off while the user communicates with the remote session.

How does one send a "BR" to programs running in the (remote) session once it's established? This function is handled by RLOG. Once the session has been established, RLOG goes into a loop, waiting one second, and then checking its "break" flag. If the flag is set RLOG schedules SYSAT again. RLOG saves the remote node and "mapped" LU numbers, so it doesn't have to find them again. For this reason, each user must have his own RLOG copy, if there is more than one terminal which is to be given remote interactive session access. The easiest way to do this is to allow RLOG to be cloned by making it a type 6 file in RTE-IV environments.

To set RLOG's "break" flag, at any terminal other than LU 1 at an RTE-IVB node, the user may simply enter "BR". At LU1, or in any other RTE operating system, the user must specify the program's name:

```
*BR,RLOG
  or
*BR,RLOxx
```

When RLOG detects that its "break" flag has been set (it should do this fairly quickly, since it checks every second), RLOG schedules SYSAT, which sends another request for system attention to the remote session. This time, however, since a session is already established, PRMPT issues a different response to the interrupt — the prompt for system attention:

```
S=<lu> COMMAND ?
```

Now, the user is talking to the remote session's operating system. Enter a "BR" to set the "break" flag of the "lowest son" in that session, according to the usual rules set by RTE-IV: thebreak bit in the user's clone of FMGR is set, unless FMGR has scheduled a program, in which case the break bit of this program, or its son, or its son, ...etc. to be set. Similarly, enter "SS" to suspend the program, thus allowing the user to temporarily put the remote session on "hold" while other operations are executed, either remotely or locally, outside of it. If the user wants to suspend the session for awhile without logging off, he/she can give the FMGR a "red-herring" command, such as a long directory list. The user doesn't care about the listing, he just wants to keep FMGR busy long enough for the necessary commands to be entered to suspend it. It is a good idea to set FMGR's "break" flag again (by typing BR,FMGxx) before re-scheduling it, so it won't continue to execute the "red-herring" command which was entered above, when it is next re-scheduled (which, incidentally, can be done by executing a "GO" command at the remote).

The first step in creating a remote interactive session is obtaining the attention of the local system. From here the user has many options, he may enter commands to the local node, or obtain the attention of a remote system, and enter commands to it. He/she may enter system-type commands to either system, FMGR commands to the remote session, or suspend the remote FMGR, and restart the local FMGR-clone, but remember, the user can only "talk" to one node at a time. This is in contrast to the capability REMAT provides, which allows talking between two nodes at a time, such as when copying a file from one node another. This difference must be borne in mind.

How are the "pool" entries returned, once allocated but no longer needed? RLOG does not "go away" as soon as it sets up the remote session. RLOGX keeps a table in which it saves the LU's it gives out, whether they are temporarily-allocated or not. This table also keeps the ID segment addresses of the FMGR-clones, which are entered as soon as they are created. If, after a certain number of times RLOG has checked and found that its break flag has not been set, it sends a "keep-alive" message to RLOGX. Each time RLOGX receives a "keep-alive" message, it scans its session table, entering the ID segment addresss of any new FMGR clones that are associated with a mapped LU, and the run-sequence number which is kept in that ID segment. RLOGX also checks the status of each FMGR-clone already in the table. If it no longer exists, or if its run-sequence number or ID segment address is different, then RLOGX returns the mapped LU if it was allocated on a temporary basis. RLOGX returns to RLOG whether or not the user's remote session is still active. If it is active, RLOGX also returns the number of sessions it's currently supporting. RLOG multiplies this number by 5 to determine how many passes thru the wait-one-second interval it should make before sending the "keep alive" message. This number is a maximum of 30 seconds, and must be a minimum of five seconds. The reason for not issuing this message every second, is to reduce the amount of time the central spends responding to "keep alive" messages. If the session no longer exists, RLOG terminates, thus returning the user to his/her local session.

There will be a slower response to the termination of a remote session than for other operator commands, due to the fact that the "keep alive" messages are sent less frequently as more remote sessions are supported. Every "keep-alive" message returns a fresh count of the number of sessions, so, if sessions terminate while the user is logged-on, the "keep-alive" messages will be sent more frequently, and response will improve.

RLOGX has table space for 99 sessions. If more than this number are allocated, a code is returned to RLOG, and the following message is printed:

```
TOO MANY SESSIONS ALREADY!!
```

When this is the case, the request for remote session log-on is denied. Since RLOGX's session table is already 99 entries big, and it checks for aborted and re-used entries, this message is an indication that all of the remote sessions that can possibly be created right now have been created. The user must try again later.

The fact that RLOGX automatically sets up the I/O map to print the "prompt" before each terminal 'read' can be annoying when you're editing (see the following examples). The program TPROM was written to "toggle" a bit which is used to indicate whether the node number of the remote session should be printed after each prompt. Hence the name "toggle prompt". Run once, TPROM will turn the bit off which means that just the ordinary program prompts will appear. Run again, the bit is turned back on, and the node numbers show up again. There are no parameters to memorize when running this program. TPROM can be run anytime after the user has logged on. It figures out what the 'true' LU is, where this LU is mapped to, etc., and runs IOMAP to set or clear the bit in the map which enables the prompt.

EXAMPLES

Below are some examples using RLOG to crete a remote interactive session.

Example 1

The user is sitting at a multipoint terminal (LU 81) connected to Node #1. Both Node 1 and Node 2 are RTE-IVB nodes, and Session Monitor is set up in both.

The user wishes to log onto Node #2.

```
:RU,RLOG,,2
```

RLOG prints the remote node numoer it is attempting to access, to confirm that the user supplied the proper number.

```
REMOTE NODE=     2
```

SYSAT prints a message indicating that it encountered no errors:

```
MESSAGE DELIVERED
```

RLOG now prints the number of the mapped-LU at the remote which it is using, and tells the user the name of the FMGR clone that will be used. It is useful to remember the name of the FMGR clone, as well as the LU number. If the user wants to set the "break" flag of either program he/she can either "break" the local copy of RLOG, or directly RU,SYSAT, <name of program>, <remote node #>.

```
REMOTE MAPPED LU=     29, FMGR CLONE= FMG29
```

The used next sees the log-on prompt from Node #2:

```
Node 2: account.group/password? MANAGER.SYS/PLUGG
```

LOGON Prompt	user responds with account name and password.

If a password is required for the account, it must be typed on the same line as the USER.GROUP. Unfortunately, too, it is thus visible. If the user has a CRT terminal, he may wish to clear the screen or the line immediately after entering the password.

This account name is valid, so the remote system responds with its usual log-on messages:

```
SESSION   29  ON   8:27 AM  FRI., 14  MAR., 1980
PREVIOUS TOTAL SESSION TIME:4356 HRS.,  16 MIN.,  44 SEC.
-------------------------------------------------------[03.05.80]----
```

Example 2

Now, suppose the user wants to compile two programs, using files stored at the remote node. Compiler output is to be spooled to the printer (LU 7) in two different ways. The source and resulting relocatable files are to be stored at the remote. Of course, it is possible to transfer the files across to the remote, either before initiating the remote session, or while the remote session is active.

```
:(     1)SL,7,,,7
FMGR-040
:(     1)??
FMGR -40 LU NOT FOUND IN SST          The user learns that LU 7 is not in his
:(     1)SL,7                         SST, so he adds it.
SLU    7=NOT DEFINE
:(     1)SL,7,7
:(     1)SL,7,,,7                     He then sets up a spool file.
:(     1)RU,FTN4,&RLOG,7,-
 $END FTN4: NO DISASTRS    NO ERRORS    NO WARNINGS
:(     1)CS,7                         The user releases the spool for printout.
```

The second way to spool the output is via the COMPL program. The advantage here is that the printout is produced at the same time the code compiles.

```
:(     1)RU,COMPL,&RLOGX,7,-
SPOOL FILE =  C09501
$END FTN4: NO DISASTRS    NO ERRORS    NO WARNINGS
/COMPL: END
```

Example 3

This example shows the "toggling of the prompt".

```
:(     1)RU,TPROM
```
Note that the "prompt" showing the remote node number is automatically set up. The user runs TPROM to turn it off.

```
:RU,TPROM
```
The prompt is no longer printed, the user runs Tprom again to turn it back on.

```
:(     1)
```
The prompt is back.

Example 4

In this example, the user wishes to verify that the correct version of a particular file is stored at the remote node, load the program, and replace the old type 6 file with a new one. Note that while one can load programs remotely via REMAT, it is cumbersome to create type 6 files afterwards. Notice also how easily this operation is done using remote interactive sessions. The load map comes out on the local terminal, the versions of the software can be verified, and the type 6 file created without cumbersome transfer files to be created, transferred, and purged later.

```
:(    2)LI,%RLOG,D
%RLOG  T=00005 IS ON CR00002 USING 00011 BLKS R=0000
:(    2)SYST,RLOG
NO SUCH PROG
:(    2)RU,LOADR,,%RLOG,1,SS
  RLOG    26042 27145  5-27-80
  SCSYS   27146 27370

  IFBRK   2738. 27424  92067-16268 REV.1913 790124
  .ENTR   27425 27514  750701  24998-16001
  LUTRU   27515 27623  92067-16268 REV.1903 790223
  RMPAR   27624 27666  781106  24998-16001
  CNUMD   27667 27706  92067-16268 REV.1903 770621
  ABREG   27707 27730  750701  24998-16001
  PARSE   27731 27750  92067-16268 REV.1903 770714
  GNODE   27751 27760  91750-1X123 REV 2013 800110
  DSERR   27761 30216  91750-1X076 REV 2013 791201 ALL
  POPEN   30217 31273  91750-1X148 REV.2013 800501 MEF RTE-RTE-MPE
  D3KMS   31274 33654  91750-1X064 REV.2013 800408 MEF
  CNUMO   33655 33674  92067-16268 REV.1903 770621
  D$CON   33675 33676  91750-1X052 REV.2013 800201 MEF: 304 3K BUF
  CLRIO   33677 33705  750701  24998-16001
  $PARS   33706 34126  92067-16268 REV.1903 780811
  $CVT3   34127 34214  92067-16268 REV.1903 770621
  LOGLU   34215 34272  92067-16268 REV.1903 790228
  REIO    34273 34417  92067-16268 REV.1903 790316
  .DFER   34420 34471  750701  24998-16001
  RNRQ    34472 34733  92067-16268 REV.1903 780222
  IFTTY   34734 35021  92067-16268 REV.1903 790118
  #MAST   35022 35737  91750-1X019 REV 2013 800403 ALL
  CLRQ    35740 36054  91750-1X047 REV 2013 791129 (IV,M2,M3)
  #GET    36055 36321  91750-1X225 REV 2013 800222 ALL
  #MAAS   36322 36741  91750-1X015 REV 2013 800415 ALL (MA)
  #MSSM   36742 40343  91750-1X020 REV.2013 800502 RTE-IVB W/S.M.
  D$3BF   40344 41024  91750-1X050 REV.2013 800201 MEF: 304 3K BUF
  $ALRN   41025 41142  92067-16268 REV.1903 770715
  $SMVE   41143 41231  92067-16268 REV.1903 790202
  SESSN   41232 41247  92067-16125 REV.1903 780413
  #LOGR   41250 41317  91750-1X013 REV 2013 800110 ALL
  #GETR   41320 41504  91750-1X010 REV 2013 800407 ALL

   7 PAGES RELOCATED    7 PAGES REQ'D   NO PAGES EMA     NO PAGES MSEG
  LINKS:BP    PROGRAM:BG   LOAD:TE   COMMON:NCSS
   /LOADR:RLOG   READY AT  8:14 AM  THU., 29  MAY , 1980

   /LOADR:$END
:(    2)PU,RLOG
:(    2)SP,RLOG
```

Example 5

In this example, the user wants to know what entries are in the SST (Session Switch Table) for this remote session.

```
:(     2)SL
SLU   1=LU  # 29 = E24
SLU   2=LU  #  2 = E 1
SLU   2=LU  #  2 = E 1
SLU  13=LU  # 13 = E 1 S 1
```

Session LU 1 is switched by Session Monitor to the mapped LU, #29.

Example 6

The user is curious about the activity at the remote.

```
:(     2)WH,AL

  7:59:28:610
 ------------------------------------------------------------------------
   PRGRM T PRIOR PT SZ DO.SC.IO.WT.ME.DS.OP.     .PRG CNTR.   .NEXT TIME
 ------------------------------------------------------------------------
 **FMG29 3 00095  4 10 * * * *  3,WHZ29  * * * * * P:34542SWP
   WHZ29 3 00001  4  7 .  1, . . . . . . . . . . . P:32647
   LUMAP 3 00030  6  6 .  1, . . . . . . . . . . . P:26105
   PROGL 3 00030  5  6 . . . .  3,CL  024  . . . . P:34462SWP
   GRPM  3 00004  3  6 . . . .  3,CL  036  . . . . P:31747
   QCLM  3 00028  1  2 . . . .  3,CL  034  . . . . P:26047
   INCNV 3 00020  6  3 . . . .  3,CL  031  . . . . P:31037SWP
   RSM   3 00020  2  4 . . . .  3,CL  032  . . . . P:30213
   OTCNV 3 00020  4  3 . . . .  3,CL  030  . . . . P:31073SWP
   PTOPM 3 00030  4  7 . . . .  3,CL  029  . . . . P:27305SWP
   RFAM  3 00030  6  9 . . . .  3,CL  023  . . . . P:41035SWP
   OPERM 3 00030  4  3 . . . .  3,CL  028  . . . . P:26251SWP
   DLIST 3 00030  4  5 . . . .  3,CL  027  . . . . P:31423SWP
   EXECW 3 00030  4  7 . . . .  3,CL  026  . . . . P:27573SWP
   EXECM 3 00030  2  4 . . . .  3,CL  025  . . . . P:30511SWP
   UPLIN 3 00003  6  3 0,  . . . . . . . . . . . . P:00000  7:59:46
   MATIC 3 00030  6  2 0,  . . . . . . . . . . . . P:00000  7:59:44
   RTRY  3 00020  1  2 . . . .  3,CL  035  . . . . P:26365SWP
   LOGON 3 00050  4 12 . . . .  3,CL  038  . . . . P:30137SWP
   LGOFF 3 00090  4 10 . . . .  3,CL  039  . . . . P:27133SWP
   R$PN$ 3 00005  5  4 . . . .  3,CL  037  . . . . P:26062
   FMG12 3 00095  6 10 . . . 2,EQ:  2,AV:2,ST:002  P:44204SWP
 ------------------------------------------------------------------------
 ALL LU'S OK
 ALL EQT'S OK
 MAX CONT. FREE TRKS :    37,  LU   2
 ------------------------------------------------------------------------
  7:59:50:810
```

Now, the user wishes to sign off of the remote system.

```
:(     2):EX,SP
 $END FMGR
FMG29
ABOVE SESSION PROGRAMS ACTIVE
OK TO ABORT ?  (Y OR N)FMG29 ABORTED

SESSION   29  OFF  8:10 AM  THU., 29  MAY , 1980
CONNECT TIME:              00 HRS.,  02 MIN.,  22 SEC.
CPU USAGE:                 00 HRS.,  00 MIN.,  10 SEC.,   460 MS.
CUMULATIVE CONNECT TIME:4082 HRS.,  34 MIN.,  48 SEC.
END OF SESSION
```

The usual session log-off messages follow. Sometime during, or after these messages, RLOG determines that the remote session has terminated, and prints the message:

```
REMOTE SESSION ENDED
```

RLOG then terminates, returning the user to his own (local) FMGR-clone. He is now "returned" to his original local session.

To illustrate, the user obtains another cartridge listing.

```
:CL
  LU  LAST TRACK   CR   LOCK  P/G/S

  34      00202   00703          P
  57      00405   00111          P
  21      00202      BR          G
  02      00254   00002          S
  03      00254   00003          S
  61      00405   00050          S
  60      00405   32767          S
  62      00405   00045          S
  20      00202   01905          S
  54      00405      TR          S
  55      00405      MM          S
  33      00202      ?C          S
  63      00405   01000          S
  27      00202      LC          S
  22      00202   02001          S
  29      00104      RM          S
:
```

Example 7

In this example, the user constructs several files to handle a repetitive task (in this case, that of sending a file over to a remote node to be compiled, and returning the relocatable file to the local node for relocation), and simply transfers to the file to do all the work. This situation might arise for example, when debugging a program at one node, with a fast printer located at a remote node, or the only copy of the compiler at the remote node etc. The user would then be in the situation where he must repeatedly send "new" versions of the source files to that remote, compile them, bring back the relocatable, load it up, run the program, find more bugs, edit the source, and go around again. Clearly, he would want transfer files to make all of these operations go quickly and uniformly each time. Although the example is shown as a Fortran program, it could just as well be PASCAL/1000, or any other language supported on HP 1000 systems.

The example program is quite trivial:

```
&TEST  T=00004 IS ON CR00002 USING 00001 BLKS R=0000

  0001  FTN4,L
  0002        PROGRAM TEST
  0003        WRITE(1,1)
  0004  1     FORMAT("THIS PROGRAM IS A TEST")
  0005        END
  0006        END$
```

The three transfer files consist of one FMGR file and two REMAT files. They all exist at the remote node (Node #1), and are executed there.

The first file simply runs REMAT, sets up a spool LU, compiles the source file, runs REMAT again, and logs off.

```
*JOB   T=00004 IS ON CR00002 USING 00001 BLKS R=0000

0001  :RU,REMAT,*REM1
0002  :SL,7,7
0003  :SL,7,,,7
0004  :RU,COMPL,&TEST,7,%TEST
0005  :RU,REMAT,*REM2
0006  :EX,SP
```

The second file purges the previous versions of the source and relocatable files, and moves the new source file to Node #1 on cartridge 703.

```
*REM1  T=00003 IS ON CR00002 USING 00010 BLKS R=0000

0001  $PU,&TEST::703
0002  $PU,%TEST::703
0003  $SW,2:MANAGER.SYS/PLUGG,,DS
0004  $ST,&TEST,&TEST::703
0005  $EX
```

The third transfer file moves the relocatable back to the user's node.

```
*REM2  T=00003 IS ON CR00002 USING 00010 BLKS R=0000

0001  $SW,1,2:MANAGER.SYS/PLUGG,DS
0002  $ST,%TEST::703,%TEST
0003  $EX
```

Note that taken together, these files simply perform the job of moving a new source file to a remote node, compiling it, and moving the relocatable to the user's node, purging all "old" copies. After each execution of these files, two copies of the "new" versions of these files exist, one at the local node and one at the remote. If something should happen, say, to one of the discs, causing one file to be lost, there still remains a redundant copy.

Example 8

Below, the user logs on and executes his "job".

```
Account/Password? MANAGER.SYS/PLUGG

SESSION    1  ON  12:26 AM  FRI., 30  MAY , 1980
PREVIOUS TOTAL SESSION TIME:4090 HRS.,  27 MIN.,  17 SEC.

:RU,RLOG,,1
REMOTE NODE=      1
MESSAGE DELIVERED
REMOTE MAPPED LU=      95, FMGR CLONE=  FMG95
 Octoputer Logon? (    1)JOE.USER/PASSWORD

SESSION   95  ON  12:26 AM  FRI., 30  MAY , 1980
PREVIOUS TOTAL SESSION TIME: 217 HRS.,  55 MIN.,  28 SEC.
------------------------OCTOPUTER SYSTEM----------------------------

:(    1):*FMG1
 :RU,REMAT,*REM1
$PU,&TEST::703
$PU,%TEST::703
$SW,2:MANAGER.SYS/PLUGG,,DS
SESSION  252  ON  12:27 AM  FRI., 30  MANAGER.SYS
$ST,&TEST,&TEST::703
$ST,&TEST,&TEST::703
$EX
 $END REMAT
SESSION  252  OFF 12:27 AM  FRI., 30  MANAGER.SYS
 :SL,7,7
 :SL,7,,,7
 :RU,COMPL,&TEST,7,%TEST
SPOOL FILE =  C09501
 $END FTN4: NO DISASTRS   NO ERRORS   NO WARNINGS
/COMPL: END
 :RU,REMAT,*REM2
$SW,1,2:MANAGER.SYS/PLUGG,DS
SESSION  252  ON  12:29 AM  FRI., 30  MANAGER.SYS
$ST,%TEST::703,%TEST
$EX
 $END REMAT
SESSION  252  OFF 12:29 AM  FRI., 30  MANAGER.SYS
 :EX,SP
 $END FMGR
FMG95 REMOVED

SESSION   95  OFF 12:29 AM  FRI., 30  MAY , 1980
CONNECT TIME:              00 HRS.,  02 MIN.,  17 SEC.
CPU USAGE:                 00 HRS.,  00 MIN.,  07 SEC.,   980 MS.
CUMULATIVE CONNECT TIME: 217 HRS.,  57 MIN.,  45 SEC.
END OF SESSION
REMOTE SESSION ENDED
:
```

STATUS INFORMATION

To obtain a printout of the current remote sessions logged on by RLOGX, simply run RLOG, specifying a third parameter:

```
:RU,RLOG,,<node>,PR
```

The "PR" specifies that, instead of a log-on request, RLOGX is to print the contents of its session table on the terminal from which it was run. An example is shown below:

```
:RU,RLOG,,2,PR
REMOTE NODE= 2
REMOTE  TERML SESSION FMGR
 NODE#   LU#    ID   CLONE
     1   127    29   FMG29   *
    98 EMPTY ENTRIES
```

The first column lists the remote node, at which the user's session exists. The second column gives that terminal's "true" (i.e., system) LU. The third column shows the session ID, and the fourth column shows the name of the FMGR-clone assigned to the user (the last two digits of the name should match the session ID number). If an asterisk (*) follows, it indicates that the LU mapping set-up is temporary. At the end of the listing, a count of the number of empty table entries is given, or "NO EMPTY ENTRIES" is printed.

INSTALLATION

The HP documentation describes the installation of LU mapping, and will not be repeated here, except the specific requirements pertaining to this application.

First and most obviously, all session nodes which are to support remote session access must have all of the LU mapping software (DVV00, LUQUE, LUMAP and IOMAP) installed; they must be programs with ID segments, not files which must first be RP'd before running them. RLOGX is subject to the same conditions. Program-to-program communications must also be enabled at the session node(s).

The program TPROM must be loaded with the subroutine %NMSC. It may have an ID segment, or it may be saved as a type 6 file.

A number of mappable LU's should be defined which "point" to the mapping driver thru EQT's. Some of these may be set-up in permanent maps, but the remainder should be part of the "pool" of LU's available for remote users.

The LU's must be less than 100, because the names of the "son" program clones in session become changed in the last two characters, into the low two digits of the terminal LU's (which, in this case, will be the mapped LU number, not the "true" LU of the remote user's own terminal). This is done by the FMGR-clones anytime they schedule a "son" program. These LU's must also have a subchannel assignment of zero. RLOGX won't allocate any LU's for mapping if the subchannel assignment is non-zero (programs determine whether or not their input LU is interactive, based upon driver type and sometimes subchannel 0). Non-zero subchannels could be reserved for related devices of the same type, such as the CTU's. If doing so, it is somewhat more efficient for RLOGX if these non-zero subchannel LU's are higher in number than the last zero-subchannel LU.

As part of the boot-up initialization process, the user may wish to set up some permanent I/O mappings. That is, grant certain terminals their own "maps" for every access. This may be desirable if, for example, only certain terminals were to be given the capability of remotely logging on (here, we assume that the physical security of the terminal's location is being utilized to prevent unauthorized persons from getting into the network). In this case, there would be no "pool" map entries, all LU's would be assigned. In less restrictive environments, it is better to leave all entries in the "pool" for RLOGX to distribute and return to the pool when the user is finished. RLOG and RLOGX will work under either LU configuration. RLOGX allows for assigned LU's by checking for maps which are already set up, before trying to allocate a "pool" entry. To control access, one simply does not provide any "pool" entries.

How does one set up these "maps"? The answer is, quite easily, using IOMAP. Suppose,for example, nodes 100, 101 and 102 are to be given remote interactive session access, each with terminal LU's 1 and 7. This requires six mappable LU's Let's further suppose that LU's 30 thru 35 have been reserved for this purpose, meaning they "point" to EQT's associated with DVV00. Separate EQT's are necessary for each LU, because of the fact that PRMPT operates from an EQT address, converting it to an LU.

So, to set up these maps at boot-up time, the following commands would be added to the WELCOM file (assume that the necessary set-up security code is DS):


```
      .
      .
      .
:RU,IOMAP,30,1,100,DS
:RU,IOMAP,31,7,100,DS
:RU,IOMAP,32,1,101,DS
:RU,IOMAP,33,7,101,DS
:RU,IOMAP,34,1,102,DS
:RU,IOMAP,35,7,102,DS
      .
      .
      .
```

At every node which is to be allowed remote interactive access, programs RLOG and SYSAT must be installed (as well as the DS/1000 programs for basic communications of course, and EXECM must be installed and enabled to handle the I/O).

When generating the session node, in addition to the other steps taken to include LU mapping, add an interrupt table entry for the select code used for the mapping (since any select code can be used, be sure to use an unique one). Usually 77B is used, so enter the following as the last interrupt table entry:

```
77,PRG,PRMPT
```

CAVEATS

Remote interactive session access has many advantages, but, as with any powerful tool, there are some dangers. The most obvious danger concerns the ability for remote users to gain access to the MANAGER.SYS (or other high-capability) accounts. From MANAGER.SYS, one can run ACCTS to modify the accounts structure. By switching other LU's to "point" to the same EQT and subchannel as mapped LU's, one can "steal" those local LUs. The devices normally associated with them would become inaccessible, including, and most especially, the system console. There are many more scenarios. Clearly, remote interactive session access is meant for environments where the users can be relied upon to respect these security limitations. Guarding the passwords for high-capability accounts becomes very important, in any case.

Furthermore, there are some "rough spots" in this implementation. Remote interactive session is not available from an HP 3000 terminal. Regaining local system control is rather cumbersome, and obtaining the remote's attention is not straightforward. The user can't switch back and forth between several remote sessions; only one is available at a time, although RLOG could be modified to make this possible, albeit cumbersome.

POSSIBLE IMPROVEMENTS AND ENHANCEMENTS

Users may wish to modify the software to fit individual tastes and applications.

The system manager might want to include in the "HI" file for each available account some FMGR commands which will do such things as check for remote use. Certain accounts perhaps should be prohibited from any remote access. With IOMAP, the user can find out whether a given LU is mapped, and if so, where. Perhaps only specific terminals are to have remote access. The manager might want to set up some station LU's to be the user's CTU's and local printer, which would be the same LU's regardless of where he/she is. Perhaps it is desired to include in the "HI" file some FMGR commands to set up the soft-keys.

Depending on the network, the user might wish to modify RLOG so that it will accept a default remote node number, if there is only one session node available for remote interactive session access.

Perhaps table space for 99 sessions is a little more than needed, and it is desired to reduce RLOGX's size (small programs take less disc space, swap faster, etc.). Simply reduce the size of SESTAB to the maximum number of mappable LU's in the given node, and change MAXSES's initialization accordingly.

CONCLUSION

This article has demonstrated how to set up a network so that session node(s) can be accessed interactively from anywhere in the network, what advantages can be obtained, and what dangers exist. The article has also described three application programs which streamline the process of creating a remote interactive session, and what enhancements might be added to them.

HP SUBROUTINE LINKAGE CONVENTIONS

Robert Niland/HP Lexington

[Editor's Note: This article is the seventh in a series which, when complete, will provide our readers with the equivalent of a manual on the standard HP linking conventions.]

7-1. HIGH LEVEL LANGUAGE CONVENTIONS

FORTRAN IV Subroutines

Typical Fortran subroutine calls look like:

```
        CALL RMPAR (ISCHED)
        CALL WRITF (IDCB,IERROR,IBUFR,LENGTH)
993     CALL STOP
```

Where the general syntax is:

[lbl] CALL < name > [(< param1 > [,< param2 > ... [,< param n >]])]

Where: [lbl] is an optional statement label.

< name > is the name of the subroutine being called. The use of the reserved word "CALL" will generate an EXT reference to < name >, unless < name > is a STATEMENT FUNCTION within the calling program.

< param n > ..to.. < param n >are any number of parameters consisting of literal constants, variables, arrays, array elements, expressions or implicit calls to FUNCTIONS.

The assembly language equivalent of the above syntax is:

```
        ...
[lbl]   JSB <name>              CALL the routine.
        DEF *+n+1               Define return point.
        DEF <param1>            Pass 1st parameter (if any)
        ...                     Pass 2nd & succeeding...
        DEF <param n>           Pass last parameter address
        ...                     Return point.
```

Note that if no parameters are passed, the equivalent assembly level call is:

```
        ...
[lbl]   JSB <name>              CALL the routine.
        DEF *+1                 Define return point.
        ...                     Return point.
```

As you can see, both of these sequences conform to the rules set down in chapters 5 and 6 [published in Vol. 4, issues 3 and 4 of the Communicator] for calling .ENTR and .ENTP type subroutines.

However, Fortran does not give the same flexibility and power as assembly language, and it is worth considering some aspects of Fortran's assumptions and limitations.

The < name > of the subroutine must be composed of 1 to 5 upper case ASCII characters, beginning with an alphabetic character. It may be 6 characters long if your compiler has the truncate extension, and may be longer, or contain punctuation or lower case if your compiler has the ALIAS extension.

A normal CALL < name > will result in an (assembler) EXT reference being generated by the compiler to that program module. It is also possible to satisfy the CALL if the < name > being called is that of a STATEMENT FUNCTION within the calling routine. Of course, by CALLing a statement function, you cannot obtain its normally returned results.

< name > may also appear in a Fortran EXTERNAL < name > statement. If you are CALLing a < name > within the routine, this declaration serves only as documentation. If < name > is a statement function you will get an error at compile time. About the only reason you might want to declare < name > EXTERNAL is if you desire to pass the address of < name > in another call, and nowhere make a CALL to < name > itself. For example, suppose we want to know the memory location of some subroutines, and we have a routine IGETA which returns the address of the operand passed to it:

```
      ...
      EXTERNAL MYSUB
      ...
      CALL SIGMA (SUMX,SUMXX,N)
      ...
      IADRSG = IGETA (SIGMA)
      ...
      IADRMY = IGETA (MYSUB)
      ...
```

The parameter list rules are:

1. The minimum number of parameters is zero, in which case the (parentheses) must be omitted.

2. The maximum number of parameters is limited by available memory or the number of continuation lines.

3. All parameters are utimately passed by reference. This fact is obvious for simple variables. It is not so obvious for literal constants, array elements and expressions. To pass parameters by value the user progran must copy that parameter into a temporary variable, and then pass the temporary by reference.

4. If the parameters are literal constants, be cautioned that the subroutine must not modify parameters passed to it. Or, conversly, the user program must not pass it any. For example, the library routine NAMR has the calling sequence:

   ```
   CALL NAMR (IPBUF,INBUF,LENGTH,ISTRC)
   ```

   For convenience in parsing strings which contain multiple FMP namrs, the character pointer ISTRC is updated after each call to NAMR. If NAMR is passed a literal constant in position 4 of this call, unexpected things will happen. Observe:

   ```
         ...
   C     Parse input STRING(1..23) into buffer FNAMR1
         CALL NAMR (FNAMR1,STRING,23,1)
         ...
   ```

   Let's suppose that NAMR parsed 17 characters from STRING into FNAMR1. Now at some later time in the program we again use the literal constant "1":

   ```
         ...
         DO 900 INDEX = 1,33
         ...
   900   CONTINUE
         ...
   ```

Much to our dismay, the loop only executes 15 times (beginning with INDEX set to 18 and ending at 33)! What has happened is that the memory location containing the literal 00001 was updated to 18 by the call to NAMR. The proper way to make the example call to NAMR is:

```
...
ISTRC = 1
CALL NAMR (FNAMR1,STRING,23,ISTRC)
...
```

Note that you cannot use the EMA call-by-value form:

```
CALL NAMR (FNAMR1,STRING,23,(1))
```

Only EMA variables will be passed by value using this notation. For ordinary constants, variables and expressions the extra parentheses are ignored.

5. Array elements are also passed by reference. The caution in this case applies to whether or not the subroutine being called requires information saved in the working registers. For example, the library routine ABREG returns to the user the contents of the A and B registers. If we call it passing an array element which requires address calculation, that calculation will be performed before the call, and may alter the registers, e.g.

```
...
CALL ABREG (KSTATS(NEXT),IB)
...
```

will assemble as:

```
        ...
        LDA NEXT          Get value of subscript.  Note: A lost!
        ADA @<KSTATS>-1   Add to base address of array.
        STA T.001         Store in temporary.
        JSB ABREG         Now call ABREG.
        DEF *+2+1         Define return point.
T.001   DEF T.001,I       Pass computed address of KSTATS(NEXT)
        DEF IB            And address of "IB".
        ...               Return point, with A = T.001 !
```

6. If the parameter is an expression, it will be evaluated prior to the call, and the address of the memory location containing its result will be passed in the call. This scheme of parameter passing is referred to as "call-by-value". EMA variables are considered to be expressions when passed to subroutines which are not expecting an EMA parameter. You may pass EMA variables by reference to subroutines which are expecting only an EMA parameter in that position. Passing EMA variables by value to a subroutine which is expecting only an EMA variable requires enclosing the parameter in an extra layer of parentheses.

7. Trailing parameters may be omitted (the user may supply a partial list), but only if the routine being called is programmed to handle this. Let's review the consequences of defaulting parameters when the subroutine is not programmed to handle it. As an example, let's take a simple case; the subroutine ADDEM which returns the sum of the parameters passed to it:

```
...
RESULT = ADDEM (11.0,33.4,3.0,4.5)
...      RESULT will = 51.9 at this point
...
RESULT = ADDEM (22.0,10.1)
...      RESULT will = 39.6, not 32.1 !
...
```

Since ADDEM does not reset the links for parameters 3 and 4 they are still pointing at the constants 3.0 and 4.5 and these will be included in the sum.

7-2. FORTRAN FUNCTION SUBROUTINES

Typical Fortran FUNCTION calls are of the form:

```
      INT = IGET (52133B)
103   FLT = SQRT (VALUE)
      CPLEX(33) = CONJG (X)
      TDP(20) = TBLE (DEX(10))
```

Where the general syntax is:

[lbl] < variable > = < name > (< param1 > [, < param2 > ... [, < param n >]])

A function can be used almost anywhere that any other expression can be used. Functions may also be CALLed as subroutines, although there are some caveats of which you must be aware.

Below is a summary of the syntax of a function call as contrasted with ordinary subroutines:

1. The element on the left side of the " =" must be one of the following:

   a. A simple variable.
   b. An array variable.
   c. An EMA variable.

2. The leading character of the < name > will determine the type of the return variable (INTEGER or REAL) unless overriden by a declaration, such as:

   ```
   DOUBLE PRECISION TBLE
   ```

   Note that the use of:

   ```
   IMPLICIT DOUBLE PRECISION (T) or (A-Z)
   ```

   does not affect the type of any function.

3. There must be at least one parameter passed to the function, even if it is only a dummy. The return value of the function may be passed as a parameter in another subroutine/function call if one or more parameters appear in < name >. An example is shown below:

   ```
   CALL SUBR (PARM1,FUNC(PARM2),PARM3)
   ```

   If FUNC requires no parameter, a dummy must be supplied, for if a function subroutine is used as a parameter of a subroutine, and no dummy is supplied, the function < name > is likely to be used to indicate the address of a local variable called < name >, or the address of the local/external function < name > if called or declared as EXTERNAL < name >.

The equivalent assembly code for a function subroutine call depends on the type (integer,real, etc.) of the subroutine. Calling a Fortran function from an assembly program and writing a Fortran-compatible assembly function both require that you understand this protocol. The following paragraphs will deal with each data type separately.

Integer Functions:

The result of an integer function is returned in the A register. The assembler code generated for the calling sequence:

< result > = < name > (< param1 [, < param2 > ... [, < param n >]])

is:

```
        ...
        JSB <name>        Call function.
        DEF ]RET          DEFine return point.
        DEF <param1>      Define parameter 1.
        DEF <param2>      Define parameter 2.
        ...               etc...
        DEF <param n>     Define parameter n.
]RET    STA <result>      Store the returned value.
        ...
```

Inside the Fortran function, the last instruction before the JMP < name >,I is an LDA < result >. In assembler you can set up A at any point, but make sure that the result is still in A before returning from the function. In Fortran, the < result > value gets created automatically when you execute:

< name > = < expression >

within the function.

Real Functions:

The result of a real function is returned in both the A and B registers. The assembly code generated for the calling sequence:

< result > = < rname > (< param1 [, < param2 > ... [, < param n >]])

is:

```
        ...
        JSB <rname>       Call function.
        DEF ]RET          DEFine return point.
        DEF <param1>      Define parameter 1.
        DEF <param2>      Define parameter 2.
        ...               etc...
        DEF <param n>     Define parameter n.
]RET    DST <result>      Store the returned value.
        ...
```

In a Fortran function, the last instruction before the JMP < rname >,I is a DLD < result >. If the user is writing a function in assembler that is to be called from Fortran he/she can set up A&B at any point, and and just leave them that way until exiting from the function. In Fortran, the < result > value gets created automatically when you execute:

< rname > = < expression >

within the function.

Extended Precision, Double Precision & Complex Functions

Extended precision (DEX), double precision (TDP), and complex (CPX) variables all require more than two 16 bit words for storage. Consequently functions returning values in these formats cannot use the A and B registers to return that value. X and Y are not brought in either, because not all 2100 series computers have those registers. So that leaves only two choices, either we pass back something which will fit in A and B (such as the address of the result), or we make the result another parameter.

For reasons which this manual will not explore, the method used is the second. DEX, TDP, and CPX functions return the result through a hidden parameter. The Fortran call:

DOUBLE PRECISION < result >, < name >, < pram1 >, .. [, < pram >]
...
< result > = < name > (< pram1 > ... [, < pram n >])

translates into the following assembly sequence:

```
...
JSB <name>          Call the function.
DEF *+n+2           Define return point.
DEF <result>        Define the result variable.
DEF <pram1>         Pass the arguement variables.
...                 etc...
DEF <pram n>        Pass the final arguement variables.
...                 Return point.
```

In a Fortran function, the caller's result variable gets set to the result value automatically when you execute:

< name > = < expression >

within the function.

In an assembly routine, you must explicitly store the result back through the hidden parameter's link. For an example of this, refer to the listing of TBLE in the appendix. [The appendices are to be printed in a future Communicator.]

Note that because of the hidden parameter, any DEX, TDP or CPX function can be invoked as a function or as a subroutine. For example, either of the the following forms are valid:

```
DOUBLE PRECISION TBLE,TDP
INTEGER DEX(3)
...
TDP = TBLE (DEX)
  or
CALL TBLE (TDP,DEX)
...
```

HOW TO READ IBM FLEXIBLE DISCS ON AN RTE-L SYSTEM

Joel Dubois/HP Grenoble

It is possible to read from or write to IBM formatted flexible discs using RTE-L Exec calls.

The 7902A flexible disc subsystem is designed as a double-sided, double density disc drive. The hardware and firmware have also been designed to read and write in the single density format which is used by IBM 3740 data entry equipment. Once a disc is inserted into the drive, the controller determines which format is present and then proceeds to interpret the data appropriately from a hardware standpoint.

The format for IBM flexible discs, however, is different from that used on HP flexible discs:

DIFFERENCES BETWEEN IBM AND HP FLEXIBLE DISCS

	HP Double Density	**IBM Single Density**
Bytes per sector	256	128
Sectors per track	30	26
Sector numbering	0-29	1-26
Sides	2	1
Tracks per side	77	77
Track numbering	0-76	0-76
Bytes per drive (formatted)	1.18 Mb (154 tracks)	243 Kb (data)

To be able to access both HP and IBM formatted floppies, the user needs to define a "dummy" subchannel pointing to the 7902. This allows the programmer to set up a track map for IBM formatted discs. Then, routines can be written to perform one or all of the following functions:*

- Initialize an IBM formatted disc
- Read and write string data from/to an IBM formatted disc
- Read and write from/to IBM discs using EBCDIC or ASCII
- Copy files from or to IBM discs

Defining a "dummy" subchannel, as mentioned above is a good way to make the software configuration compatible with the IBM flexible disc. The hardware (as was mentioned above also) is smart enough to determine which type of disc is inserted into the drive. This is not true in the case of DD.30, the RTE-L disc driver, which has been written for HP discs. As a result, the unaware user can encounter some trouble when reading an IBM flexible disc. The author discovered, after making some tests, at least two of these problems. Presented below are these two problems and their solutions in hopes that users will find it easy to interface with IBM formatted floppies.

The first problem deals with the definition of the last sector of the disc. When a disc LU is defined, the user specifies the number of sectors per track (30 for HP, 26 for IBM). The sectors are numbered from 0 to 29 for HP formatted discs, and 1 to 26 for IBM format.

When the user tries to read or write one sector, DD.30 checks to see if the sector number that the user specified in his Exec call is valid. To perform the test, DD.30 demands that the specified value be strictly smaller than the number of sectors per track.

For HP discs, all 30 sectors have sector numbers less than 30 since the numbering convention begins at zero. However, since IBM disc sectors are numbered from 1 to 26, the number of the last sector is not less than the number of sectors per track (26). Whenever the user attempts to reference sector 26 the test that DD.30 makes (as described above) fails, the call is rejected, and sector 26 cannot be accessed.

The solution to this problem is a simple one. At generation time, when configuring the IBM disc LU, specify 27 as the number of sectors per track. When programming however, the user must remember that the are physically only 26 sectors per track.

The second problem that could arise when doing I/O to an IBM formatted floppy deals with the buffer length of a read or write.

First, recall some details about the RTE-L Exec call used when accessing a disc LU:

- Bits 6 thru 11 in the control word must be correctly set
- Bit 13 in the control word can be set to allow the user to analyze errors
- The starting sector must be an even number

Another parameter of interest is the buffer length (the number of words to be read or written). Recall that HP flexible discs contain 256 bytes per physical sector (which is not the same as the number of bytes per RTE logical sector). IBM discs contain 128 bytes per sector. Translating bytes into words (divide by two), the HP disc has 128 words per sector, and the IBM format has 64. DD.30 takes the buffer length (in words) that is specified in the Exec call and divides it by 128 to find the number of sectors to read.

If the requested buffer length is 128 words, DD.30 will start a buffered read of 128 words, but the disc controller will discover an end of sector after 64 words when reading from an IBM disc. The driver will report a transmission error, and the program will be aborted. If the user does not want the program to be aborted, bit 13 in the control word must be set.

The solution to this problem is never to use 128 words as the specified buffer length. The user may specify 64 words, or a multiple of 64 words (with the exception of 128). A multiple of 64 words is acceptable because the controller can handle a request in one of two different ways. When performing a buffered read, a single HP sector can be read. Alternatively, when doing an unbuffered read, several consecutive sectors can be read. Therefore in some cases it is permissible to have a read access multiple sectors.

If the user takes note of all these details, and finds an ASCII to EBCDIC and EBCDIC to ASCII conversion routine, it should be a simple task to read or write from/to any sector of the IBM disc.

Some software will still have to be written to access information on the IBM formatted disc as an IBM file, because the user is responsible for for decoding the directory track on cylinder zero.

In conclusion, RTE-L does not support file access to IBM formatted discs, but if the user writes his own software, he will be able to read and write to floppies of this format through DD.30. As an added note, the driver will not allow the user to format IBM discs. During this process some commands must be sent to the controller, a mode of operation which DD.30 does not allow.

*1 Information is drawn from an article by Ed Brummit of HP Data Systems Division which appeared in an internal newsletter earlier this year.

RTE-IVE: NEW EXECUTE-ONLY MEMORY-BASED OPERATING SYSTEM

Mark Beswetherick/HP Data Systems Division

RTE-IVE is a new member of the compatible Real Time Executive (RTE) family of operating systems. RTE-IVE is an execute-only, memory-based subset of the powerful RTE-IVB disc based operating system. RTE-IVE runs on HP 1000 M-, E-,or F-Series computers. RTE-IVE incorporates many of the powerful features of RTE-IVB, including:

- Management of up to 64 multi-user program partitions in up to 2.048 Megabytes of memory.
- Up to 54K bytes per partition for user's program code, independent of physical memory used by the operating system or drivers.
- Transparent user access to extended memory area (EMA) for data, with space limited only by available memory (nearly 2 Megabytes in a 2.048 megabyte system).

RTE-IVE is essentially an RTE-IVB system without a system disc. Those RTE-IVB features that require a system disc are not available in RTE-IVE. Such features include program development and the Session Monitor user interface. Then how does one do RTE-IVE system generation and program development? On a host RTE-IVB system. The RTE-IVE system code can be loaded into the RTE-IVE system via a DS/1000-IV network link, or via minicartridge. Once the RTE-IVE system is booted up, programs may be loaded into memory via a DS/1000-IV network link or from any supported input device on the RTE-IVE system. RTE-IVE system generation and program development procedures are very similar to RTE-IVB procedures. This increases ease of use; the same set of procedures allows users to create both disc-based and memory-based applications.

This combination of features makes RTE-IVE a superior product to its predecessor, RTE-M. RTE-M will be placed in inactive status (removed from the Corporate Price List) in May, 1981. Now is the time to upgrade to RTE-IVE! (See ordering information for upgrade options.)

RTE-IVE is particularly well suited for Distributed Systems environments. RTE-IVE supports all of the features of DS/1000-IV. Combine the powerful program and memory management capabilities of RTE-IVE, the extensive network capabilities of DS/1000-IV, and the processor throughput of an HP 1000 M-, E-, or F-Series computer. The result: a powerful, yet economical, Nodecomputer. The Nodecomputer is economical because it does not require a local hard disc. Other remote peripherals may be shared as well, further reducing the cost. The Nodecomputer's lack of a hard disc provides another benefit: greater environmental versatility. It may be impractical to maintain a hard disc in an environment that has a high concentration of particles in the air, or where significant vibration is present. In many cases a Nodecomputer can withstand such environments. This combination of features make the Nodecomputer particularly well-suited to real-time monitoring and/or control in factory floor environments.

RTE-IVE systems can also function in stand-alone mode without network support. In this case, a disc-based RTE-IVB system must be available for doing program modifications or system generations for the RTE-IVE system, when necessary. In a stand alone environment, RTE-IVE's prime application is to provide an environmentally-rugged system. An example: Performing on-board aircraft testing. Here, environmental motion precludes the use of a disc and a network link is impractical.

Ordering Information

92068E:	$1500	License to generate and execute an RTE-IVE system plus EMA firmware.

Prerequisite — RTE-IVB purchased as a line item or in an HP 1000 system.

Option 001:	–$600	Upgrade discount from RTE-M, no support services.
Option 002:	–$900	Upgrade discount from RTE-M, if on support services.
92068V + Option 001:	$40/month	Extends RTE-IVB CSS to an RTE-IVE system.
92068W + Option 001:	$25/month	Extends RTE-IVB SSS to an RTE-IVE system.

By popular demand, we are making 92068E Option 002 available for 180 days (through May 31, 1981) instead of 90 days as originally stated in the NPT.

Remember the Nodecomputer, and RTE-IVE, for powerful yet economical configurations based on HP M-, E-, and F-Series computers.

JOIN AN HP 1000 USER GROUP!

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

NORTH AMERICAN HP 1000 USER GROUPS

Area	User Group Contact
Boston	LEXUS P.O. Box 1000 Norwood, Mass. 02062
Chicago	David Olson Computer Systems Consultant 1846 W. Eddy St. Chicago, Illinois 60657 (312) 525-0519
Greenville/N. C.	Henry Lucius III American Hoechst Corp. Film Division Box 1400 Greer, South Carolina 29651 (803) 877-8471
New Mexico/El Paso	Guy Gallaway Dynalectron Corporation Radar Backscatter Division P.O. Drawer O Holloman AFB, NM 88330
New York/New Jersey	Paul Miller Corp. Computer Systems 675 Line Road Aberdeen, N.J. 07746 (201) 583-4422
Philadelphia	Dr. Barry Perlman RCA Laboratories P.O. Box 432 Princeton, N.J. 08540
Pittsburgh	Eric Belmont Alliance Research Ctr. 1562 Beeson St. Alliance, Ohio 44601 (216) 821-9110 X417
San Diego	Jim Metts Hewlett-Packard Co. P.O. Box 23333 San Diego, CA 92123

NORTH AMERICAN HP 1000 USER GROUPS (CONTINUED)

Area	User Group Contact
Toronto	Nancy Swartz Grant Hallman Associates 43 Eglinton Av. East Suite 902 Toronto M4P1A2
Washington/Baltimore	Paul Taltavull Hewlett-Packard Co. 2 Choke Cherry Rd. Rockville, MD. 20850
General Electric Co. (GE employees only)	Stu Troop Special Purpose Computer Ctr. General Electric Co. 1285 Boston Ave. Bridgeport, Conn. 06602

OVERSEAS HP 1000 USER GROUPS

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

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. 12/80

Part No. 5951-6111