THE MULTI-TASKER

Volume 15, Number 9 April 1982

The Newsletter of the RSX-11/IAS Special Interest Group

Contributions should be sent to: Editor, The Multi-Tasker, c/o DECUS, One Iron Way, MR2-3/E55, Marlboro, MA 01752

European members should send contributions to: Colin A. Mercer, Tennant Post, High Street, FAREHAM, PO16 7BQ, Hants, England

Members in Australia or New Zealand should send contributions to: Clive Edington, CSIRO, Computing Research 314 Albert St., East Melbourne, VIC 3002, Australia

Letters and articles for publication are requested from members of the SIG. They may include helpful hints, inquiries to other users, reports on SIG business, summaries of SPR's submitted to Digital or other information for the members of RSX-11/IAS SIG.

All contributions should be "camera-ready copy" e.g. sharp black type in a 160x240 mm area (8 1/2" x 11" paper with 1" margins) and should not include xerox copies. If you use RUNOFF to prepare your contribution the following parameters have been found to be satisfactory:

.PAPER SIZE 60,80 .LEFT MARGIN 8 .RIGHT MARGIN 72 .SPACING 1

These parameters assume output on a lineprinter with a pitch of 10 char/inch. Adjust the parameters to maintain the same margins if another pitch is used.

TABLE OF CONTENTS

Columns

From the Editor. 2
From Five Years Ago. 3
New Users. 4
DECUS/RSX SIG Library News 6
 TALK Program Corrections
 Fall 1981 SIG Tape Distribution
 Corrections to ICR Submissions
Hints and Things 20
 Topological Walk to an ODL

Articles

Theory of Interactive Debuggers. 21
The DDT Debugger for PDP-11. 26
Make Use of Secondary Pool Under RSX-11M-Plus. . 35
How the RSX-11M Shuffler Works 37

Special Section

Files-11 On-Disk Structure Specification 1

READ THIS FIRST

A serious error was made by the Multi-Tasker in publishing the article on library compression article (December/January 1982 Multi-Tasker, Vol. 15, No. 6). The code fragment on page 46 as published:

```
.; COPY LIBRARY AND PREALLOCATE DISK BLOCKS
PIP 'F'.'E'/PU
.;
```

should read as follows:

```
.; COPY LIBRARY AND PREALLOCATE DISK BLOCKS
PIP 'F'.'E'/CO/BL:'B'./NV='F'.'E'
.;
.; PURGE COMPRESSED AND BLOCK DEALLOCATED LIBRARY
PIP 'F'.'E'/PU
.;
```

Please pencil in the correction on the original article. Also, since the actual value of the blocks used by the newly compressed library is known by the symbol 'B', arithmetic operations may be performed on 'B' to handle special applications. For example, a user may wish to issue a warning message if the size of the library exceeds some predefined limit.

From the Editor

The technical content of this issue is perhaps the best since I became editor. To start with, Digital has released the Files-11 On-Disk Structure Specification (ODS-1) for publication. There are user papers on debugging theory for PDP-11's, use of secondary pool under RSX-11M-Plus, and an explanation of how the RSX-11M shuffler works.

Also, a new column on DECUS library and RSX SIG tapes debuts in this issue. Anytime you use a library or SIG tape program, drop a note to the Multi-Tasker. There is so much free software available out there, that no single site can keep track of it all. But by combining forces, we can start to get a handle on it.

Finally, about the time you read this, my group will be moving to a new address. Please address any correspondence to DECUS and they will be able to route it to me. Otherwise, there is a good chance the company mail will lose it. My phone number will be the same.

Ralph Stamerjohn
Multi-Tasker Editor

Phone: (314) 694-4252 (3-5 pm, CST)

From Five Years Ago

The December 1976 issue of the Multi-Tasker contained a brief questionnaire requesting system profile information and data on operating system problems. The results appeared in the April 1977 issue. The hardware profile information is summarized below. The memory sizes and amounts of mass storage listed are median values.

IAS (12 responses) - 165KW memory, 120MB mass storage
9 11/70s, 3 11/45s

RSX-11D (108 responses) - 110KW memory, 20MB mass storage
9 11/70s, 60 11/45s, 35 11/40s, 3 11/35s, 1 11/34

RSX-11M, mapped (86 responses) - 56KW memory, 3.75MB mass storage
9 11/70s, 15 11/45s, 45 11/40s, 12 11/34s, 5 11/35s

RSX-11M, unmapped (18 responses) - 24KW memory, 3.75MB mass storage,
3 11/35s, 1 11/34, 5 11/20s, 9 11/10s, 1 11/05

RSX-11S (7 responses) - 24KW memory

Problems

Users were asked to report one problem that he or she would like to see solved. The response summaries are listed below, by operating system.

The only IAS difficulty to be reported more than once was problems with the spooler. Other reported problems were the failure of tasks to get at user defaults, a too frequent need to reboot the system because of problems, time scheduler slow-downs, and the inconvenient means for performing terminal I/O from real-time tasks.

RSX-11D users complained most frequently about memory fragmentation, task size limitations, and adequacy of documentation. Other problems reported by more than one user included tasks getting stuck on the MRL, time-slicer inadequacy, problems involving contiguous disk space, and unexplained system crashes. Dissatisfaction with the accounting package was expressed by two users, while equal numbers complained about slow SPR response time and the inability to use F4P (for floating point operations) on 11/40s.

The 104 RSX-11M users reported 37 separate problems. The most frequent plea was for improved documentation, followed by complaints about system hang ups occurring when pool space becomes exhausted. There were complaints about the lack of shareable tasks, random system crashes, the need for a full duplex terminal driver, and disk fragmentation. One user who reported random crashes admitted they were usually caused by hardware or by software (drivers and/or privileged tasks) that were added to the system. His major plea was for crash dump and crash dump analyzer facilities.

Only two problems were reported for RSX-11S. One user complained about sysgen and TKB when RSX-11S was run from an RSX-11D host. The other report was a request for dynamic partitions for VMR.

Documentation Review

The SIG announced that it would be performing a review of the RSX-11M V3.0 documentation.

RSX-11D/IAS Batch Working Group

Although DEC had announced the stabilization of RSX-11D, the BATCH working group noted that BATCH could probably be changed without changing the exec, and hoped that the following BATCH problem areas would still be improved: error logging(11D), security(11D/IAS), unjustified incompatiblities with interactive mode(11D), lack of conditional return capability(IAS), too restrictive command language(11D). The working group solicited user comments about BATCH problems.

User-Written Utilities Working Group

The re-activation of the User-Written Utilities Working Group was announced. Its charter was expanded to include user maintenance of the unsupported software that Digital was including with RSX/IAS distributions. The working group also planned to address the problems of organizing submissions to the DECUS Program Library.

MULTI-TASKER Subscription Fees

Mark Lewis, the U.S. SIG Coordinator, reported that the DECUS International Liason Committee was considering a plan to charge fees for SIG newsletters that would both recover printing costs and subsidize other DECUS-wide activities. Mark stated his objections to the plan and to the fact that the SIGs had not been consulted on the proposal.

New Users

This column is for the quiet majority of the RSX-11/IAS SIG - new users. We would like to answer your questions, print hints on using RSX, and publish your experiences. The best people to help new users are new users. Help clear up the confusion and send any comments, questions, or contributions to Multi-Tasker - New Users, c/o DECUS, One Iron Way, MR2-3/E55, Marlboro, MA 01752.

New Users Questions

How to You Get the Multi-Tasker?

Q. One old-timer at our site gets the Multi-Tasker, and by the time it gets to me, it is so ragged I can hardly read it. Is there anyway I can get my own copy?

A. You get the Multi-Tasker by joining DECUS and the RSX-11/IAS SIG. The easist thing to do is call the DECUS membership number (617) 467-4168 and ask for a membership kit to be sent to you. When it comes, make sure you check box 17 (RSX-11/IAS SIG). The Multi-Tasker is automatically sent to all members of the SIG.

Symposium Proceedings

Q. With the economy in trouble, money to attend the symposium is very hard to come by. How can I find out what happens at the symposium without actually attending?

A. The Symposium Proceedings for any or all five annual symposia (Europe, Fall U.S., Canada, Spring U.S, or Australia) may be ordered from DECUS. Many other back issues are also available. Typical cost is $15.00 for one particular symposia and $75.00 for all five. Call the DECUS Publications group at (617) 467-4143 for ordering information.

The Proceedings will have all the technical sessions. The Multi-Tasker will try to suplement them with articles and transciptions of the interactive sessions.

Waiting or Stopping?

Q. I am confused about the difference between waiting or stopping for an event flag. How to I choose which version to use in my programs?

A. The choice depends on the answer to the question: "When the event flag is set, must my program react in a timely (real-time) fashion?" If the answer is "yes", use the wait form. Otherwise, the stop form is acceptable.

The crucial difference between waiting or stopping is based on priority and memory. RSX-11M allocates all resources, including memory, based on a task's priority. When a program enters a wait state, it no longer competes for the CPU. But the task continues to compete for memory at its priority. So unless higher-priority tasks cause the program to checkpoint, it remains in memory and when the event flag is set, can immediately begin competing for the CPU.

When a program stops, its effective priority is lowered to zero. Now any task can cause the program to checkpoint. When the event flag is set, the program regains its priority. But it could have been swapped out by lower

priority tasks, so sometimes it cannot immediately begin competing for the CPU. Therefore stopping for an event flag should not be used when a timely response is needed.

DECUS/RSX SIG Library News

Over the years, DECUS, through the DECUS library, and the RSX-11/IAS SIG, through the SIG tapes, have accumulated a huge set of useful software. If you have news about any of this software, please send to the Multi-Tasker c/o this column. This includes any problems discovered, patches to existing software, short notes on library submissions you found useful, or any other information you may have. Send submissions to Multi-Tasker - Library News, c/o DECUS, One Iron Way, MR2-3/E55, Marlboro, MA 10752.

TALK Program Corrections

Bob Turkelson

NASA/Goddard Space Flight Center
Code 935
Greenbelt, Maryland 20771

The terminal emulation and inter-computer file transfer program TALK, found in [352,2] on the Fall 1981 Los Angeles RSX SIG tape, should be modified as described below if you are experiencing any character loss during file transfers. This is most likely to occur at baud rates of 2400 or above. With these changes we have had no trouble running TALK on a PDP-11/70 connected to a VAX-11/780 at 9600 baud.

The SLP correction file printed below includes several other changes. The TALK task is set non-privileged after the connection to the port is established. Previously, any TALK user could have specified an input or output file in any UIC. Also, if a user logs off the terminal after "detaching" from TALK with command D, instead of exiting with command Z, TALK will be aborted by BYE. Previously, since TALK was privileged, the task simply remained in the stopped state. The new version eliminates the extra null records at the beginning and end of a file transferred from a VAX/VMS. Null records are not written to the PDP-11 output file. Additionally, VAX and Sigma 9 files containing the monitor prompt character ($ or !) at the beginning of records, may now be transferred to the PDP-11 using command B. If you do not talk to a VAX or Sigma 9, you may omit all modifications to BUILD.MAC and CONTRL.MAC from the SLP correction file.

Use this procedure to modify TALK:

1. Rename some of the Version 5.09 files as shown:

```
PIP TALKCMD.509/RE=TALK.CMD
PIP *.509/RE=TALK.MAC,BUILD,CNPRT,CONTRL
```

2. Create the file TALK.COR, shown below.

3. SLP @TALK.COR

4. @TALK

   When prompted, specify that the new versions of TALK, BUILD, CNPRT, and CONTRL are to be assembled. If TALK has not been built previously, specify that ALL files should be assembled.

While testing TALK, we found that we could transfer files between a PDP-11/23 and a VAX-11/780 without a direct link between these computers, by running TALK on the 11/23 to connect to a PDP-11/70, and then running TALK on the 11/70 to connect to the VAX. (TALK on the 11/23 was built as if it were communicating directly to a VAX, so that the correct COPY commands would be generated.) In order to exit, TALK must be in command mode and this is entered by typing a control-W on the user's terminal. Typing a control-W on the 11/23 would allow us to exit TALK on the 11/23, but the second TALK would be left active on the 11/70. This demonstrates the need for command C, which sends control-W to the host computer. We originally included command C simply for completeness, in case someone connected to a computer on which control-W had some meaning. Well, in this situation we needed to send a control-W to TALK on the 11/70 in order to be able to exit the TALK program.

The file TALK.COR:

```
TALK.MAC=TALK.509/CS:037014
-,,/; 5.10/
-1,1
; TALK.MAC      17MAR82  SLP UPDATE
-/.IDENT/,.+1
        .IDENT  /VM5.10/
INTRO:: .ASCII  <15><12>/TLK: Version VM5.10  /
-/WT2FLG::/,.
WT2FLG::.BYTE   0                ; NUMBER OF RECS TO IGNORE WHEN A HELP MODE
;                                ;   CMD IS ECHOED
NLRFLG::.BYTE   0                ; FLAG INDICATING IF PREVIOUS PORT RECORD NULL
-/PTINTR:/
-/BICB...#100,@RCSR/,.
-/6$:/
-/BEQ...9$/
        CLRB    PTISEF                  ; CLEAR INDICATOR
-/@#$SETF/
-/MOVB...#1,PTDATA/,.
-/9$:/,.
9$:
/

BUILD.MAC=BUILD.509/CS:051550
-,,/; 5.10/
-1
; BUILD.MAC      17MAR82  SLP UPDATE
-/BUILD::/
-/4$:/+1
```

```
        .IF DF  VAX
        INCB    WT2FLG                      ; ALSO IGNORE EXTRA <CR> ON VAX
        .ENDC
        CLRB    NLRFLG                      ; CLEAR NULL RECORD FLAG
/

CNPRT.MAC=CNPRT.509/CS:164727
-,,/; 5.10/
-1
; CNPRT.MAC      17MAR82  SLP UPDATE
-/.MCALL/,.+2
        .MCALL  CINT$,TCBDF$
;
        TCBDF$                              ; DEFINE TCB OFFSETS
        CINT$                               ; DEFINE CINT$ OFFSETS (C.INVE)
-/CNPRT::/
-/QIOMAC...#CNMSG/
        CALL    $SWSTK,6$                   ; SWITCH TO SYSTEM STATE
        MOV     $TKTCB,R1                   ;; GET TCB ADDRESS
        BIC     #T3.PRV,T.ST3(R1)           ;; SET TASK NON-PRIVILEGED
        RETURN                              ; RETURN FROM SYSTEM STATE
6$:
-/BR...1$/,.
        JMP     1$                          ; REQUEST ANOTHER PORT NUMBER
-/DCNPRT::/,.
DCNPRT::
        CALL    $SWSTK,1$                   ; SWITCH TO SYSTEM STATE
        MOV     $TKTCB,R1                   ;; GET TCB ADDRESS
        BIS     #T3.PRV,T.ST3(R1)           ;; SET TASK PRIVILEGED
        RETURN                              ; RETURN FROM SYSTEM STATE
1$:     DIR$    #DCINTR                     ; DISCONNECT INTERRUPT FOR RECEIVE
/

CONTRL.MAC=CONTRL.509/CS:001120
-,,/; 5.10/
-1
; CONTRL.MAC      17MAR82  SLP UPDATE
-/PRTIN::/
-/CLRB...WT2FLG/,.
        DECB    WT2FLG                      ; IGNORED THIS RECORD - COUNT IT
-/12$:/
-/CMP...R0,#PROMPT/+1
        .IF DF  SIGMA ! VAX
        TSTB    NLRFLG                      ; IF LAST RECORD NOT NULL, ASSUME PROMPT
        BEQ     14$                         ;   CHARACTER WAS IN THE FILE - CONTINUE
        .ENDC
-/20$:/+1,.+1
        .IF DF  SIGMA ! VAX                 ; DO NOT WRITE NULL RECORDS IF HELPING
;                                           ;   AVOIDS NULL REC BEFORE PROMPT CHAR
-/BEQ...25$/
        INCB    NLRFLG                      ; ASSUME A NULL RECORD
-/BEQ...30$/
        CLRB    NLRFLG                      ; INDICATE RECORD NOT NULL
/
```

```
TALK.CMD/-AU=TALKCMD.509/CS:117454
-1,1
.; TALK.CMD  -  17MAR82  SLP UPDATE
-/PIP...TALKPRE.TMP/,.
PIP TALKPRTMP.MAC=TALKSYS'SUF'.MAC,TALKPRE.MAC
-/PIP...TALKPRE.TMP;*/,.
PIP TALKPRTMP.MAC;*/DE
-/MAC...'FILE'/,.
.SETF    RSXMC
.IF      FILE  EQ  "CNPRT"  .SETT  RSXMC
.IFT     RSXMC   PIP TALKMCTMP.MAC=LB:[11,10]RSXMC.MAC,SY:'<UIC>'TALKPRTMP.MAC
.SETS    PREFIL  "TALKPRTMP"
.IFT     RSXMC   .SETS  PREFIL  "TALKMCTMP"
MAC 'OUTFIL''LST'=LB:[1,1]EXEMC/ML,SY:'<UIC>''PREFIL','FILE'
.IFT     RSXMC   PIP TALKMCTMP.MAC;*/DE
/
```

Fall 1981 SIG Tape Distribution

Jim Neeland

RSX-11/IAS SIG Tape Coordinator
Hughes Research Labs
3011 Malibu Canyon Road
Malibu, California 90265

The RSX-11/IAS SIG Tape collections from the Fall 1981 Los Angeles Symposium is now in distribution to Local User Groups through the SIG Tape Copy Tree. Also, a copy is being place into the DECUS library that anyone can order.

The programs on this tape are from user submissions. The DECUS staff, RSX/IAS SIG staff, and Digital are all in relative ignorance of the contents of the tapes. No warranty of any kind is implied in the distribution of the tape. The programs may or may not be well documented, they may or may not work, they may even crash your system. If you have a problem with the content of the tapes, contact the author of the particular program. Do not contact DECUS, Digital, or the RSX-11/IAS SIG.

The tape contains about 36,000 blocks of software in 1900 files. Since this will fit on a single 2400 foot, 800 BPI BRU tape, it will be distributed as such. It is in a RK07 image, the smallest Digital disk it will fit on.

The UIC [300,1] contains several files of interest. The file RSXF81.DIR contains a directory of the tape. The file RSXF81TPE.DOC contains an abstract of the contents of the tape by UIC. The file README.ALL contains a concatenated list of all the README files on the collection. The file UICSETF81.CMD contains UFD commands to create all the needed UIC's on device XX:. Edit it to match your needs before using BRU to extract the tape contents. Note, that a partial extraction can be achieved by only creating the desired UIC's.

The file [300,1]SUBMIT.DOC contains the guidelines for submissions to the RSX/IAS SIG Tape collection. This is must reading for everyone who desires to submit a program to the SIG tapes. The more people that follow these guidelines, the faster we can turn a tape around and distributed it.

The UIC account [300,2] contains the program that is used to copy this and other tapes, BIGTPC. This is a new version of TPC with various new features. See the .DOC file, also in [300,2], for further information. The source for this version has been supplied courtesy of Glenn Everhart, UIC [312,315] on this tape. To use BIGTPC to make copies of this tape, one needs a disk with at least 38,000 blocks of free space, not necessarily contiguous.

The distribution is being made through the SIG tree-structured distribution system. A geographically oriented distribution scheme is used, where each LUG makes a few copies of the tape and sends them onto other LUGs, and so on. The is a volunteer operation and DECUS is NOT paying for the postage or free copies of the magnetic tapes. So it will take some time for the distribution to filter down to every LUG.

The tree itself, for readability and clarity, has been split into three parts. First is an errata sheet for changes in contacts names, address, or phone numbers. Always consult this sheet before contacting someone. With the number of people involved, things are always changing. Next is the tree with all of its branches in three pages: EASTCOAST, MIDWEST, AND WESTCOAST. Find you LUG on the appropriate sheet. From this you can learn the ZIP (and LUG and contact name). This indexes into the final part, which is a complete list of all participants with full addresses.

Good luck! I hope there are no glaring oversights in this undertaking, but I have discovered the hard way mow much effort it takes to create, verify, document, and distribute the SIG tape and tree. My congratulations to Phil Cannon for managing to do it for as long as he did. The following people burned the midnight oil to create the 1981 Fall RSX/IAS SIG Tape: Ken Radford, Steve Lazarus, Phil Cannon, Glen Everhart, Bob Denny, and myself.

From the Editor

Following is the tree for the Fall 1981 SIG Tape. To get your copy, find your Local User Group and contact them. They are probably the contacts for other SIG tapes. Also, RSXF81TPE.DOC is reprinted so you have a brief idea of what is on the tape.

Linda A. Slawson
Puerto Rico NUG (chang of LUG name)
RCA Service Co.
P.O. Box 3935 USNS
FPO Miami, FL 34051
(809) 865-7314

Jim Barnes
Dayton LUG
AFWAL/AAAN-3
WP Air Force Base, OH 45433 (change of zip)
(513) 255-6843

Jerry Wray (1600 bpi only)
Central Illinois LUG
University of Illinois
457 Loomis Laboratory of Physics
1110 W. Green St.
Urbana, IL 61801
(217) 333-4922

Peter Reinecke
Greater Houston Area LUG
Texas Instruments
11510 Scottsdale Ct. (change of street)
Stafford, TX 77477
(713) 490-3691

Doug Gladden
Southern New Mexico LUG
White Sands Missile Range
NR-AD-A
WSMR, NM 88002 (change of city abbr.)
(505) 678-3348

Allan Leslie Van Lahn
Lawrence Livermore Natl. Lab LUG
Lawrence Livermore Natl. Lab (change of company name)
L-233 (Shipping address: 7000 East Ave.)
P.O. Box 808 (Bldg. 151 Room 2323)
Livermore, CA 94550
(415) 422-6652

James Fine
Tulsa LUG
9426 E. 26th Place
Tulsa, OK 74129
(918) 627-6498 (off ph: (918) 665-4477)

Martin A. Booker (1600 bpi only)
Southeastern Wisconsin LUG
Milwaukee School of Engineering
1025 N. Milwaukee
Milwaukee, WI 53201
(414) 277-7231 (corrected phone #)

Victor Johansen (replacement for Douglas Brown)
Arizona LUG
ADR Ultrasound (new addr, etc.)
734 W. Alameda Dr.
Tempe, AZ 85282
(602) 963-7401

Larry Olin Horn
Chimneyville LUG
Millsaps College
Computer Services
1701 S. State St.
Jackson, MS 39210
(601) 354-5201 Ext. 236 (add ext.)

William Patterson (1600 bpi only)
Madison Wisconsin LUG
U. of Wisconsin Medical School
Div. of Neurosurgery
600 Highland Ave.
Madison, WI 53792
(608) 263-5227

Steve Hansen
Kansas City LUG
University of Missouri
Computing Services
5100 Rockhill Rd.
Kansas City, MO 64110
(816) 276-1181 (change of phone #)

Paul Tompkins (change of person, address)
Brazosport LUG
Dow Chemical USA, O.C.D.
P.O. Box BB
Freeport, TX 77541
(713) 233-9004

Richard A. Baldwin
San Diego Commercial LUG
North County Comp. Svc.
2235 Meyers Ave. (change of street)
Escondido, CA 92025
(714) 745-6006

Bill Bagley (1600 bpi only)
Portland Area PDP-11 LUG
Tektronix
MS 56-037
P.O. Box 500
Beaverton, OR 97077
(503) 642-8936

Warren B. Weintraub (1600 bpi only)
Chicago Area Commrc. Users Group
3500 Bayside Drive, Apt. 6
Palatine, IL 60067
(312) 520-3245 (new phone #)

Mark Paulk (replacement for Bill Welch)
North Alabama LUG
System Development Corp. (new addr, etc.)
4810 Bradford Blvd. NW
Huntsville, AL 35805
(205) 837-7610

```
                    **********  EAST COAST LUG TREE  **********

                                                                                 Brian Hughes
                                                                              |--MIT PDP-11 LUG
                                                                              |  Cambridge, MA  02139
Dr. L. Michael Frazer                                  John Guidi             |
Washington Area LUG----------|---------------------------Maine PDP-11 LUG ----|  Thomas A. Viana
Bethesda, MD  20814          |                         Bar Harbor, ME  04609  |--Naval Underwater Systems LUG
            ^^               |                                                |  Newport, RI  02840
            ||               |                                                |
            ||               |                                                |  Douglas Bickford
------------||-------------| |                                                |--Vermont Install. & Assoc.  LUG
James K. Neeland           | |                                                |  Burlington, VT  05405-0125
RSX/IAS SIG Tape Coordinator| |                                               |
Malibu, CA  90265          | |                                                |  Donald E. Merusi
---------------------------| |                                                |--Connecticut Valley LUG
                             |                                                   Rockville, CT  06066
                             |                                                   Carl Friedberg
                             |                        |--------------------------New York Metro LUG
                             |                        |                          New York, NY  10038
                             |                        |
                             |                        |                          Mary Anne Feerick
                             |                        |                       |--New York Education LUG
                             |                        |                       |  Bronx, NY  10471
                             |                        |                       |
                             |  Glenn Everhart         |  Richard Marisa       |  Alfred H. Scholldorf
                             |--Cherry Hill, NJ  08358 ---|--Greater Rochester LUG---|--Long Island LUG
                             |                        |  Rochester, NY  14627 |  Stony Brook, NY  11794
                             |                        |                       |
                             |                        |                       |  Edward F. Beadel, Jr.
                             |                        |                       |--Lakeshore LUG
                             |                        |                          Oswego, NY  13126
                             |                        |                          Rick Cochran
                             |                        |--------------------------Ithaca Minicomputer LUG
                             |                                                   Ithaca, NY  14853
                             |                                                   John F. Stitzinger
                             |                                                |--Penn State LUG
                             |                                                |  State College, PA  16801
                             |                                                |
                             |                         Tom Hunter             |  Robert F. Curley
                             |---------------------------Pittsburgh Area LUG--|--Delaware Valley IAS LUG
                             |                         Pittsburgh, PA  15236  |  Flourtown, PA  19031-0322
                             |                                                |
                             |                                                |  R. E. Grandle
                             |                                                |--Tidewater LUG
                             |                                                   Hampton, VA  23665
                             |                                                   Janet Anderson
                             |                                                |--Research Triangle Park LUG
                             |                                                |  Chapel Hill, NC  27514
                             |                                                |
                             |                         Le Huu Nguyen          |  James C. Boyt
                             |---------------------------Florida PDP-11 LUG---|--Atlanta LUG
                                                       Gainesville, FL  32611 |  Atlanta, GA  30340
                                                                              |
                                                                              |  Linda A. Slawson
                                                                              |--Puerto Rico LUG
                                                                                 FPO Miami, FL  34051
```

********** WEST COAST LUG TREE **********

```
Julie Cibelli                                                                                      Jose R. Cen-Zubieta
DECUS USA--------------------------------------------------------------------------------------|--DECUS Mexico LUG
Marlboro, MA  01752                                                                             |  Mexico 20 DF, MEXICO
             ^^                                                                                 |
             ||                                                                                 |  Carlos Mario Hugueney
             ||                                                                                 |--Brazil RSX/IAS LUG
-------------||----------------|                                                                |  Campinas SP 13100, BRAZIL
James K. Neeland               |                                   Bradford A. Lubell              Mark Bartelt
RSX/IAS SIG Tape Coordinator--|-----------------------------------UCLA Biomedical LUG----------|--Caltech/JPL  LUG
Southern Calif. LUG (RSX/IAS)  |                                   Los Angeles, CA  90024       |  Pasadena, CA  91125
Malibu, CA  90265              |                                                                |
-------------||----------------|                                                                |  Michael N. Levine
             ||                                                                                 |--China Lake LUG
             ||                                                                                    China Lake, CA  93555
             ||                                                                                    Ronald L. Webster
             ||                                                                                 |--Phoenix LUG
             vv                                                                                 |  Tempe, AZ  85287
Teri Wise                                                          Douglas Brown                |  Sam Westmoreland
Bay Area RSX/IAS LUG-----------|-----------------------------------Arizona LUG------------------|--Tucson LUG
Palo Alto, CA  94303           |                                   Phoenix, AZ  85027           |  Tucson, AZ  85713
                               |                                                                |  James F. Harrison
                               |                                                                |--Los Alamos Users of RSX
                               |                                                                |  Los Alamos, NM  87545
                               |                                                                |
                               |                                                                |  Doug Gladden
                               |                                                                |--Southern New Mexico LUG
                               |                                                                   WHMR, NM  88002
                               |                                                                   Jim Sagamang
                               |                                                                |--Southern California RSX LUG
                               |                                                                |  Corona, CA  91720
                               |                                   Edward H. Mueller            |  Richard A. Baldwin
                               |-----------------------------------San Diego PDP-11/VAX  LUG----|--San Diego Commercial LUG
                               |                                   Rancho Bernardo, CA  92127   |  Escondido, CA  92025
                               |                                                                |
                               |                                                                |  Dr. Sidney Karin
                               |                                                                |--Baja (10/20) LUG
                               |                                                                   San Diego, CA  92138
                               |                                   Allan Leslie Van Lehn           Robert Walraven
                               |-----------------------------------Lawrence Livermore Natl. Lab LUG-U. C. Davis LUG
                               |                                   Livermore, CA  94550            Davis, CA  95616
                               |                                   Max W. Starr                    Dr. Donald L. Mickey
                               |-----------------------------------Hilo LUG------------------------Maui PDP-11 LUG
                               |                                   Honolulu, HI  96848             Kula, HI  96790
                               |  Raymond French                   Bill Bagley                     Ron Tenison
                               |--Seattle Area LUG-------------|--Portland Area PDP-11 LUG------|--Northwest LSI Educational LUG
                                  Seattle, WA  98124            |  Beaverton, OR  97077         |  Portland, OR  97225
                                                                |                                |
                                                                |                                |  Sheldon Clem
                                                                |                                |--Pacific LUG
                                                                |                                   Corvallis, OR  97330
                                                                |                                   Roger Engleman
                                                                |-----------------------------------Idaho LUG
                                                                                                    Boise, ID  83705
```

```
                                    ***********  MIDWEST LUG TREE  **********

                                                                                                  Bill Welch
                                                                                                |--North Alabama LUG
                                                                                                |  Huntsville, AL  35805
                                                                                                |
Roger S. Miles                  Robert W. Hayes                                                 |  Ray Stiles
Chicago Area Real Time Society|--East Tennessee PDP-11 LUG-----|--------------------------------|--Tennessee PDP-11/VAX Users Group
Hoffman Estates, IL  60195    |  Oak Ridge, TN  37830          |                                |  Gallatin, TN  37066
               ^^             |                                |                                |
               ||             |                                |                                |  John K. Doyle, Jr.
               ||             |                                |                                |--Kentucky PDP-11 LUG
               ||             |                                |                                   Louisville, KY  40232
---------------||--------------|                                |
James K. Neeland               |                                |  John W. Nunnally                 Larry Olin Horn
RSX/IAS SIG Tape Coordinator   |                                |--Arkansas LUG----------------------Chimneyville LUG
Malibu, CA  90265              |                                |  Searcy, AR  72143                 Jackson, MS  39210
-------------------------------|                                |
                              |  Jim Downward                     H. E. Chadwick                    Dean Goranson
                              |--Southeastern Michigan LUG-----|--Central Ohio LUG---------------|--Tri-State LUG
                              |  Ann Arbor, MI  48106          |  Columbus, OH  43213           |  Cincinnati, OH  45227
                              |                                |                                 |
                              |                                |                                 |  Jim Barnes
                              |                                |                                 |--Dayton LUG
                              |                                |                                    WP Air Force Base, OH  45434
                              |                                |
                              |                                |  Douglas W. Fair                   Warren H. March
                              |                                |--Western Reserve LUG---------------West Michigan LUG
                              |                                   Berea, OH  44017                  Grand Rapids, MI  49503
                              |
                              |  Warren B. Weintraub              William Patterson                 Martin A. Booker
                              |--Chicago Area Commrc. Users Group-Madison Wisconsin LUG----------|--Southeastern Wisconsin LUG
                              |  Palatine, IL  60067           |  Madison, WI  53792             |  Milwaukee, WI  53201
                              |                                |                                 |
                              |                                |                                 |  Randall Brown
                              |                                |                                 |--Southwest Minnesota LUG
                              |                                |                                    Rochester, MN  55905
                              |                                |
                              |                                |  Jerry Wray                        Naren T. Sanghvi
                              |                                |--Central Illinois LUG--------------Indiana PDP-11 LUG
                              |                                   Urbana, IL  61801                 Indianapolis, IN  46223
                              |
                              |                                   Dennis V. Jensen                  Robert A. Horick
                              |                                |--Skunk River Small Systems LUG-----Bi-State LUG
                              |                                |  Ames, IA  50011                   Cedar Rapids, IA  52402
                              |                                |
                              |  Richard F. Wrenn              |  Steve Hansen                      Robert B. Mack
                              |--St. Louis LUG-----------------|--Kansas City LUG-------------------Midlands LUG
                              |  St. Louis, MO  63110          |  Kansas City, MO  64110            Omaha, NE  68147
                              |                                |
                              |                                |  Thomas Barnum                     James Fine
                              |                                |--Central Oklahoma LUG--------------Tulsa LUG
                              |                                   Oklahoma City, OK  73105          Tulsa, OK  74129
                              |
                              |  John Jenkinson                   Douglas H. Threatt                Peter Reinecke
                              |--The North Texas LUG-----------|--Alamo PDP-3/PDP-11 LUG---------|--Greater Houston Area LUG
                                 Carrollton, TX  75006         |  San Antonio, TX  78235         |  Stafford, TX  77477
                                                               |                                 |
                                                               |                                 |  Robert W. Hutchinson
                                                               |                                 |--Brazosport LUG
                                                               |                                    Freeport, TX  77541
                                                               |
                                                               |  Margaret H. Knox                  Paul Painter
                                                               |--Austin Minicomputer LUG--------|--Texas A & M  LUG
                                                                  Austin, TX  78712              |  College Station, TX  77843
                                                                                                 |
                                                                                                 |  R. R. Rodriguez
                                                                                                 |--LOTA  LUG
                                                                                                    San Marcos, TX  78666
```

*********** RSX/IAS SIG TAPE PARTICIPATING LUGS - SORTED BY ZIP ***********

Carlos Mario Hugueney
Brazil RSX/IAS Local Users Group
Telebras
C.P. 1579
Campinas SP 13100
BRAZIL

Jose R. Cen-Zubieta
DECUS Mexico LUG
El Colegio de Mexico
Unidad de Computo
Camino al Ajusco 20
Mexico 20 DF, MEXICO
(568) 60--33 Ext. 393

Julie Cibelli
DECUS USA
MR2-3/E55
One Iron Way
Marlboro, MA 01752

Brian Hughes
MIT PDP-11 LUG
HOS, Inc.
121 Magazine St.
Cambridge, MA 02139
(617) 661-5851

Thomas A. Viana
Naval Underwater Systems LUG
Naval Underwater Systems Ctr.
Code 3511, Bldg. 1117-1
Newport, RI 02840
(401) 841-3354

John Guidi
Maine PDP-11 LUG
The Jackson Laboratory
The Computing Service
Otter Creek Rd.
Bar Harbor, ME 04609
(207) 288-3371 Ext. 306

Douglas Bickford
Vermont Install. & Assoc. LUG
University of Vermont
Academic Computing Center
Cook Physical Science Building
Burlington, VT 05405-0125
(802) 656-3190

Donald E. Merusi
Connecticut Valley LUG
171 South St., #46
Rockville, CT 06066
(203) 565-5444

Glenn Everhart
RCA Government Systems Div.
Mail Stop 206-1
Route 38
Cherry Hill, NJ 08358
(609) 338-6022

Carl Friedberg
New York Metro LUG
In House Systems
165 William St.
New York, NY 10038
(212) 233-5470

Mary Anne Feerick
New York Education LUG
Riverdale Country School
5250 Fieldston Rd.
Bronx, NY 10471
(212) 549-8044

Alfred H. Scholldorf
Long Island LUG
SUNY at Stonybrook
Physics Dept.
Stony Brook, NY 11794
(516) 246-7110

Edward F. Beadel, Jr.
Lakeshore LUG
SUNY at Oswego
Chemistry Dept.
Oswego, NY 13126
(315) 341-2340

Richard Marisa
Greater Rochester LUG
University of Rochester
Production Automation Project
River Campus, Hopeman 409
Rochester, NY 14627
(716) 275-5342

Rick Cochran
Ithaca Minicomputer LUG
Cornell University
Theoretical & Applied Mechanics
301 Thurston Hall
Ithaca, NY 14853
(607) 256-7344

Tom Hunter
Pittspurgh Area LUG
U.S. Dept. of Energy
58-M 209
P.O. Box 10940
Pittsburgh, PA 15236
(412) 675-6006

John F. Stitzinger
Penn State LUG
HRB Singer
P.O. Box 60
300 Science Park Rd.
State College, PA 16801
(814) 238-4311

Robert F. Curley
Delaware Valley IAS LUG
University of Pennsylvania
P.O. Box 322
Flourtown, PA 19031-0322
(215) 662-3083

Dr. L. Michael Frazer
Washington Area LUG
Armed Forces Radiobiol. Res Inst
Computer Science Dept.
Bethesda, MD 20814
(202) 295-1372

R. E. Grandle
Tidewater LUG
NASA
MS 461
Langley Research Ctr.
Hampton, VA 23665
(804) 827-2645

Janet Anderson
Research Triangle Park LUG
Rockwell International
800 Eastowne Dr., Suite 200
Chapel Hill, NC 27514
(919) 493-2471

James C. Boyt
Atlanta LUG
Scientific-Atlanta Inc.
3845 Pleasantdale Rd.
Atlanta, GA 30340
(404) 449-2300

Le Huu Nguyen
Florida PDP-11 LUG
University of Florida
Center for Instr. & Res. Comp.
411 Weil Hall
Gainesville, FL 32611
(904) 392-0906

Linda A. Slawson
Puerto Rico LUG
RCA Service Co.
P.O. Box 3935 USNS
FPO Miami, FL 34051
(809) 865-7314

Bill Welch
North Alabama LUG
General Digital Industries
500 Wynn Drive, Suite 504
Huntsville, AL 35805
(205) 837-8305

Ray Stiles
Tennessee PDP-11/VAX Users Group
Volunteer State Community College
Nashville Pike
Gallatin, TN 37066
(615) 452-8600 Ext. 317

Robert W. Hayes
East Tennessee PDP-11 LUG
Oak Ridge National Laboratory
Bldg. 3500
P.O. Box X
Oak Ridge, TN 37830
(615) 574-5726

Larry Olin Horn
Chimneyville LUG
Millsaps College
Computer Services
1701 S. State St.
Jackson, MS 39210
(601) 354-5201

John K. Doyle, Jr.
Kentucky PDP-11 LUG
The Federal Land Bank
P.O. Box 32390
Louisville, KY 40232
(502) 566-7164

H. E. Chadwick
Central Ohio LUG
Western Electric Co., Inc.
Dept. 42330
6200 E. Broad St.
Columbus, OH 43213
(614) 860-2093

Douglas W. Fair
Western Reserve LUG
Ohio Turnpike Commission
682 Prospect St.
Berea, OH 44017
(216) 234-2081 Ext. 310

Dean Goranson
Tri-State LUG
Cincinnati Gear Co.
5657 Wooster Pike
Cincinnati, OH 45227
(513) 271-7700

Jim Barnes
Dayton LUG
AFWAL/AAAN-3
WP Air Force Base, OH 45434
(513) 255-6843

Naren T. Sanghvi
Indiana PDP-11 LUG
University Hospital
U.H. A-32
1100 W. Michigan St.
Indianapolis, IN 46223
(317) 264-4618

Jim Downward
Southeastern Michigan LUG
KMS Fusion Inc.
P.O. Box 1567
3621 S. State St.
Ann Arbor, MI 48106
(313) 769-8500

Warren H. March
West Michigan LUG
H. H. Cutler Co.
120 Ionia S W
Grand Rapids, MI 49503
(616) 459-9101

Dennis V. Jensen
Skunk River Small Systems LUG
Ames Laboratory ISU/USDOE
310 Metallurgy
Ames, IA 50011
(515) 294-4823

Robert A. Horick
Bi-State LUG
CMC Colleges Associated
1220 First Avenue NE
Cedar Rapids, IA 52402
(319) 399-8560

Martin A. Booker
Southeastern Wisconsin LUG
Milwaukee School of Engineering
1025 N. Milwaukee
Milwaukee, WI 53201
(414) 277-2781

William Patterson
Madison Wisconsin LUG
U. of Wisconsin Medical School
Div. of Neurosurgery
600 Highland Ave.
Madison, WI 53792
(608) 263-5227

********** RSX/IAS SIG TAPE PARTICIPATING LUGS - SORTED BY ZIP **********

Randall Brown
Southwest Minnesota LUG
Mayo Foundation
Section of Engineering
200 First Street SW
Rochester, MN 55905
(507) 284-2539

Warren B. Weintraub
Chicago Area Commrc. Users Group
3500 Bayside Drive, Apt. 6
Palatine, IL 60067
(312) 640-4530

Rodger S. Miles
Chicago Area Real Time Society
Telemed Cardio-Pulmonary Systems
2345 Pembroke Ave.
Hoffman Estates, IL 60195
(312) 884-5900

Jerry Wray
Central Illinois LUG
University of Illinois
487 Loomis Laboratory of Physics
1110 W. Green St.
Urbana, IL 61801
(217) 333-4922

Richard F. Wrenn
St. Louis LUG
Washington Univ. School of Med.
Dept. of Biological Chemistry
660 S. Euclid Ave.
St. Louis, MO 63110
(314) 454-2179

Steve Hansen
Kansas City LUG
University of Missouri
Computing Services
5100 Rockhill Rd.
Kansas City, MO 64110
(816) 276-1583

Robert B. Mack
Midlands LUG
Informatics, Inc.
2806 Emeline St.
Omaha, NE 68147
(402) 291-8300

John W. Nunnally
Arkansas LUG
Harding University
Box 890, Station A
Searcy, AR 72143
(501) 268-6161 Ext. 440

Thomas Barnum
Central Oklahoma LUG
Oklahoma Real Estate Commission
4040 N. Lincoln, Suite 100
Oklahoma City, OK 73105
(405) 521-2137

James Fine
Tulsa LUG
9426 E. 26th Place
Tulsa, OK 74129
(918) 627-6498

John Jenkinson
The North Texas LUG
Mostek Corp.
MS 32
1215 W. Crosby Rd.
Carrollton, TX 75006
(214) 323-6401

Peter Reinecke
Greater Houston Area LUG
Texas Instruments
4000 Greenbriar
Stafford, TX 77477
(713) 490-3691

Robert W. Hutchinson
Brazosport LUG
Dow Chemical U.S.A.
Texas Division
A P Beutel Building
Freeport, TX 77541
(713) 233-1737

Paul Painter
Texas A & M LUG
Texas A & M University
Dept. of Electrical Engineering
College Station, TX 77843
(713) 845-7530

Douglas H. Threatt
Alamo PDP-8/PDP-11 LUG
School of Aerospace Medicine
SAM/BRS
Brooks Air Force Base
San Antonio, TX 78235
(512) 536-3886

R. R. Rodriguez
LOTA LUG
Southwest Texas State University
Computer Services
San Marcos, TX 78666
(512) 245-2501

Margaret H. Knox
Austin Minicomputer LUG
University of Texas
Computation Center
Austin, TX 78712
(512) 471-3241

Roger Engleman
Idaho LUG
USDA ARS
1175 S. Orchard, Suite 116
Boise, ID 83705
(208) 334-1363

Douglas Brown
Arizona LUG
GTE Automatic Electric Labs
2500 Utopia Rd.
Phoenix, AZ 85027
(602) 582-7570

Ronald L. Webster
Phoenix LUG
Arizona State University
ECA 108
Computer Services
Tempe, AZ 85287
(602) 965-1203

Sam Westmoreland
Tucson LUG
Cholla High School
2001 W. 22nd Street
Tucson, AZ 85713
(602) 791-6789

James F. Harrison
Los Alamos Users of RSX
Los Alamos National Lab
MP-1, MS 828
P.O. Box 1663
Los Alamos, NM 87545
(505) 667-5688

Doug Gladden
Southern New Mexico LUG
White Sands Missile Range
NR-AD-A
WHMR, NM 88002
(505) 678-3348

Bradford A. Lubell
UCLA Biomedical LUG
L.A. Cardiovascular Research Lab
A3-381 CHS
UCLA
Los Angeles, CA 90024
(213) 825-6713

James K. Neeland
Southern Calif. LUG (RSX/IAS)
Hughes Research Labs
3011 Malibu Canyon Rd.
Malibu, CA 90265
(213) 456-6411 Ext. 333

Mark Bartelt
Caltech/JPL LUG
Calif. Institute of Technology
MS 356-48
1201 E. California Blvd.
Pasadena, CA 91125
(213) 356-6663

Jim Sagamang
Southern California RSX LUG
Fleet Analysis Center
CODE 84C1
Corona, CA 91720
(714) 736-4632

Richard A. Baldwin
San Diego Commercial LUG
North County Comp. Svc.
2235 Myers Ave.
Escondido, CA 92025
(714) 745-6006

Edward H. Mueller
San Diego PDP-11/VAX LUG
Oak Industries
16935 W. Bernardo Dr.
Rancho Bernardo, CA 92127
(714) 485-9300

Dr. Sidney Karin
Baja (10/20) LUG
General Atomic Co.
P.O. Box 81608
San Diego, CA 92138
(714) 455-4474

Michael N. Levine
China Lake LUG
Naval Weapons Center
Code 3513
China Lake, CA 93555
(714) 939-2417

Teri Wise
Bay Area RSX/IAS LUG
Ford Aerospace
MS X-90
3939 Fabian Way
Palo Alto, CA 94303
(415) 494-7400 Ext. 5015

Allan Leslie Van Lehn
Lawrence Livermore Natl. Lab LUG
Lawrence Livermore Labs
L-233
P.O. Box 808
Livermore, CA 94550
(415) 422-6652

Robert Walraven
U. C. Davis LUG
University of California Davis
Applied Science
Davis, CA 95616
(916) 752-0360

Dr. Donald L. Mickey
Maui PDP-11 LUG
University of Hawaii
Institute for Astronomy
P.O. Box 209
Kula, HI 96790
(808) 244-5565

Max W. Starr
Hilo LUG
East-West Center
Finance & Management Systems
1777 East-West Rd.
Honolulu, HI 96848
(808) 944-7970

Bill Bagley
Portland Area PDP-11 LUG
Tektronix
MS 56-037
P.O. Box 500
Beaverton, OR 97077
(503) 642-8936

Ron Tenison
Northwest LSI Educational LUG
Catlin Gabel School
8825 S.W. Barnes Rd.
Portland, OR 97225
(503) 297-1894 Ext.

Sheldon Clem
Pacific LUG
CH2M HILL
1600 SW Western
Corvallis, OR 97330
(503) 752-4271

Raymond French
Seattle Area LUG
Boeing Commercial Airplane Co.
MS 9W-31
P.O. Box 3707
Seattle, WA 98124
(206) 237-6192

RSXF81TPE.DOC

Brief description of tape contents by directory.

[005,005]	C Runtime I/O library fixes for Structured Languages SIG Spring 1981 tape
[300,001]	General information about this tape
[300,101]	RUNOFF fixes and enhancements (for which version??)
[300,102]	Mods to TECO V36 - default directory for EI, 11D/IAS fixes, etc.
[300,111]	Copy of Fall 81 DECUS paper on RSX-11M System operations using SIG tapes
[300,112]	Fixes for WHO from Chicago Spring 80 SIG tape [307,20]
[300,113]	C File utilities, DIR, OD (Dump), and GREP
[300,120]	VS: driver for intertask communications
[300,121]	Fortran-callable routines for VT100, VT105 features
[300,123]	Multi-Tasker Articles on non-standard AST's, an error-logger task
[300,125]	PARSIZ - shrink any partition (e.g. GEN) on a running system
[300,126]	Archive system for moving files to/from tape automatically
[300,130]	Pinochle (in PASCAL)
[300,131]	MTREK - multi-player Startrek with robot ships, etc.
[300,132]	FCB list for a volume, LUT display, Receive Queue list
[301,062]	VAL - Fortran terminal I/O w/ defaults, range-checking, etc.
[301,063]	CLONE - Multi-user interpretive command language
[301,064]	ERN - Error-logger current error count display for M, M+
[302,212]	Fortran Symbolic Debugger - main program stub and 2nd debugger task
[305,302]	RUNOFF (Standard) + better hyphenation and Dill's TEXT support
[307,020]	M+ Multiuser F11ACP, fast Fortran block I/O, Checkpoint space contents, Versatec M+ driver, STTY - set many terminal charac.
[307,022]	Disk Disaster Recovery programs and documentation
[307,036]	Home directory for privilege users, M Multiuser F11ACP, FCSRES command files for utilities
[307,100]	Virtual Disk (VD:) for M or M+, DECUS CALC for EIS only
[307,101]	Starfleet - Startrek w/ performance records, mail, etc.
[307,105]	F11ACP for Dual-ported disks w/ two processors!
[31[illegible],131]	Slides from RSX11M Device Commons paper - Fall 81 DECUS
[312,315]	DISASM - disassembler for task images, CAM another one from Amsterdam DECUS tape, BASH - allows task to have previous mode be kernel (sneaky!), CSI parser skeleton, DDT22 - a symbolic macro debugger w/ separate main and debugger tasks, IBM to PDP-11 floating-point conversion, update to XMITR from Spring submission, FPEM - latest floating-point emulator, DISOWN + TSKREN to transfer task ownership to CO:, new FFL (fast FLX), new multicolumn lister
[312,316]	PLOTA - subs for histograms, etc. on HP 4/8 pen plotters
[312,317]	TREAD/TWRITE to handle IBM RECFM=FB labelled/unlabled tapes
[314,001]	RATFOR from Structured Language Working Group of RSX SIG
[315,100]	Add bad blks to [0,0]BADBLK.SYS, find files modified after some time, modify task lun assigns w/o rebuild, ASN capability for IAS, find file which has given LBN, cancel all copies of IAS multiuser task, block-mode file compare, faster-than-PIP copy, task dump ala CDA, disk-space/UIC, delete by FID w/ bad headers, IAS device info, reconstruct locked files, new FRG, enhanced GREP, graceful exit if I/O rundown fail (IAS), MCM/ MCX to switch to real MCR/DCL terminal on IAS, a file dump utility, fast magtape ops w/ multibuffering, CRT bargraph display of IAS system, tape copy utilities, IAS task timer, translate RT tapes to RSX files, file undelete, etc, etc.
[315,111]	Triangle LUG RUNOFF, supports INCLUDE files, etc.

[330,001]	File lister/scanner w/ string searches, wild-chararacter file names
[330,002]	F4P Symbolic Debugger
[330,003]	Compile only newly changed modules and insert them in .OLB
[330,004]	Generate command lines from SRD output
[330,005]	Enhanced SRD with: /RVision-date,/OWner, etc.
[330,006]	TAPE read/write utility for various foreign tape formats
[330,010]	Burst concatenated FORTRAN subroutines into individual modules
[330,011]	Resequence F4P source programs
[330,012]	Truncate only those files needing truncation, don't touch others
[330,013]	IAS program to search directories for file to XEQ (RUN)
[330,014]	(IAS) restrict game-playing hours, run task on NL:, schedule programs
[330,015]	RATFIV V2, enhanced RATFOR - FORTRAN pre-processor language
[330,016]	Multitrek - in RATFIV, support for different terminal types
[332,060]	Enhanced version of Jim Downward's CCL
[332,100]	DSC tape directory, selective restore, tape format information
[336,300]	RSX Network Mail
[337,030]	SFGL70 - latest version of Tektronix graphics subroutines
[343,001]	Probe - % intrrupt, kernel, user, null + Fortran task subroutine history
[343,010]	Who has mounted non-public device(s)
[343,011]	Write RX01,02 with bootable task image
[343,012]	UNDELETE
[343,013]	Downline load of LSI via TT: line
[343,014]	KILLER - BYE on another terminal w/ confirmation
[343,021]	Updated FORTH from prior SIG tape
[343,022]	VT100/52 subroutines for direct cursor output
[343,023]	RT-tape read/write
[343,025]	All the Data Management SIG Newsletter articles
[343,026]	OMSI Pascal to RMS-11 interface routines
[343,031-33]	IFTRAN Fortran pre-compiler
[343,034]	M+ HELLO mods for custom banner, pswd strikeover, nolog message
[343,035]	A VT100 film! A MUST if you have a VT100 or equivalent at 9600 baud
[343,040]	Foreign Tape Processor to read, write, dump non-RSX tapes
[343,050]	LIST - screen-at-a-time TYPE for VT100/52, Tektronix terminals
[343,051]	ASCII file transfers via async ports from VAX to VAX or RSX
[343,052]	EDT V2 as a TECO macro, also TECO DRAW macro (useful for RT)
[343,053]	PONG for VT52, VT100
[343,054]	RSX Directive or I/O Error code message display
[343,060]	IAS dynamic task scan display, dynamic node usage, corrected SRD
[343,070]	SEE - real-time memory display on VT52
[344,062]	Jim Downward's CCL for 11M version 4.0
[347,101]	Enhancements to FMS-11 Form Driver
[350,200]	Convert file read from RT tape via PIP to RSX format (ASCII)
[352,002]	TALK for terminal emulator to another computer, w/ file transfer
[352,004]	Corrections to SRD V6.0 of RSX SIG tape S81 [373,4]
[360,235]	Modified Triangle RNO for Greek characterss, super/subscripts, etc.
[370,130]	FOR/F4P Cross-reference, claims most complete of all on SIG tapes, subroutines to profile instruction execution, MAZE (3D) for VT100.
[374,001]	Games BOGGLE, HANOI, utilities (source in C) for SORT, TODAY, Superdump, dictionary of computer JARGON

Corrections to ICR Fall 1981 Tape Submissions

William P. Wood, Jr.

Institute for Cancer Research
Philadelphia, Pennsylvania

At the Fall 1981 DECUS symposium, the latest version of "The Best of ICR" was submitted to the RSX/IAS SIG tape (UIC's [330,1] through [330,16]). Also, the RATFIV preprocessor was submitted to the Structured Languages SIG tape. Since then, a bug has been found which causes LIST, BURSTF, SRDCMD, and RATFIV to abort with an open error. This bug only occurs on RSX systems (not IAS).

The problem is that these programs try to open the terminal as 'TO:', a device not available on RSX systems. The fix is to change every occurance of 'TO:' in the Fortran sources and in SYMBOLS.* to 'TI:'. I believe that the following sources are the only ones which need changing:

1. LIST - LIST.FTN and SYMBOLS.RAT
2. BURSTF - IO.FTN, GETARG.FTN, and SYMBOLS.
3. SRDCMD - IO.FTN, GETARG.FTN, and SYMBOLS.
4. RATFIV - IO.FTN, GETARG.FTN, and SYMBOLS.

If, however you have built RATFIV, the the Ratfiv sources corresponding to the above Fortran sources for LIST, SRDCMD, and BURSTF may be recompiled with RATFIV - after modifying the SYMBOLS files.

Another problem was that some of the TKB command files used an IAS switch, /RW. This should be removed before task building on RSX systems.

Finally, if you are using Fortran-77 and get errors from Fortram because the INDEX function has incompatible arguments, ignore them. In this case, INDEX is a user-supplied function.

Hints and Things

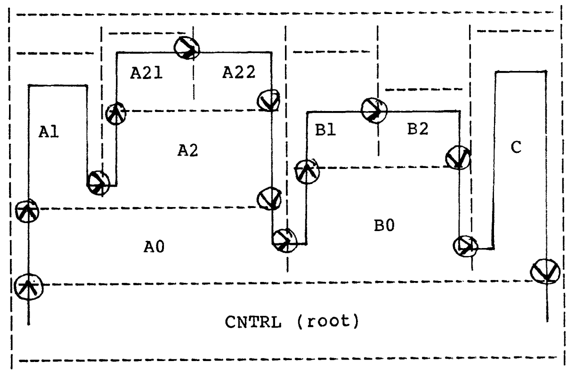
"Hints and Things" is a monthly potpouri of helpful tidbits and rumors. Readers are encouraged to submit items to this column. Any input about any way to make life easier on RSX/IAS is needed. Please beware that items in this column have not been checked for accuracy. Send any contributions to Multi-Tasker - Hints and Things, c/o DECUS, One Iron Way, MR2-3/E55, Marlboro, MA 01752.

Topological Walk to an ODL

John Covert

Digital Equipment Corporation
Nashua, New Hampshire

A graphic method may be used to convert a memory allocation diagram into the correct task builder overlay descriptor language. Consider the following diagram (taken from the Task Builder Manual, page 4-6).



The solid line drawn is the topological walk. The rules for drawning this line are quite simple:

1. Start in the lower left corner of the root segment.
2. Proceed up as far as you can go without hitting the top or empty space. Cross into new segments as needed.
3. Proceed to the right until you hit a vertical line.
4. If you are at the lowest segment of the vertical line, cross it and go back to step 2.
5. If not at the lowest segment, proceed downward the vertical line until you are adjacent to the lowest segment.
6. If you are not in the root, cross the vertical line and go back to step 2.

7. When you reach the root, you have finished the walk.

Once the line has been drawn, you should go back over it and verify all the above rules were followed. While doing this, draw arrows at each point a line was crossed to indicate the direction.

You are now ready to write down the ODL file:

1. Write ".ROOT root-segment".

2. Follow the walk. Write down the next ODL element each time the walk crosses a segment boundary, based on the direction of the arrows:

   ↑ Write "-(name-of-new-segment"

   → Write ",name-of-new-segment"

   ↓ Write ")"

3. When you return to the root you are done. The result for the example:

   .ROOT CNTRL-(A0-(A1,A2-(A21,A22)),B0-(B1,B2),C)

Theory of Interactive Debuggers

Glenn C. Everhart

RCA
Cherry Hill, New Jersey

We deal here with 3 classes of issues in debuggers: capabilities the debuggers offer to users (with random illustrations of their uses), support facilities needed to implement debuggers (hardware support and what it can do for you, and software support in terms of what language processors may provide, including strategies for source code replication), and effects of a debug aid on the context of the process being debugged (also the multi-task context, treating a multitasked application as a context to be debugged). This talk is a survey of selected debuggers and is not intended to be exhaustive or to teach anyone to use a particular debug aid.

1.0 INTRODUCTION

Interactive debuggers work by allowing partial execution of programs and selective examination of a program, replacing the dumps of yesterday. They do not substitute for compiler error checkers, strongly typed languages, etc., but are a necessary part of the real world of getting code to work as designed.

The idea of partial execution of programs with intermediate examination is that a small part of a program is easier to understand than the whole, and if enough information is available about the state of each part, the whole may be understood a bit at a time. This may mean state information not a part of usual language rules (e.g., who called a routine) is needed in the debug process, but at a minimum one needs to control execution and examine data. Additional complications arise where tracing the source of a fault is needed. Sometimes extra state information is used to track backwards to the source of the error. Where this is impossible, stepping forward may be the only way to find a problem. The most common use of code modification is to avoid the long edit / recompile / relink cycle after every fault that slows the rate of fault isolation to the rate of this cycle. There are cases of complier errors or undetected types in programs that are easiest to see where code can be viewed during the debug cycle, though.

2.0 DEBUG FACILITIES

There are several facilities a debug aid needs to be useful to the programmer. In a rough priority order of importance, they include the following:

1. Instruction Breakpoints' (stops on a given instruction). It is preferable to allow more than one of these.

2. Data Memory Display (the more data types known to the debugger, the better; a good debugger should know about all types in the language being debugged). The first 2 list items provide the most basic debug features needed, allowing partial execution of programs to examine intermediate results for errors.

3. Instruction Memory Display (preferably in the language being debugged).

4. Data Memory Modification (again many data types should be usable).

5. Instruction Memory Modification (preferably at least in assembly).

6. Machine State Display (registers, PSWs, etc.).

7. Memory Breakpoints (with subtypes allowing breaks on only writes, any accesses, modification accesses, instruction memory, or data memory). This is handy where your program is clobbering some memory location before it dies and you need to find out what is clobbering memory, and not that zeroed memory is not your program. Without something like this, locating such a bug is a matter of trial and MUCH error.

8. Single Stepping (which is handy where logic is complex and inserting/removing breakpoint wastes time).

9. Tracebacks and History Displays.

It is also desirable to support any unusual features of the system such as overlays, when these might affect the status of viisble memory. These are the basic features a good debug aid will offer. To be most useful, they should work in whatever language the program being debugged is written in. The further the debugger is from this, the harder it is to use. Thus, a purely numeric (e.g., octal) debugger is inferior to one that allows code to be displayed or entered in assembler, and that is inferior to one allowing display or entry in, say, PASCAL (where PASCAL is the language in use; it is worse to have a pascal debugger when your program is written in, say, PL/I, than one showing you assembler and at least not misinterpreting your data structures).

Another desirable feature is to allow user symbols to be known to the debugger, to permit him to avoid magic numbers as his references.

Symbolic debuggers for HOLs generally do not display HOL code from memory, since the translator would be expensive even if feasible. What they normally offer is the ability to work with named symbols and some compiler constructs (such as line numbers) to control execution. Consider FDT and FODT from the RSX SIQ tapes; they allow symbols to be accessed and execution of programs to proceed a line at a time or to stop on a given line number. They never attempt to show FORTRAN code, but assume a listing is available. ADT under UNIX knows how to display symbols used in C. and also understands the C stack frames so it can show calling traceback sequences and arguments. Display is in assembler for instructions however. Very little is really needed to support a language where compilers and linkers preserve symbols and some uniform code identifiers (line numbers, for instance). Some debuggers have differing command features depending on language too, but the point is that a set of minimal features can support HOL debugging as well as assembler debug, so long as this is understood to mean symbol access, not interactive compiling / decompiling. A general debugger might have any of the features mentioned above, though. These features each have costs; I will now discuss how some are imlemented on various machines, with special attention to PDP11.

3.0 SUPPORT NEEDED

Even providing simple instruction breakpoints can be a tough job on a machine without a breakpoint trap. Machines like the PDP9 and PDP10 enter breakpoints by overwriting instructions with subroutine calls. This traps the execution of that instruction all right, but when one tries to proceed from the break, the debugger must emulate the instruction in software. On the PDP11 the debugger needs only to replace the instruction and execute it IN PLACE with the T bit set to trap after it finishes, then replace its trap and go on. The versions of DDT on the PDP9 and PDP10 therefore cannot breakpoint jumps and subroutine calls because they cannot emulate them; on the 11, the only instructions that cause trouble are the RTI and RTT instructions, which are (fortunately) rare in user code. Machines with a hardware breakpoint register may act either way, depending on details of how it works. In the worst case, it may not be possible to have more than one breakpoint - a real disadvantage.

Display of data and instruction memory by address is usually not difficult, and display of multiple formats requires only that memory for suitable conversion routines be available. Anyone who has tried debugging programs that contain floating point numbers with DDT (Octal Debugging Tool) will see the

value of a debugger that can display data in multiple formats. If a language supports some strange packed formats, it is desirable that its debugger be able to make them human readable. There are places where this can be a problem though. Consider the new MIL-STD 1862 architecture. It has protection modes which make certain parts of the process's status (e.g. context stack) unavailable to the rest of the process. A debugger that must display this information must hope an operating system call is available to allow it to obtain the information. This sort of problem can arise whenever process-relevant information is forced to be hidden; the normal process code will not normally have any business obtaining this information, but a debugger often must be able to. (This is a caution to machine designers not to try too hard to protect users against themselves; it can result in protecting the machine against being used.)

Modification of memory is not much of a problem unless there are access protections to memory, but modification in a HOL may interact with display in that HOL since parts of a compiler and code generator would be needed to insert code in anything higher than assembler, and optimization data is not available. I have never seen a debugger that goes much higher than assembler in inserting code. I also do not believe such entry will often be, worth while in any higher level than assembler. Where provision is made for display of source code, it may be necessary to flag areas modified by the user to show where new code has been inserted if code is allowed to change. (Changing code can reduce debug time considerably to one who is knowledgeable, though, by making it possible to defer a re-assembly/relink sequence. Entering data to correct the flow of a program that depends on it is easier, and this is far more common. The same considerations as mentioned in display apply. Display of machine state is generally no harder (or easier) than memory display and again, hardware protection can make life difficult.

Memory breakpoints of most types are very nearly impossible without some Hardware assist, and even with some forms of assisting hardware (which may interrupt when the memory bus sees a particular address), cache can make read breakpoints occur imperfectly. Write breakpoints are easier (since most chache systems use write-through) and are more interesting. A true write breakpoint will detect writing the same value as was in a location. This really can ONLY be handled with hardware. Another type of pseudo-breakpoint is the "watchpoint" as implemented in DDT-11. This is really an automated single step program which will stop execution if the value at an address changes. Because every instruction must be stepped, the debugger must be sure not to lose control (an RTI will do it in). The main effect is to GREATLY slow down the program. On machines like the PDP11 there is little choice, though. Notice that on machines like PDP9 this would have been much harder because breaks cannot be placed on calls there.

Single stepping is a shorthand for inserting breakpoints and proceeding from old ones. Where there is a trace bit (a la PDP11) it's easy. Without one, it may require interpretive execution of an assembly language program. Some refinements on single steps which do simple decisions about whether to actually allow user interaction with the debugger are fairly common too. These may include repeat counts or tests of special conditions (e.g., break if address foobar is negative, not otherwise), or whatever seems reasonable. A particularly flexible way to handle these conditions is to allow a set of stored commands to be run at a break to do the decision. However, very general facilities of this kind are hard to find that help more than a few real life problems. Tracebacks or code profiles are usually easy to produce provided there is space for them: every breakpoint facility I know os gives access to the program location at a break, and that is enough to construct histograms of how often each was reached.

4.0 SYMBOLIC DISPLAYS

To allow display of instructions or variables symbolically, there have been several methods proposed. The most common way to access user symbols is to have language processors save them in files the debugger can access, either by having them produce data structures in the debugger or by having them read off disk. On the PDP11 this has not been well supported. The taskbuilder outputs global symbols, but local ones are not output by the more common DEC languages. Some DECUS workarounds, have been written (most notable FDT which produces symbol files from listings and maps), but the problem remains that even where the languages save internal symbols, TKB does not. Since most 11 users cannot run LINK11 on a PDP10, this defeats all but the workarounds that use map files. On other machines, symbols are very often available. Program code is generally displayed either in a numeric radix or as assembly code via a disassembler, which is usually not too hard to write. Where a higher order language is required, much cooperation is needed with that language's compiler. One method of code reconstruction has compilers placing tokens in the output to tell what kind of statement is in use. Compiled code then has constructs (possibly switchable) which let a debugger know it is, say, in the middle of a DO-WHILE construct. Another proposal suggested the compiler save the actual source file in indexed form and generate pointers to that form, so a debugger could extract the complete source text, with comments. Obviously, changing the program would invalidate this display.

5.0 CONTEXT

Most of us have probable had the experience of loading a debugger into a task and have it not fit, or having the problem that made us load the debug aid go away mysteriously. These are context effects. The context of a process as used here means the machine resources relevant to its computation, which may include memory, proccessors, trap vectors, and so forth. The context of a debugger depends on machine and operating system. in RSX-11, debuggers are normally part of the space of a task and share the traps of that task, and in single user systems like DOS-11 and RT11 this is necessarily the case when no mapping is available. However, UNIX has debuggers normally outside a process' space and traps are seen by the OS first, then passed to a debugger (larger context objects than single tasks are known). VAX can also handle larger objects than tasks. As distributed applications evolve, a context may encompass more than one task and moe than one CPU. Debugging such applications can be quite complicated because the idea of a debug aid is to allow examination of any part of the application, which is not feasible in some cases.

Where a debug aid is part of the space of a normal task, linking it into the task will often change addresses of tasks even if it fits. This can create of mask problems. Where the debug aid can be in a separate area of memory, these effects can be minimized. This can usually be done given some sort of

reasonable multiprogramming environment with some intertask commuication, because the amount of information that must be sent to a debug aid at a breakpoint is fairly small. DDT can get by easily with the 13-word RSX packets and use a small "kernel" of debug features built into the task, with the bulk of the debugger in a separate task. Given a transport mechanism for the packets, target and debugging tasks could even be in different machines. The way this works is that task-local contexts are saved locally, so only the PC and PS and some pointers need be passed. Data can be moved each way be sending packets containing "MOV" instructions which minimizes the size of the "kernel". By suitable exec modifications, even this kernel can be eliminated, at least on any single machine, since RSX saves all the information needed also. Where the debugger is already separate in context (consider the old PDP11 ID debugger), interference with address does not occur. This is less disturbing, but does usualy have some timing disadvantages where single stepping should be kept quickly more system context swapping is needed.

In the case of multitask applications, a useful kind of debug aid would allow several communicating tasks to be debugged. I have heard of 2 ways to approach this. The approach taken by DDT and by SPEX ODT is to serially debug one task at a time in orders determined by breakpoints in several at a time. This means that a breakpoint in one task halts that task and allows examination of its actions, but allows others to keep running. This is not a faithful stopping of time for the system, but may be better than such for finding intertask data. The other, approach was taken by a specially modified XDT called MDT which stops all other processes (except some I/O) whenever any breakpoint is seen. This really does prevent errors due to improper timing relationships, but can have some problems where some parts of an application must run continously (e.g., watchdog server processes, to keep another processor from cutting out). The first scenario is far easier to arrange in distributed systems, but at times the second may be needed.

The DDT Debugger for PDP-11

Glenn C. Everhart

RCA
Cherry Hill, New Jersey

This document is a beginner's introduction to using DDT. It does not contain all DDT commands, but a "novice subset" sufficient to be useful. DDT is found on the RSX SIG tapes in account [312,315]. The latest version is on the Fall 1981 SIG tape (Los Angeles).

1.0 WHAT IS DDT?

DDT is a symbolic debug aid. With existing debug aids like ODT, you can execute programs a bit at a time, but you can display programs only in octal and need to refer continually to maps and listing files. Also there are some numbers (e.g. floating point) that are hard to display: the octal radix is not meaningful. With DDT you can display or enter MACRO-11 instructions, octal or decimal numbers, text, floating point, or other formats, do all DDT functions, and refer to your symbols by name rather than as octal mystery numbers. This allows debugs with only source listings, which need not be recreated with EVERY edit since DDT's instruction display makes it easy to find code sequences you know. Since debug aids sometimes don't fit, a special DDT kernel (DDTKNL) can be built into your program, taking as little as 128 words (more typically 200), and you can debug with DDT in a separate task. Or you can just include DDT in your task's space as you do with DDT. DDT works in any PDP11 on any DEC operating system (even has code for user mode I and D space for RSX11M+ V2.0), though under RSTS using RT11 or RSX emulation its operation is not tested. The debugging from a separate task works under RSX11M, M+, or IAS (though under IAS the tasks must run realtime). Descriptions here will be RSX oriented.

2.0 GETTING STARTED WITH DDT

To first use DDT, you must include a copy of DDT into your task with the taskbuilder (TKB). We will assume it fits first.

You include DDT as a debug aid by specifying it with the /DA switch on the INPUT side of your taskbuild command line (or, with the /DA switch in the root of your .ODL file). That is, you would use a command like

```
        TKB>myfile.tsk,myfile.map,myfile.stb=myfile,subl,...,[1,1]DDT/DA
```

which will include DDT from [1,1] (substitute another UIC if that is where DDT.OBJ is) and set (it up as a debug aid. Note you do NOT specify /DA on the output side of the TKB command line: that will load DDT and cause conflicts.

DDT is built by running DDTBLD.CMD and answering questions: it is assumed your system manager has done this and the DDT.OBJ is your "system standard" DDT, a "1-task" DDT version. Since DDT can be rather large, if you get an error message like "ADDRESS SPACE OVERFLOW - ALLOCATION DELETED" on large programs, there is a special DDT kernel called (mysteriously enough) DDTKNL which will allow your task to be built and debugged with most of DDT in a separate task. If you use DDTKNL, just replace the "[1,1]DDT/DA" with "[1,1]DDTKNL/DA" in your command lines.

When TKB finishes, if DDT is in your task space you can just run the task and DDT will be active. If DDTKNL is used, you must activate DDT22M first. To run DDT22M, type:

```
        RUN [1,1]DDT22M/TASK=DDT22M
```

(or the equivalent if DDT22M is either an installed task or in another UIC). Then type <esc>UM and then <esc>Q to start DDT22M. The "<esc>" means the ESCAPE key (octal 33), echoing as $, not the 5 characters shown. In the following, the "$" character will be used to mean ESCAPE, and NOT the literal "$" character (that is, it will represent ASCII code 33, not ASCII code 44). Now your task is ready to run. (Note some versions of DDT22M auto-start). Then run your task. Once DDT22M types out a message, type $UM (escape, UM) to ensure DDT22M is looking at the target task. Now you can pretend you are debugging from within your task: most things will be the same. The $UM command tells DDT to use the remotely mapped task's space for areas to examine/modify. It is undone by the $UM command which tells DDT to use its own virtual address space. These

commands switch address spaces.

3.0 GENERAL CONSOLE FEATURES

DDT handles the console in a somewhat nonstandard way. When it starts, it types a message out, but does NOT prompt! That is, it will NOT type "-" the way ODT does to say it is waiting for input: it just waits. You type to it (in either case: DDT is not case sensitive) and DDT acknowledges correct actions by typing a tab (actually it types from 1 to 7 spaces). Thus, several lines may be placed on a line. If you type a "delete" or "rubout" (ASCII 177) character, DDT will type a "XXX" and wait for you to retype the whole command. It will NOT allow single character corrections as RSX does: its parser is much too simple for that.

It is important that you realize DDT will act as soon as you enter a command. There is no "command terminator" such as a carriage return or double escape to start the command. All you have is the DDT standard acknowledgement of completion by typing a tab (actually, some spaces). If DDT didn't type the tab, most likely something went wrong. DDT is picky about syntax and not very helpful about errors. It types U for an undefined symbol, and ? for most other errors.

4.0 FURTHER SETUP

Your program is permitted to use the TRAP or EMT or other trap instructions. If it does, zero locations DDTTRP, DDTEMT, or similar names. Some versions of DDT will not catch TRAP instructions automatically since FORTRAN and F4P use them. (If using DDTKNL, forget this: you change what traps DDTKNL gets by editing and reassembling DDTKNL). Use one of the commands below to put a zero into the location named DDTTRP or DDTEMT for TRAP and EMT respectively.

5.0 EXAMPLES

If you run DDT as part of your task, your input might look like this, assuming you have a map and subroutine MUMBLE is the one to be debugged. Comments at the right are not typed, but are here to explain individual steps. The <CR>, <LF>, or <EBC> symbols are used to represent the RETURN, LINEFEED, or ESCAPE keys.

Notice that the commands "/" (open in current mode, default instruction mode), "[" (open as numeric, default octal), and <LF> (close, open next) are used with some other commands below. We will describe these commands in detail later. For the present, the comments describe what is happening.

```
        >RUN MYPROG                   Run the task to be debugged

        DDT-11 V004A RSX/FPU          DDT prints identifier. (This tells some
                                      available features in DDT, in this
```

```
                                   case that floating point display is
                                   available.)

     31642<MUMBLE:                 Define address of MUMBLE as 31642
     MUMBLE+62<ESC>B               Set a breakpoint at MUMBLE+62
     <ESC>G                        Start the task

     MUMBLE+62 >> 1B               Eventually reach the breakpoint
     R2[3274                       Examine R2 in octal, find 3274
     R1[1                          Examine R1 in octal, find 1
     R5[32740    [31               Examine R5, find it contains 32740.
                                   Next "[" examines location 32740 and
                                   finds it contains 31.

     MUMBLE+62/ MOVB (R4)+,(R0)+<LF> Examine 3 instructions starting at
     MUMBLE+64/ JSR PC,35776<LF>   the breakpoint address to check the
     MUMBLE+70/ SOB R0,MUMBLE+20<CR> code at breakpoint

     <ESC>P                        Proceed from the breakpoint.
```

If using DDTKNL, you will operate similarly, but the first faw steps of a debug session will look like this (assuming there is a runnable DDT22M in LB:[1,1]):

```
     >RUN LB:[1,1]DDT22M/TASK=DDT22M The "/TASK=DDT22M" MUST be there!
     DDT-11 V004A RSX/FPU/MTSK     DDT22M identifies itself
     <ESC>UM                       Set up DDT22M to examine the target
                                   task space once it gets a breakpoint
     <ESC>G                        Start up DDT22M. Note the terminal is
                                   not attached, so you can now proceed.
     <CR>

     >ACT                          Check that DDT22M is really active
     ...MCR
     DDT22M

     >RUN MYPROG                   Run your task as above
     TT7 BGN22M+1536 >> BPT!       Startup message of a DPT to DDT
     31642<MUMBLE:                 Define address of MUMBLE as 31642
     MUMBLE+62<ESC>B               Set a breakpoint at MUMBLE+62
     <ESC>G                        Start the task
```

The rest of the session goes as above. When done with the session, type "ABORT DDT22M" to kill the debugger. With DDT22M, it will frequently be possible to use a STB file to define locations automatically. Thus one would issue a command like:

```
     <ESC>UO .STB FILE:MYPROG<CR>    You just type MYPROG<CR> (STB assumed)
```

Note that the STB file is closed when DDT cannot find a value for a named symbol. It may be re-opened with the <ESC>UO command, and symbols are cached.

6.0 BASIC DDT COMMANDS

The following are a "basic subset" of DDT commands. DDT has a large set of commands, most of which are not needed for basic user interaction. A number of these set "modes" governing some details of other commands (e.g., default typeout format).

6.1 LABELS

A DDT label is any 6 or less alphanumeric characters (with and $ permitted also). You may refer to a location in several ways:

* LABEL or LABEL+nnn or LABEL-nnn (nnn a number)

* nnnnnn (an octal number)

* nnnnnn. (a decimal number)

Where the term "address" is used below, it means any of the above. Where a LABEL is used, it must be defined first: DDT is not prescient.

6.2 DDT Commands That Display Or Modify Memory

1. address/   (open address in current mode)

   This command will display the contents of the word (or up to 3 words in instruction mode) viewed in the current mode. These modes may be MACRO-11 instructions, octal numbers, decimal numbers, ASCII text, or RAD50 packed text (as the most common selections). The default when DDT starts is to display instructions. DDT will tab over to the right when the location is displayed, awaiting a further command. The normal ones are to possibly insert a new value, then type either <CR> or <LF> to close the location (and open the next if <LF>). New values are specified as one of:

   1. MACRO-11 instruction.

   2. Octal number (if default radix is unchanged).

   3. Decimal number (period after a number means it is in decimal).

   4. "'cc' (double quote, delimiter, 2 ASCII characters, then the same delimiter again, inserts the 2 characters in ASCII into the location.

   5. "<ESC>'ccc' (double quote, escape, delimiter, 3 RAD50 characters in RAD50 into the location.

   Several commands change the modes used for the / command. (Note that using a single <ESC> will allow reopen in a new mode until a <CR> is entered: the forms given are effective until changed.)

2. <ESC><ESC>S (Symbolic mode)

   This command changes the mode to symbolic instruction mode, i.e., MACRO-11 instructions. DDT starts in this mode.

3. <ESC><ESC>A (Absolute addresses)

   This command causes any numeric symbol type outs to appear as "pure" numbers instead of offsets from the next lower defined label if there is a label whose value is "close" to that number (this is defined as 128 or less at start.)

4. <ESC><ESC>R (Relative addresses)

   This command causes numbers to be typed relative to the next lower symbol if that symbol is "close" (within 128) before the number. Note that the numeric value may always be seen by typing "=" (equal sign) to print as (normally) pure octal.

5. <ESC><ESC>T (Text typeout)

   This command causes the / command to display memory as ASCII text.

6. <ESC><ESC>H (Halfword typeout)

   This command causes / to display bytes. It is reset by the <ESC><ESC>S or the <ESC><ESC>5T commands. Normally the address for / is expected to be even unless in this mode.

7. <ESC><ESC>5T (RAD50 typeout)

   This command causes / to display memory as unpacked RAD50.

8. address[(Open address as numeric)

   This command displays the contents of address in numeric radix, normally octal. A second [to a displayed address will display the contents of where it points.

   The command <ESC><ESC>10R will change the display radix to decimal and the command <ESC><ESC>8R will change it to octal. The <ESC><ESC>16R changes the radix to hex, but not all numbers can be handled. This arises due to DDT's inability to distinguish symbols from hex numbers beginning in A through F. There are extended commands able to handle hex more consistently.

9. address? (Display address in RAD50)

   This command displays the address in RAD50. It also forces the address to be even by zeroing the low bit.

   Another group of display commands are used to terminate the display and modify sequence. The most commonly needed are the following:

10. [number]<CR> (modify and close location)

    This command will insert the number (instruction, ASCII or RAD50 text, or whatever) in the location given and close the location, terminating the command. If no number is entered, the location is simply closed. THIS IS IMPORTANT! If you could not close a location, the next address you wished to examine might be inadvertently placed in the last location. It is essential to close a location before beginning work on another one.

11. [number]<LF> (modify and close location, open next location)

    This command first performs the function of <CR>, i.e., optionally modifies the currently open location and then closes it. It then opens the next location in the same way as the last and displays it and its contents. Note that the "next" location depends somewhat on how the display is set up. A byte display will advance by 1. word displays by 2, and instructions may show a next location that is 1, 2, or 3 words later. The insertion of an instruction is not done until a terminating command is given, even if it is 2 or 3 words long. Thus, a "delete" (rubout) keyin will abort it even if most of the instruction is typed in. As an aside, use of Rather than <LF> will back up but always by 1 word or byte.

12. @ (close location, open location addressed)

    This command closes the current address an opens the address it points to. This is handy for following a pointer and quickly examing a table it points to.

6.3 DDT Commands That Control DDT Options

1. address<LABEL: (define address to have name LABEL)

   This command assigns the symbolic name LABEL to the given address. The LABEL literal may have up to 6 characters legal in RAD50 (i.e., A-Z, 0-9, ., and $) and must begin in ann aloha. Note you can enter LABEL: whenever you have a location open too, to assign the name: the colon is the operator. Up to 160 user defined labels may be entered. One uses these much as one uses DDT "relocation registers", as bases for addresses. Thus, it is legal to specify an address as LABEL+const (e.g., "LABEL+1026") and normal to do so. One may use a form like "LABEL-const" too.

2. <ESC>UO (Prompt for symbol table file)

   This command allows DDT to read symbol table files to try to resolve symbols. The files are closed by <ESC>UQ or when an undefined symbol occurs. Thus, it is good practice to use the <ESC>UO command and answer the "STB FILE:" prompt with the name of your STB file, when immediately use the = command to get DDT to evaluate symbols. An example would be something like this:

```
        $UO
                STB FILE:msx.stb
        s.rsav=23442
        s.rres=23462
        mx.stl=1076    mx.tbl-1104    stated4=13206
```

At this point, the named symbols are known to DDT and can be used. DDT caches about 20 symbols from the .STB file in memory, permitting this.

6.4 Commands To DDT For Controlling Program Execution

1. address<ESC>B   (Insert Breakpoint)

   This command sets a breakpoint at the given address. That is, the program will stop execution when it reaches that instruction and DDT will print a message indicating where it is and allow user examination or modification of memory. Note further that DDT breakpoints must be on instructions. Breakpoints on data, or on instruction words other than the first of an instruction, will not ever be effective, and will probably result in incorrect execution as well. (DDT replaces the instruction with a trap unless in single step mode, so the trap must be executed to be effective.) Breakpoints may be on any instruction except an RTT or RTI instruction. Note that you hit a breakpoint BEFORE the instruction you "break" is executed. It will be executed when you proceed.

2. <ESC>B   (Remove all breakpoints)

   This command removes all breakpoints. Note that to remove a single breakpoint 0 to 7, setting its address to 0 does this (the command would be 0<ESC>3B to remove breakpoint 3 only.) You run out of breakpoints after number 7 and this lets you clear them out to reuse them at other addresses.

3. <ESC>P   (Proceed from breakpoint)

   This command resumes execution of a program after a breakpoint. There is a variant to allow n passes through the break before any type-out is done. You cannot proceed after a fatal error however; you can only use the Go command (below).

4. <ESC>G   (Start Program at default start address)

   This command just starts the program at its default start address from the taskbuilder (recorded in the cell JOBSA in DDT an available in that name. Normally this address is labelled BON too and can be examined in that form. It may be called BONTnn at times as well.

5. address<ESC>G   (Start program at address)

   The address$G command (Go) starts execution of a program at the address given. It is legal anytime the program is stopped for any reason.

6. <ESC>1UT   (Turn on single stepping)

   This $1UT command turns on single stepping by arranging a breakpoint after every instruction. (It executes all instructions with the T bit set.) You get a breakpoint message for ficticious breakpoint 8 after each instruction and use the Proceed command to execute the next instruction. This mode is handy if you know the program is leaping off into space somewhere and you want to find out where it does it.

7. <ESC>UT   (Turn off single stepping)

   This command restores normal operation and turns off single step mode. You need to be in the normal mode for exiting to work normally.

6.5 Other Useful Features Not Described

DDT is able to watch up to 8 locations (more by reassembly) while executing a program and generate a memory breakpoint whenever any of the locations changes vale. If something in your program is clobbering a part of memory, this command (the UZ command) allows you to find the culprit provided it doesn't clobber DDT first.

There is logic in DDT to check some memory address when a break is reached and skip the breakpoint unless that address, ANDed with a mask, is equal to a test pattern. This is good for breaking on a single bit or a few bits changing where the content of a full word is not of interest. Obviously, one can go very far providing conditional breaks, but DDT does only this, a fairly simple test.

DDT can display or enter 2 or 4 word floating point numbers if built with that support, and can examine or modify floating accumulators (internally named AC0 through AC5). There are several control words to control this.

DDT is able to display 32 bit integers in any radix, or enter them in any radix (up to 36: beyond radix 36 it gets silly). The storage convention it uses is that of FORTRAN.

DDT can examine any area of physical memory or any file on disk as though that file were memory, providing a symbolic ZAP. It can also do nonfile structured modifications of disks, or open files or disks in read-only mode.

DDT has support for a special breakpoint in overlay load code (symbolic location $ALBP2) that will not modify overlays, allowing that (and only that) break to signal the user to remove old breakpoints and insert some that are meaningful to the new overlay.

6.6 Automatically Defined Variables In DDT

When you start in DDT a number of variables are defined for you automatically. Among them are the following, which are examined or modified with ordinary memory examine/modify commands as shown above:

RO through R5, SP, and PC	- These are the PDP11 general registers as normal in MACRO-11.
$DSW	- The RSX Directive Status Word at the break
<ESC>I	- The program's PSW
<ESC>21	- DDT's PSW
ACO through AC5	- FPU accumulators
JOBSA	- Word holding program start address
BYE	- Address in DDT to start at to exit program. (Also may be reached by typing control Z).
D.FILV	- Word holding pattern (initially 0) to be filled in by the $Z (memory Zero) command.
<TASK NAME>	- The start address of the task
DDTEMT	- Address for DDT to use for non-RSX EMTs. Zero to allow task access to these.
DDTRES	- Address for trapping reserved instructions Zero to allow task access to these.
DDTTRP	- Address for trapping task TRAP instructions. Zero to allow task access to these.
DDTIOT	- Address for trapping task IOT instructions. Zero to allow task access to these.
DDTODD	- Address for handling odd address errors. Zero to allow task access to these.
DDTBQN	- Start address of DDT itself.

Making Use of Secondary Pool Under RSX-11M-Plus

Glen Hoffing

RCA Government Communcations Systens
Camden, New Jersey

One of the nice enhancements of RSX-11-PLUS was the creation of secondary pool space. Its primary benefit is to offload much of the work from primary pool space, so that the pool space problems endemic to RSX-11M are no longer a worry. Another benefit that may not have occurred to many M-PLUS users is that the secondary pool, in conjunction with the variable send/receive data packet directives also implemented in M-PLUS, can now become a very useful tool for memory buffering of significant amounts of data.

An example of this is two applications tasks, one of which provides data acquisition and the other of which performs data reduction on the acquired data. Under normal operating circumstances, the reduction task is more than able to keep up with the acquisition task, but under peak loading conditions it is not.

The solution, of course, is to provide a data buffering capability between the two tasks. One way, slow but simple, is to buffer the data to disk. A more efficient way is to buffer the data in memory. The user can set up his own shared common region for the data, complete with a circular queueing mechanism and "overflow valve" checks, but it is the message of this article that secondary pool can provide everything that is needed without having to create these data structures.

The first step in using secondary pool in this manner is to make the secondary pool area big enough to serve as an effective buffer. It is not necessary to perform a new SYSGEN in order to do this, but it is necessary to build a new system image from the virgin RSX11M.TSK file, using the SYSVMR.CMD file (which I am sure we have all kept up to date). The command in the SYSVMR.CMD file which sets up the secondary pool partition will typically be SET /PAR=SECPOL:*:100:POOL, which will create a secondary pool of 100 octal, or 64, blocks of 32 words each. We have increased the size of our secondary pool to 2000 octal, or 1024 32-word blocks, for a total size of 32k words. Those of you weaned on 11/34s (or smaller) may gasp, but this is a relativedly small investment out of our megabyte of memory. It is assumed that most M-PLUS users running dedicated applications can afford to allocate at least some extra memory to secondary pool.

Once we have a secondary pool of the desired size, it is a fairly simple matter to put it to use. The user who is sure he will never exhaust his dedicated amount of secondary pool need only issue variable send and receive data calls to his heart's content. In the real world, it is necessary to take some precautions against the exhaustion of secondary pool space. The simplest way is to check the directive status word on each send data directive issued. A return status of -1 indicates no secondary pool space available, and the user can wait for a significant event and reissue the directive. (A note of caution - those of you who have not applied the patch to SYSLIB described in the February 81 Software Dispatch, page 21-22, should clear the directive status word whenever it becomes set to -1, otherwise it will "latch" at that value forever).

There is a drawback to this approach. Secondary pool is a common resource and other users or tasks may not appreciate having it depleted. For instance, Fortran file opens will fail if there is no secondary pool available. A more sophisticated approach is to keep a global counter of secondary pool usage, which is incremented on each send data directive and decremented on each receive data directive. This counter can be compared to a threshhold of, say 90% of available secondary pool, and whenever that threshold is exceeded, the sending tasks can be instructed to mark time until the counter falls below the threshold. This will assure that at least 10% of seondary pool is always available for other system users.

In order to implement this it is necessary to calculate the amount of secondary pool space used for any given send data directive. Fortunately, this turns out to be an easy exercise. Secondary pool is always allocated in units of 32-word blocks for send data packets, and there is a fixed 8-word overhead in addition to the size of the packet. In our application we devine two global variables, a secondary pool counter SPOOLC, and a threshold, SPOOLT, which we set to 900. (approximately 90% of our 1024 32-word blocks of secondary pool). Whenever we issue a send data directive for a packet of size NSIZE words, we increment SPOOLC by ((NSIZE+7)/32)+1. The received packet is always two words longer than the sent packet, as the operating system appends the sending task name to it, so whenever we receive a packet we decrement SPOOLC by ((NSIZE+5)/32)+1. SPOOLC thus always reflects the number of 32-word blocks currently in use, and as such is a useful tool in monitoring system bottlenecks. Special care must be taken if a given task issues both send and receive data directives. Its threshold test for a send data must be slightly higher that other tasks' threshold, lest it become locked out of sending data and thus also out of receiving data, which could cause a system deadlock.

When implemented correctly, this approach would seem to alleviate the need to check the directive status word for an error status of -1, as we cannot by definition run out of secondary pool. In cases where all send data packets only require one block of secondary pool (24 words or less of data in the packet) this is true. However, packets requiring more than one block of 32-words apparently must find a contiguous set of blocks, as we have found that we can get a return status of -1 even when there is ample available secondary pool, when we send data packets of more that 24 words. Our experience is that secondary pool becomes fragmented very quickly when sending packets of varying 32-word block lengths, and our only solution to date is to check the directive status word and reissue the send data directive as described above. Although not the best of all possible worlds, this seems to work well enough in our application and would appear adequate for most other applications.

In summary, secondary pool has proved an excellent buffering mechanism for our application, which consists of about twelve applications tasks running in a dedicated fashion and communicating with each other extensively via variable send and receive data packets. The ability to size secondary pool to fit the application and to monitor its usage in software should make it a useful tool to anyone in a similar position.

How the RSX-11M Shuffler Works

Brian Donoghue

MCC Powers
Northbrook, Illinois

This paper is a hierarchy of discussions about the shuffler. First is a basic explanation. Though not totally accurate, it is rather easy to read. An intermediate discussion follows which goes into more detail, though accuracy is still slightly compromised. The third discussion leads up to the flowchart by pointing out the inaccuracies in the previous discussion. The flowchart contains no intentional inaccuracies.

1.0 THE SHUFFLER'S ALGORITHM -- A BASIC DESCRIPTION

1. Find the partition in trouble - the one which has a task waiting for memory.

2. Checkpoint/shuffle the tasks in that partition, thus creating a large hole of free space at the high end of the partition.

3. Find out if the waiting task can be satisified now. If so, the shuffler may exit. Otherwise, go to the final step.

4. Checkpoint tasks of a lower priority than the waiting task to create even more free space. Shuffle the leftovers to merge free space into one giant hole. Exit.

2.0 THE SHUFFLER'S ALGORITHM -- AN INTERMEDIATE DESCRIPTION

This discussion begins in the Executive...

Executive: "Gee, I sure would like to allocate some main memory to this task I have got here waiting out on disk. Let's see, he wants to run in a particular system-controlled partition. Is there enough free space in the partition to just load the task? No. Shucks. If I checkpoint a continuous block of lower priority tasks which do not have any outstanding I/O, would a sufficient amout of memory appear? Nope. What if I try the same thing but ignore outstanding I/O? Still no room. Damn. There is only one guy who can help me out of this jam: the Shuffler!"

Upon being requested to run by the Executive, the shuffler follows the algorithm below:

1. First it scans the list of partition control blocks (PCBs), searching for a system-controlled partition with at least one task waiting for main mamory. When such a partition is found, the shuffler begins its first pass algorithm by going to step #2. If, however, the PCB list is exhausted before finding a partition in need, the shuffler exits.

2. The shuffler examines each task within the partition in order of increasing address and performs the following test:

   Is the task checkpointable and either stopped or blocked?

   ```
   YES -- Checkpoint it
   NO  -- Shuffle it to a lower address if any free space
          lies below the task
   ```

   After a task is operated on in this fashion, a test is made to see if the partition is satisfied (by "satisfied", I mean no tasks are competing for memory in the partition). If not, the shuffler operates again on the next task residing in the partition. If the partition is satisfied, the shuffler goes back to step #1 to see if any other system-controlled partitions need help.

   When there are no more tasks to examine in the partition and a task is still waiting for main memory, the shuffler goes onto step #3 to begin its "second-pass algorithm."

3. The shuffler creates a list of all tasks currently resident in the partition in order of increasing priority. (Actually, the list is made for each fragment in the partition, where a fragment is a contiguous portion of memory bounded by unshufflable things such as partition boundaries, drivers, tasks fixed by parity errors, and such).

   From this list, the shuffler determines whether the waiting task may fit if lower priority tasks are checkpointed. If so, checkpointing occurs and the first pass algorithm (step #2) is reexecuted. If not, the partition is declared to have reached a stable state and the shuffler fors back to step #1 to see if any other system-controlled partitions need help.

3.0 COMMENTS REGARDING THE ACCURACY OF THE INTERMEDIATE DISCUSSION

First, outstanding I/O was not treated at all. During the first pass, outstanding I/O is ignored when considering whether or not to checkpoint a task. However, when shuffling a task to a lower memory location, the shuffler waits for up to a half-a-second for the I/O to complete. If the I/O count is still non-zero at the end of that time, the shuffler leaves the task where it found it and proceeds to the next task. Because the shuffler allocates the free space below the task in advance, that free space may be tied up (wasted) for up to a half-a-second.

Second, the intermediate discussion does not make clear the shuffler's extreme devotion to iteration. So that this is clear when you inspect the flowchart, I have simplified the shuffler's algorithm to the base essentials below:

1. Find a partition in trouble.
2. Find the first task within that partition which can be either be checkpointed or shuffled.
3. Checkpoint or shuffle that single task as desired.
4. Start all over again. Do not bother remembering anything - just start over from the beginning.

See how simple that is? In fact, it may remind you of recursion.

4.0 THE SHUFFLER'S ALGORITHM -- FLOWCHART FORM

See the attached flowchart. I have no particular comments at this point.

5.0 NEW FEATURE: THE SHUFFLER NOW HAS A RUN LIMITER (RSX-11M V4.0)

There is a location in the Executive which contains the minimum delay (in ticks) between calls to the shuffler ($SHFCT). Before calling the shuffler, the executive must first determine whether the delay has expired. If it has not, the Executive graciously accepts the hardship, does not call the shuffler, and gets on with its work.

6.0 CONCLUSIONS

These facts are evident:

1. The shuffler runs only when needed.
2. The shuffler does not blindly charge through GEN during its first pass. If all waiting tasks are satisfied early, the first pass is left incomplete.

The following statement seems reasonable:

The consequence of not using the shuffler on a busy system is that low priority tasks may not have a timely response. The shuffler tends to give low priority tasks more chance to get into memory at the expense of burdening the system with more overhead.

Therefore:

If you want low priority tasks to have a timely response, use the shuffler on your system. If on the other hand, low priority task can suffer, do not use the shuffler. In gray situations, use the shuffler but set is run limiter up from the default to prevent excessive overhead.

7.0 REFERENCES

Suffler source code (SHUFL.MAC) for RSX-11M V4.0.

Executive source code (TDSCH.MAC, REQSB.MAC) for RSX-11M V4.0.

RSX-11M System Generation Manual, V4.0 field test version, pages 4-33 to 4-35.

THE SHUFFLER'S ALGORITHM

RSX-11M V4.0

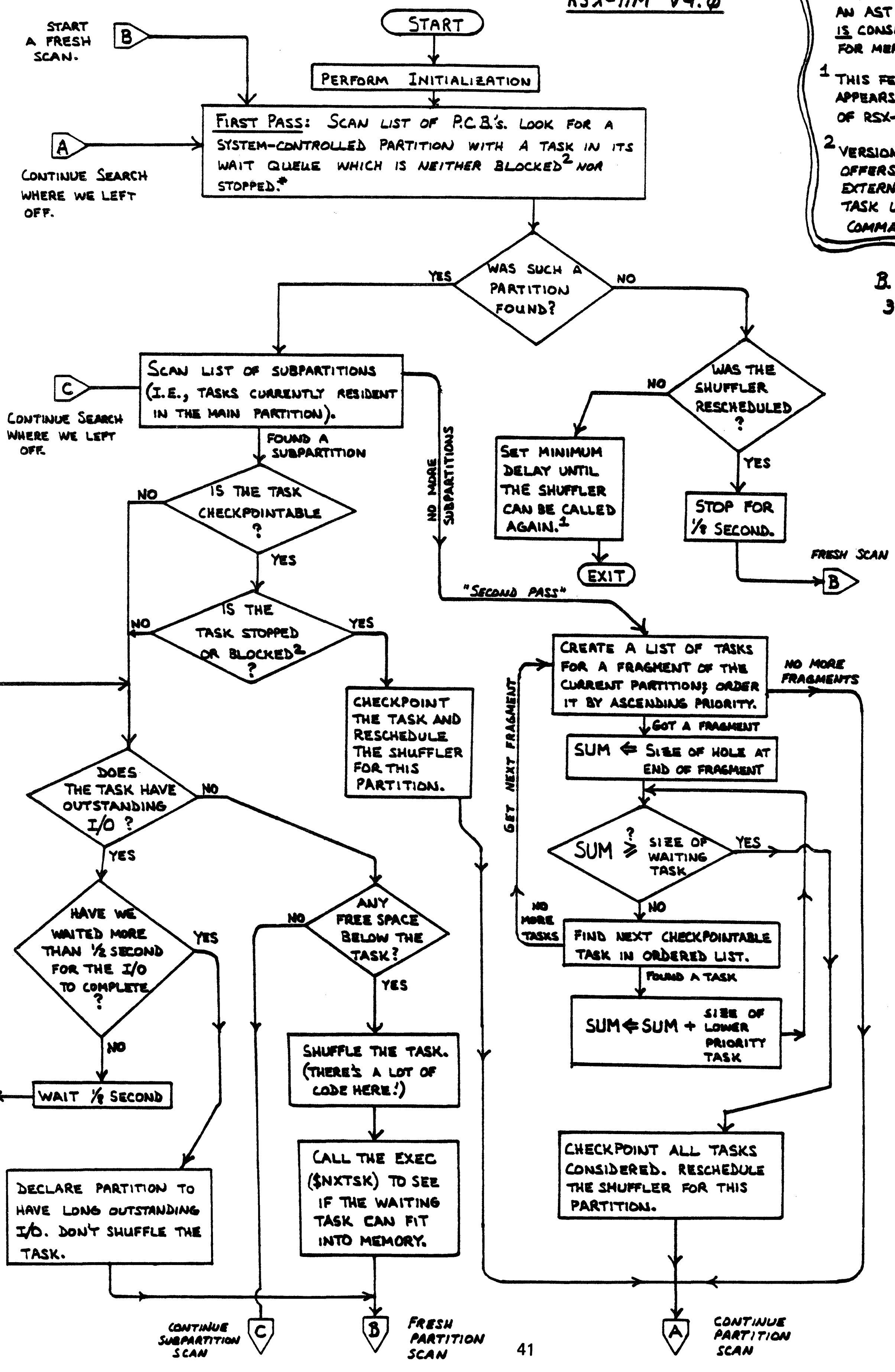
NOTES:

* A STOPPED TASK WITH AN AST QUEUED, HOWEVER, IS CONSIDERED ELIGIBLE FOR MEMORY.

1 THIS FEATURE ONLY APPEARS IN VERSION 4.0 OF RSX-11M.

2 VERSION 4.0 OF RSX-11M OFFERS THE ABILITY TO EXTERNALLY BLOCK A TASK USING THE "BLK" COMMAND.

B. DONOGHUE
3-22-82


START
PERFORM INITIALIZATION
START A FRESH SCAN. B
A CONTINUE SEARCH WHERE WE LEFT OFF.
FIRST PASS: SCAN LIST OF P.C.B.'S. LOOK FOR A SYSTEM-CONTROLLED PARTITION WITH A TASK IN ITS WAIT QUEUE WHICH IS NEITHER BLOCKED2 NOR STOPPED.*
WAS SUCH A PARTITION FOUND?
YES
NO
SCAN LIST OF SUBPARTITIONS (I.E., TASKS CURRENTLY RESIDENT IN THE MAIN PARTITION).
C CONTINUE SEARCH WHERE WE LEFT OFF.
FOUND A SUBPARTITION
NO MORE SUBPARTITIONS
WAS THE SHUFFLER RESCHEDULED ?
NO
YES
SET MINIMUM DELAY UNTIL THE SHUFFLER CAN BE CALLED AGAIN.1
EXIT
STOP FOR 1/8 SECOND.
FRESH SCAN B
IS THE TASK CHECKPOINTABLE ?
NO
YES
IS THE TASK STOPPED OR BLOCKED2 ?
NO
YES
CHECKPOINT THE TASK AND RESCHEDULE THE SHUFFLER FOR THIS PARTITION.
"SECOND PASS"
CREATE A LIST OF TASKS FOR A FRAGMENT OF THE CURRENT PARTITION; ORDER IT BY ASCENDING PRIORITY.
NO MORE FRAGMENTS
GOT A FRAGMENT
GET NEXT FRAGMENT
SUM ⇐ SIZE OF HOLE AT END OF FRAGMENT
SUM ≥? SIZE OF WAITING TASK
YES
NO
NO MORE TASKS
FIND NEXT CHECKPOINTABLE TASK IN ORDERED LIST.
FOUND A TASK
SUM ⇐ SUM + SIZE OF LOWER PRIORITY TASK
DOES THE TASK HAVE OUTSTANDING I/O ?
NO
YES
HAVE WE WAITED MORE THAN 1/2 SECOND FOR THE I/O TO COMPLETE ?
YES
NO
WAIT 1/8 SECOND
ANY FREE SPACE BELOW THE TASK?
NO
YES
SHUFFLE THE TASK. (THERE'S A LOT OF CODE HERE!)
CALL THE EXEC ($NXTSK) TO SEE IF THE WAITING TASK CAN FIT INTO MEMORY.
DECLARE PARTITION TO HAVE LONG OUTSTANDING I/O. DON'T SHUFFLE THE TASK.
CHECKPOINT ALL TASKS CONSIDERED. RESCHEDULE THE SHUFFLER FOR THIS PARTITION.
CONTINUE SUBPARTITION SCAN C
B FRESH PARTITION SCAN
A CONTINUE PARTITION SCAN

19-June-1975

Revised 15-June-1977

Revised 15-April-1981

1.Ø Scope

This document is a specification of the on-media structure that is used by Files-11. Files-11 is a general purpose file structure which is intended to be the standard file structure for all medium to large PDP-11 systems. Small systems such as RT-11 have been specifically excluded because the complexity of Files-11 would impose too great a burden on their simplicity and small size.

This document describes structure level 1 of Files-11, also referred to as ODS-1 (on-disk structure version 1). This has been implemented on the RSX-11 family, (RSX-11M, RSX-11M-PLUS, IAS, and RSX-11D) and on VMS. This document describes the final level of functionality for ODS-1. Structure level 2 (ODS-2) has been implemented on VMS and is the basis for all new disk structure enhancements.

1.1 Summary of revisions made to this specification

1. Expanded File Characteristics to include most ODS-2 options.
2. Corrected H.FPRO to H.DFPR.
3. Added new fields to home block for date and count of home block modifications.
4. Added Single Directory Support description and home block field.
5. Added field in home block for pack serial number (H.PKSR).
6. Added description of modified storage control block format to support large disks.
7. Restricted maximum number of blocks supported on a volume to 1,Ø44,48Ø.
8. Restricted ODS-1 to one block "clusters".
9. Restricted ODS-1 to single volume structures.
10. Clarified and expanded references to operating system support and relationship to ODS-2.
11. Removed RMS-11 definitions, to be provided in separate specification common to ODS-1 and ODS-2.

2.0 Medium

Files-11 is a structure which is imposed on a medium. That medium must have certain properties, which are described in the following section. Generally speaking, block addressable storage devices such as disks and Dectape are suitable for Files-11; hence Files-11 structured media are generically referred to as disks.

2.1 Volume

The basic medium that carries a Files-11 structure is referred to as a volume. A volume (also often referred to as a unit) is defined as an ordered set of logical blocks. A logical block is an array of 512 8-bit bytes. The logical blocks in a volume are consecutively numbered from 0 to n-1, where the volume contains n logical blocks. The number assigned to a logical block is called its logical block number, or LBN. Files-11 is theoretically capable of describing volumes up to 232 blocks in size. In practice, a volume should be at least 100 blocks in size to be useful; current implementations of Files-11 will handle volumes up to 224 blocks.

The logical blocks of a volume must be randomly addressable. The volume must also allow transfers of any length up to 65k bytes, in multiples of four bytes. When a transfer is longer than 512 bytes, consecutively numbered logical blocks are transfered until the byte count is satisfied. In other words, the volume can be viewed as a partitioned array of bytes. It must allow reads and writes of arrays of any length less than 65k bytes, provided that they start on a logical block boundary and that the length is a multiple of four bytes. When only part of a block is written, the contents of the remainder of that logical block will be undefined.

2.2 Volume Sets

This section is of historical interest only. ODS-1 does not and will not support volume sets. A volume set is a collection of related units that are normally treated as one logical device in the usual operating system concept. Each unit contains its own Files-11 structure; however, files on the various units in a volume set may be referenced with a relative volume number, which uniquely determines which unit in the set the file is located on. Other sections in this specification will make occasional reference to volume sets and relative volume numbers where hooks for their implementation exist. Since volume sets have not been implemented as yet, however, no complete specification is provided here.

3.0 Files

Any data in a volume or volume set that is of any interest (i.e., all blocks not available for allocation) is contained in a file. A file is an ordered set of virtual blocks, where a virtual block is an array of 512 8 bit bytes. The virtual blocks of a file are consecutively numbered from 1 to n, where n blocks have been allocated to the file. The number assigned to a virtual block is called (obviously) its virtual block number, or VBN. Each virtual block is mapped to a unique logical block in the volume set by Files-11. Virtual blocks may be processed in the same manner as logical blocks. Any array of bytes less than 65k in length may be read or written, provided that the transfer starts on a virtual block boundary and that its length is a multiple of four.

3.1 File ID

Each file in a volume set is uniquely identified by a File ID. A File ID is a binary value consisting of 48 bits (3 PDP-11 words). It is supplied by the file system when the file is created, and must be supplied by the user whenever he wishes to reference a particular file.

The three words of the File ID are used as follows:

Word 1 File Number

Locates the file within a particular unit of the volume set. File numbers must lie in the range 1 through 65535. The set of file numbers on a unit is moderately (but not totally) dense; at any instant in time a file number uniquely identifies one file within that unit.

Word 2 File Sequence Number

Identifies the current use of an individual file number on a unit. File numbers are re-used; when a file is deleted its file number becomes available for future use for some other file. Each time a file number is re-used, a different file sequence number is assigned to distinguish the uses of that file number. The file sequence number is essential since it is perfectly legal for users to remember and attempt to use a File ID long after that file has been deleted.

Word 3 Relative Volume Number

Identifies which unit of a volume set the file is located on. Volume sets are at present not implemented; the only legal value for the relative volume number in any context is zero.

3.2 File Header

Each file on a Files-11 volume is described by a file header. The file header is a block that contains all the information necessary to access the file. It is not part of the file; rather, it is contained in the volume's index file. (The index file is described in section 5.1). The header block is organized into four areas, of which the first three are variable in size.

3.2.1 Header Area

The information in the header area permits the file system to verify that this block is in fact a file header and, in particular, is the header being sought by the user. It contains the file number and file sequence number of the file, as well as its ownership and protection codes. This area also contains offsets to the other areas of the file header, thus defining their size. Finally, the header area contains a user attribute area, which may be used by the user to store a limited amount of data describing the file.

3.2.2 Ident Area

The ident area of a file header contains identification and accounting data about the file. Stored here are the primary name of the file, its creation date and time, revision count, date, and time, and expiration date.

3.2.3 Map Area

The map area describes the mapping of virtual blocks of the file to the logical blocks of the volume. The mapping data consists of a list of retrieval pointers. Each retrieval pointer describes one logically contiguous segment of the file. The map area also contains the linkage to the next extension header of the file, if such exists.

3.2.4 End Checksum

The last two bytes of the file header contain a 16 bit additive checksum of the remaining 255 words of the file header. The checksum is used to help verify that the block is in fact a file header.

3.3 Extension Headers

Since the file header is of fixed size, it is inevitable that for some files the mapping information will not fit in the allocated space. A file with a large amount of mapping data is therefore represented with a chain of file headers. Each header maps a consecutive set of virtual blocks; the extension linkage in the map area links the headers together in order of ascending virtual block numbers.

Multiple headers are also needed for files that span units in a volume set. A header may only map logical blocks located on its unit; therefore a multi-volume file is represented by headers on all units that contain portions of that file.

3.4 File Header - Detailed Description

This section describes in detail the items contained in the file header. Each item is identified by a symbol which represents the offset address of that item within its area in the file header. Any item may be located in the file header by locating the area to which it belongs and then adding the value of its offset address. Users who concern themselves with the contents of file headers are strongly urged to use the offset symbols. The symbols may be defined in assembly language programs by calling and invoking the macro FHDOF$, which may be found in the macro library of any system that supports Files-11. Alternatively, one may find the macro in the file F11MAC.MAC, which may be obtained from the author.

3.4.1 Header Area Description

The header area of the file header always starts at byte Ø. It contains the basic information needed for checking the validity of accesses to the file.

3.4.1.1 H.IDOF 1 Byte Ident Area Offset

This byte contains the number of 16 bit words between the start of the file header and the start of the ident area. It defines the location of the ident area and the size of the header area.

3.4.1.2 H.MPOF 1 Byte Map Area Offset

This byte contains the number of 16 bit words between the start of the file header and the start of the map area. It defines the location of the map area and, together with H.IDOF, the size of the ident area.

3.4.1.3	H.FNUM	2 Bytes	File Number

This word contains the file number of the file.

3.4.1.4	H.FSEQ	2 Bytes	File Sequence Number

This word contains the file sequence number of the file.

3.4.1.5	H.FLEV	2 Bytes	File Structure Level

The file structure level is used to identify different versions of Files-11 as they affect the structure of the file header. This permits upwards compatibility of file structures as Files-11 evolves, in that the structure level word identifies the version of Files-11 that created this particular file. This document describes version 1 of Files-11; the only legal contents for H.FLEV is 401 octal.

3.4.1.6	H.FOWN	2 Bytes	File Owner UIC
	H.PROG =	H.FOWN+0	Programmer (Member) Number
	H.PROJ =	H.FOWN+1	Project (Group) Number

This word contains the binary user identification code (UIC) of the owner of the file. The file owner is usually (but not necessarily) the creator of the file.

3.4.1.7	H.FPRO	2 Bytes	File Protection Code

This word controls what access all users in the system may have to the file. Accessors of a file are categorized according to the relationship between the UIC of the accessor and the UIC of the owner of the file. Each category is controlled by a four bit field in the protection word. The category of the accessor is selected as follows:

System Bits 0 - 3

The accessor is subject to system protection if the project number of the UIC under which he is running is 10 octal or less.

Owner Bits 4 - 7

The accessor is subject to owner protection if the UIC under which he is running exactly matches the file owner UIC.

Group Bits 8 - 11

The accessor is subject to group protection if the project number of his UIC matches the project number of the file owner UIC.

World Bits 12 - 15

The accessor is subject to world protection if he does not fit into any of the above categories.

Four types of access intents are defined in Files-11: read, write, extend, and delete. Each four bit field in the protection word is bit encoded to permit or deny any combination of the four types of access to that category of accessors. Setting a bit denies that type of access to that category. The bits are defined as follows (these values apply to a right-justified protection field):

FP.RDV	Deny read access
FP.WRV	Deny write access
FP.EXT	Deny extend access
FP.DEL	Deny delete access

When a user attempts to access a file, protection checks are performed in all the categories to which he is eligible, in the order system - owner - group - world. The user is granted access to the file if any of the categories to which he is eligible grants him access.

3.4.1.8	H.FCHA	2 Bytes	File Characteristics
	H.UCHA =	H.FCHA+0	User Controlled Char.
	H.SCHA =	H.FCHA+1	System Controlled Char.

The user controlled characteristics byte contains the following flag bits:

-	1 Bit, Reserved.
UC.NID	Set if incremental dump (backup) is to be disabled for this file.
UC.WBC	Set if the file is to be write-back cached; i.e., if a cache is used for the file data, data written by a user is only written back to the disk when is it removed from the cache. Clear for write-through cache operation.

UC.RCK	Set if the file is to be read-checked. All read operations on the file, including reads of the file header(s), will be performed with a read, read-compare to assure data integrity.
UC.WCK	Set if the file is to be write-checked. All write operations on the file, including modifications of the file header(s), will be performed with a write, read-compare to assure data integrity.
UC.CNB	Set if the file is allocated contiguous best effort; i.e., as contiguous as possible.
UC.DLK	Set if the file is deaccess-locked. This bit is used as a flag warning that the file was not properly closed and may contain inconsistent data. Access to the file is denied if this bit is set.
UC.CON	Set if the file is logically contiguous; i.e., if for all virtual blocks in the file, virtual block i maps to logical block k+i on one unit for some constant k. This bit may be implicitly set or cleared by file system operations that allocate space to the file; the user may only clear it explicitly.

The system controlled characteristics byte contains the following flag bits:

-	3 Bits, Reserved.
-	Reserved (Access Control List).
SC.SPL	Set if the file is an intermediate file for spooling.
SC.DIR	Set if the file is a directory.
SC.BAD	Set if there is a bad data block in the file. This bit is as yet unimplemented. It is intended for dynamic bad block handling.
SC.MDL	Set if the file is marked for delete. If this bit is set, further accesses to the file are denied, and the file will be physically deleted when no users are accessing it.

3.4.1.9 H.UFAT 32 Bytes User Attribute Area

This area is intended for the storage of a limited quantity of "user file attributes", i.e., any data the user deems useful for processing the file that is not part of the file itself. An example of the use of the user attribute area is presented in section 6.1 (FCS File Format).

3.4.1.10 S.HDHD 46 Bytes Size of Header Area

This symbol represents the total size of the header area containing all of the above entries.

3.4.2 Ident Area Description

The ident area of the file header begins at the word indicated by H.IDOF. It contains identification and accounting data about the file.

3.4.2.1 I.FNAM 6 Bytes File Name

These three words contain the name of the file, packed three Radix-50 characters to the word. This name usually, but not necessarily, corresponds to the name of the file's primary directory entry.

3.4.2.2 I.FTYP 2 Bytes File Type

This word contains the type of the file in the form of three Radix-50 characters.

3.4.2.3 I.FVER 2 Bytes Version Number

This word contains the version number of the file in binary form.

3.4.2.4 I.RVNO 2 Bytes Revision Number

This word contains the revision count of the file. The revision count is the number of times the file has been accessed for write.

3.4.2.5 I.RVDT 7 Bytes Revision Date

The revision date is the date on which the file was last deaccessed after being accessed for write. It is stored in ASCII in the form "DDMMMYY", where DD is two digits representing the day of the month, MMM is three characters representing the month, and YY is the last two digits of the year.

3.4.2.6 I.RVTI 6 Bytes Revision Time

The revision time is the time of day on which the file was last deaccessed after being accessed for write. It is stored in ASCII in the format "HHMMSS", where HH is the hour, MM is the minute, and SS is the second.

3.4.2.7 I.CRDT 7 Bytes Creation Date

These seven bytes contain the date on which the file was created. The format is the same as that of the revision date above.

3.4.2.8 I.CRTI 6 Bytes Creation Time

These six bytes contain the time of day at which the file was created. The format is the same as that of the revision time above.

3.4.2.9 I.EXDT 7 Bytes Expiration Date

These seven bytes contain the date on which the file becomes eligible to be deleted. The format is the same as that of the revision and creation dates above.

3.4.2.10 - 1 Byte (unused)

This unused byte is present to round up the size if the ident area to a word boundary.

3.4.2.11 S.IDHD 46 Bytes Size of Ident Area

This symbol represents the size of the ident area containing all of the above entries.

3.4.3 Map Area Description

The map area of the file header starts at the word indicated by H.MPOF. It contains the information necessary to map the virtual blocks of the file to the logical blocks of the volume.

3.4.3.1 M.ESQN 1 Byte Extension Segment Number

This byte contains the value n, where this header is the n+1th header of the file; i.e., headers of a file are numbered sequentially starting with Ø.

3.4.3.2 M.ERVN 1 Byte Extension Relative Volume No.

This byte contains the relative volume number of the unit in the volume set that contains the next sequential extension header for this file. If there is no extension header, or if the extension header is located on the same unit as this header, this byte contains Ø.

3.4.3.3 M.EFNU 2 Bytes Extension File Number

This word contains the file number of the next sequential extension header for this file. If there is no extension header, this word contains Ø.

3.4.3.4 M.EFSQ 2 Bytes Extension File Sequence Number

This word contains the file sequence number of the next sequential extension header for this file. If there is no extension header, this word contains Ø.

3.4.3.5 M.CTSZ 1 Byte Block Count Field Size

This byte contains a count of the number of bytes used to represent the count field in the retrieval pointers in the map area. The retrieval pointer format is described in section 3.4.3.9 below.

3.4.3.6 M.LBSZ 1 Byte LBN Field Size

This byte contains a count of the number of bytes used to represent the logical block number field in the retrieval pointers in the map area. The contents of M.CTSZ and M.LBSZ must add up to an even number.

3.4.3.7 M.USE 1 Byte Map Words In Use

This byte contains a count of the number of words in the map area that are presently occupied by retrieval pointers.

3.4.3.8 M.MAX 1 Byte Map Words Available

This byte contains the total number of words available for retrieval pointers in the map area.

3.4.3.9 M.RTRV variable Retrieval Pointers

This area contains the retrieval pointers that actually map the virtual blocks of the file to the logical blocks of the volume. Each retrieval pointer describes a consecutively numbered group of logical blocks which is part of the file. The count field contains the binary value n to represent a group of n+1 logical blocks. The logical block number field contains the logical block number of the first logical block in the group. Thus each retrieval pointer maps virtual blocks j through j+n into logical blocks k through k+n, respectively, where j is the total number plus one of virtual blocks represented by all preceding retrieval pointers in this and all preceding headers of the file, n is the value contained in the count field, and k is the value contained in the logical block number field.

Although the data in the map area provides for arbitrarily extensible retrieval pointer formats, Files-11 has defined only three. Of these, only the first is currently implemented; the other two are presented out of historical interest; they will never be supported.

Format 1: M.CTSZ = 1
M.LBSZ = 3

The total retrieval pointer length is four bytes. Byte 1 contains the high order bits of the 24 bit LBN. Byte 2 contains the count field, and bytes 3 and 4 contain the low 16 bits of the LBN.

```
|-----------|-----------|
|  Count    |   High    |
|-----------|-         -|
|     Low Order LBN     |
|-----------------------|
```

Format 2: M.CTSZ = 2
M.LBSZ = 2

The total retrieval pointer length is four bytes. The first word contains a 16 bit count field and the second word contains a 16 bit LBN field.

```
|-----------------------|
|         Count         |
|-----------------------|
|          LBN          |
|-----------------------|
```

Format 3: M.CTSZ = 2
M.LBSZ = 4

The total retrieval pointer length is six bytes. The first word contains a 16 bit count field and the second and third words contain a 32 bit LBN field.

```
|-----------------------|
|         Count         |
|-----------------------|
|      High             |
|--         LBN       --|
|      Low              |
|-----------------------|
```

3.4.3.1Ø S.MPHD 1Ø Bytes Size of Map Area

This symbol represents the size of the map area, not including the space used for the retrieval pointers.

3.4.4 End Checksum Description

The header check sum occupies the last two bytes of the file header. It is verified every time a header is read, and is recomputed every time a header is written.

3.4.4.1 H.CKSM 2 Bytes Block Checksum

This word is a simple additive checksum of all other words in the block. It is computed by the following PDP-11 routine or its equivalent:

```
        MOV     Header-address,RØ
        CLR     R1
        MOV     #255.,R2
1ØS:    ADD     (RØ)+,R1
        SOB     R2,1ØS
        MOV     R1,(RØ)
```

3.4.A File Header Layout

Header Area

```
        |--------------------|--------------------|
H.MPOF  | Map Area Offset    | Ident Area Offset  |   H.IDOF
        |--------------------|--------------------|
        |               File Number               |   H.FNUM
        |-----------------------------------------|
        |          File Sequence Number           |   H.FSEQ
        |-----------------------------------------|
        |          File Structure Level           |   H.FLEV
        |-----------------------------------------|   H.FOWN
H.PROJ  |             File Owner UIC              |   H.PROG
        |-----------------------------------------|
        |             File Protection             |   H.FPRO
        |--------------------|--------------------|   H.FCHA
H.SCHA  |    System Char.    |     User Char.     |   H.UCHA
        |--------------------|--------------------|
        |                                         |   H.UFAT
        |                                         |
        |                                         |
        |          User Attribute Area            |
        |                                         |
        |                                         |
        |                                         |
        |-----------------------------------------|   S.HDHD
```

Ident Area

```
        |-----------------------------------------|
        |                                         |   I.FNAM
        |--                                     --|
        |                File Name                |
        |--                                     --|
        |                                         |
        |-----------------------------------------|
        |                File Type                |   I.FTYP
        |-----------------------------------------|
        |             Version Number              |   I.FVER
        |-----------------------------------------|
        |             Revision Number             |   I.RVNO
        |-----------------------------------------|
        |                                         |   I.RVDT
        |--                                     --|
        |              Revision Date              |
        |--                                     --|
        |                                         |
        |--------------------|                  --|
I.RVTI  |                    |                    |
        |--                  |--------------------|
        |                                         |
        |--            Revision Time            --|
```

```
        |                                         |
        |--------------------|                  --|
I.CRDT  |                    |                    |
        |--                  |--------------------|
        |                                         |
        |--                                     --|
        |              Creation Date              |
        |--                                     --|
        |                                         |
        |-----------------------------------------|
        |                                         |   I.CRTI
        |--                                     --|
        |              Creation Time              |
        |--                                     --|
        |                                         |
        |-----------------------------------------|
        |                                         |   I.EXDT
        |--                                     --|
        |             Expiration Date             |
        |--                                     --|
        |                                         |
        |--------------------|                  --|
        |     (not used)     |                    |
        |--------------------|--------------------|   S.IDHD
```

Map Area

```
        |--------------------|--------------------|
M.ERVN  |   Extension RVN    |  Ext. Seg. Num.    |   M.ESQN
        |--------------------|--------------------|
        |          Extension File Number          |   M.EFNU
        |-----------------------------------------|
        |        Extension File Seq. Num.         |   M.EFSQ
        |--------------------|--------------------|
M.LBSZ  |   LBN Field Size   | Count Field Size   |   M.CTSZ
        |--------------------|--------------------|
M.MAX   | Map Words Avail.   | Map Words in Use   |   M.USE
        |--------------------|--------------------|   S.MPHD
        |                                         |   M.RTRV
        |                                         |
        |                                         |
        |           Retrieval Pointers            |
        |                                         |
        |                                         |
        |                                         |
        |-----------------------------------------|
        |          File Header Checksum           |   H.CKSM
        |-----------------------------------------|
```

4.0 Directories

Files-11 provides directories to allow the organization of files in a meaningful way. While the File ID is sufficient to locate a file uniquely on a volume set, it is hardly mnemonic. Directories are files whose sole function is to associate file name strings with File ID's.

4.1 Directory Heirarchies

Since directories are files with no special attributes, directories may list files that are in turn directories. Thus the user may construct directory heirarchies of arbitrary depth and complexity to structure his files as he pleases.

4.1.1 User File Directories

Current implementations of Files-11 all support a two level directory heirarchy which is tied in with the user identification mechanism of the operating system. Each UIC is associated with a user file directory (UFD). References to files that do not specify a directory are generally defaulted to the UFD associated with the user's UIC. All UFD's are listed in the volume's MFD under a file name constructed from the UIC. A UIC of [n,m] associates with a directory name of "nnnmmm.DIR;1", where nnn and mmm are n and m padded out to three digits each with leading zeroes. Note that all number conversions are done in octal.

Two points should be noted here. The UFD structure described here is not intrinsically part of the Files-11 on-disk structure; rather, it is a convenient cataloging system applied by various operating systems. Also, there is no hard and fast relationship between the owner UIC of a file and the UFD in which it is listed. Generally, they will correspond, but not necessarily.

4.2 Directory Structure

A directory is a file consisting of 16 byte records. It is structured as an FCS fixed length record file, with no carriage control attributes (see section 6 for a description of FCS files). Each record is a directory entry. The entries are not required to be ordered, or densely packed, nor do they have any other relationship to each other, except that no two entries in one directory may contain the same name, type, and version. Each entry contains the following:

File ID	The three word binary File ID of the file that this directory entry represents. If the file number portion of the File ID field is zero, then this record is empty and may be used for a new directory entry.
Name	The name of the file may be up to 9 characters. It is stored as three words, each containing three Radix-50 packed characters.
Type	The type of the file (also historically referred to as the extension) may be up to three characters. It is stored as one word of Radix-50 packed characters.
Version	The version number of the file is stored in binary in one word.

```
|----------------------|
|                      |
|--                  --|
|        File ID       |
|--                  --|
|                      |
|----------------------|
|                      |
|--                  --|
|         Name         |
|--                  --|
|                      |
|----------------------|
|         Type         |
|----------------------|
|        Version       |
|----------------------|
```

4.3 Directory Protection

Since directories are files with no special characteristics, they may be accessed like all other files, and are subject to the same protection mechanism. However, implementations of Files-11 support three special functions for the management of directories, namely FIND, REMOVE, and ENTER. A user performing such a directory operation must have the following privileges to be allowed the various functions:

Find:	READ
Remove:	READ, WRITE
Enter:	READ, WRITE

Note that the same privilege is required for both enter and remove. The recovery for an operation that involves a remove at the beginning of the sequence is an enter.

5.0 Known Files

Clearly any file system must maintain some data structure on the medium which is used to control the file organization. In Files-11 this data is kept in five files. These files are created when a new volume is initialized. They are unique in that their File ID's are known constants. These five files have the following uses:

File ID 1,1,0 is the index file. The index file is the root of the entire Files-11 structure. It contains the volume's bootstrap block and the home block, which is used to identify the volume and locate the rest of the file structure. The index file also contains all of the file headers for the volume, and a bitmap to control the allocation of file headers.

File ID 2,2,0 is the storage bitmap file. It is used to control the allocation of logical blocks on the volume.

File ID 3,3,0 is the bad block file. It is a file containing all of the known bad blocks on the volume.

File ID 4,4,0 is the volume master file directory (or MFD). It forms the root of the volume's directory structure. The MFD lists the five known files, all first level user directories, and whatever other files the user chooses to enter.

File ID 5,5,0 is the system core image file. Its use is operating system dependent; its basic purpose is to provide a file of known File ID for the use of the operating system.

5.1 Index File

The index file is File ID 1,1,0. It is listed in the MFD as INDEXF.SYS;1. The index file is the root of the Files-11 structure in that it provides the means for identification and initial access to a Files-11 volume, and contains the access data for all files on the volume (including itself).

5.1.1 Bootstrap Block

Virtual block 1 of the index file is the volume's boot block. It is always mapped to logical block 0 of the volume. If the volume is the system device of an operating system, the boot block contains an operating system dependent program which reads the operating system into memory when the boot block is read and executed by a machine's hardware bootstrap. If the volume is not a system device, the boot block contains a small program that outputs

a message on the system console to inform the operator to that effect.

5.1.2 Home Block

Virtual block 2 of the index file is the volume's home block. The logical block containing the home block is the first good block on the volume out of the sequence 1, 256, 512, 768, 1024, 1280, ... 256n. The purpose of the home block is to identify the volume as Files-11, establish the specific identity of the volume, and serve as the ground zero entry point into the volume's file structure. The home block is recognized as a home block by the presence of checksums in known places and by the presence of predictable values in certain locations.

Items contained in the home block are identified by symbolic offsets in the same manner as items in the file header. The symbols may be defined in assembly language programs by calling and invoking the macro HMBOF$, which may be found in the macro library of any system that supports Files-11. Alternatively, one may find the macro in the file F11MAC.MAC, which is available from the author.

5.1.2.1 H.IBSZ 2 Bytes Index File Bitmap Size

This 16 bit word contains the number of blocks that make up the index file bitmap. (The index file bitmap is discussed in section 5.1.3.) This value must be non-zero for a valid home block.

5.1.2.2 H.IBLB 4 Bytes Index File Bitmap LBN

This double word contains the starting logical block address of the index file bitmap. Once the home block of a volume has been found, it is this value that provides access to the rest of the index file and to the volume. The LBN is stored with the high order in the first 16 bits, followed by the low order portion. This value must be non-zero for a valid home block.

5.1.2.3 H.FMAX 2 Bytes Maximum Number of Files

This word contains the maximum number of files that may be present on the volume at any time. This value must be non-zero for a valid home block.

5.1.2.4 H.SBCL 2 Bytes Storage Bitmap Cluster Factor

This word contains the cluster factor used in the storage bitmap file. The cluster factor is the number of blocks represented by each bit in the storage bitmap. Volume clustering can not implemented in ODS-1; the only legal value for this item is 1.

5.1.2.5 H.DVTY 2 Bytes Disk Device Type

This word is an index identifying the type of disk that contains this volume. It is currently not used and always contains 0.

5.1.2.6 H.VLEV 2 Bytes Volume Structure Level

This word identifies the volume's structure level. Like the file structure level, this word identifies the version of Files-11 which created this volume and permits upwards compatibility of media as Files-11 evolves. The volume structure level is affected by all portions of the Files-11 structure except the contents of the file header. This document describes Files-11 version 1; the only legal values for the structure level are 401 and 402 octal. The former (401) is the standard value for most volumes. The latter (402) is an advisory that the volume contains a multiheader index file. (A multiheader index file is required to support more than about 26,000 files. The index file may in fact be multiheader without the volume having a structure level of 402).

5.1.2.7 H.VNAM 12 Bytes Volume Name

This area contains the volume label as an ASCII string. It is padded out to 12 bytes with nulls. The volume label is used to identify individual Files-11 volumes.

5.1.2.8 - 4 Bytes Not Used

5.1.2.9 H.VOWN 2 Bytes Volume Owner UIC

This word contains the binary UIC of the owner of the volume. The format is the same as that of the file owner UIC stored in the file header.

5.1.2.10 H.VPRO 2 Bytes Volume Protection Code

This word contains the protection code for the entire volume. Its contents are coded in the same manner as the file protection code stored in the file header, and it is interpreted in the same way in conjunction with the volume owner UIC. All operations on all files on the volume must pass both the volume and the file protection check to be permitted. (Refer to the discussion on file protection in section 3.4.1.7).

5.1.2.11 H.VCHA 2 Bytes Volume Characteristics

This word contains bits which provide additional control over access to the volume. The following bits are defined:

CH.NDC	Obsolete, used by RSX-11D/IAS. Set if device control functions are not permitted on this volume. Device control functions are those which can threaten the integrity of the volume, such as direct reading and writing of logical blocks, etc.
CH.NAT	Obsolete, used by RSX-11D/IAS. Set if the volume may not be attached, i.e., reserved for the sole use by one task.
CH.SDI	Set if the volume contains only a single directory. If this bit is set, no directories should be created on the volume other than the MFD. The access methods should also be informed of this situation, e.g. by setting the DV.SDI bit in the device characteristics word.

5.1.2.12 H.DFPR 2 Bytes Default File Protection

This word contains the file protection that will be assigned to all files created on this volume if no file protection is specified by the user.

5.1.2.13 - 6 Bytes Not Used

5.1.2.14 H.WISZ 1 Byte Default Window Size

This byte contains the number of retrieval pointers that will be used for the "window" (in core file access data) when files are accessed on the volume, if not otherwise specified by the accessor.

5.1.2.15 H.FIEX 1 Byte Default File Extend

This byte contains the number of blocks that will be allocated to a file when a user extends the file and asks for the system default value for allocation.

5.1.2.16 H.LRUC 1 Byte Directory Pre-access Limit

This byte contains a count of the number of directories to be stored in the file system's directory access cache. More generally, it is an estimate of the number of concurrent users of the volume and its use may be generalized in the future.

5.1.2.17 H.REVD 7 Bytes Date of Last Home Block Revision

This field ill defined field is in the standard ASCII date format and reflects the date of the last modifications to fields in the home block.

5.1.2.17 H.REVC 2 Bytes Count of Home Block Revisions

This field reflects the number of above mentioned modifications.

5.1.2.17 - 2 Bytes Not Used

5.1.2.18 H.CHK1 2 Bytes First Checksum

This word is an additive checksum of all entries preceding in the home block (i.e., all those listed above). It is computed by the same sort of algorithm as the file header checksum (see section 3.4.4.1).

5.1.2.19 H.VDAT 14 Bytes Volume Creation Date

This area contains the date and time that the volume was initialized. It is in the format "DDMMMYYHHMMSS", followed followed by a single null. (The same format is used in the ident area of the file header, section 3.4.2).

5.1.2.2Ø - 382 Bytes Not Used

This area is reserved for the relative volume table for volume sets. This field will not be used, although some versions of DSC referenced this area.

5.1.2.21 H.PKSR 4 Bytes Pack Serial Number

This area contains the manufacturer supplied serial number for the physical volume. For last track devices, the pack serial number is contained on the volume in the manufacturer data. For other devices the user must supply this information manually. The serial number is contained in the home block for convenience and consistency.

5.1.2.22 - 12 Bytes Not Used

5.1.2.23 H.INDN 12 Bytes Volume Name

This area contains another copy of the ASCII volume label. It is padded out to 12 bytes with spaces. It is placed here in accordance with the volume identification standard (STD 167).

5.1.2.24 H.INDO 12 Bytes Volume Owner

This area contains an ASCII expansion of the volume owner UIC in the form "[proj,prog]". Both numbers are expressed in decimal and are padded to three digits with leading zeroes. The area is padded out to 12 bytes with trailing spaces. It is placed here in accordance with the volume identification standard (STD 167).

5.1.2.25 H.INDF 12 Bytes Format Type

This field contains the ASCII string "DECFILE11A" padded out to 12 bytes with spaces. It identifies the volume as being of Files-11 format. It is placed here in accordance with the volume identification standard (STD 167).

5.1.2.26 - 2 Bytes Not Used

5.1.2.27 H.CHK2 2 Bytes Second Checksum

This word is the last word of the home block. It contains an additive checksum of the preceding 255 words of the home block, computed according to the algorithm listed in section 3.4.4.1.

5.1.2.A   Home Block Layout

```
                |-----------------------------------------|
                |         Index File Bitmap Size          |    H.IBSZ
                |-----------------------------------------|
                |               Index File                |    H.IBLB
                |--                                     --|
                |               Bitmap LBN                |
                |-----------------------------------------|
                |        Maximum Number of Files          |    H.FMAX
                |-----------------------------------------|
                |     Storage Bitmap Cluster Factor       |    H.SBCL
                |-----------------------------------------|
                |           Disk Device Type              |    H.DVTY
                |-----------------------------------------|
                |        Volume Structure Level           |    H.VLEV
                |-----------------------------------------|
                |                                         |    H.VNAM
                |--                                     --|
                |                                         |
                |--                                     --|
                |              Volume Name                |
                |--                                     --|
                |                                         |
                |--                                     --|
                |                                         |
                |--                                     --|
                |                                         |
                |-----------------------------------------|
                |                                         |
                |--             (not used)              --|
                |                                         |
                |-----------------------------------------|
                |           Volume Owner UIC              |    H.VOWN
                |-----------------------------------------|
                |           Volume Protection             |    H.VPRO
                |-----------------------------------------|
                |        Volume Characteristics           |    H.VCHA
                |-----------------------------------------|
                |        Default File Protection          |    H.DFPR
                |-----------------------------------------|
                |                                         |
                |--                                     --|
                |               (not used)                |
                |--                                     --|
                |                                         |
                |-------------------|---------------------|
H.FIEX          | Def. File Extend  | Def. Window Size    |    H.WISZ
                |-------------------|---------------------|
H.REVD          |                   |  Directory Limit    |    H.LRUC
                |--                 |---------------------|
                |                                         |
                |--                                     --|
                |        Volume Modification Date         |
```

```
                |--                                     --|
                |                                         |
                |-----------------------------------------|
                |       Volume Modification Count         |    H.REVC
                |-----------------------------------------|
                |               (not used)                |
                |-----------------------------------------|
                |             First Checksum              |    H.CHK1
                |-----------------------------------------|
                |                                         |    H.VDAT
                |--                                     --|
                |                                         |
                |--                                     --|
                |                                         |
                |--                                     --|
                |          Volume Creation Date           |
                |--                                     --|
                |                                         |
                |--                                     --|
                |                                         |
                |--                                     --|
                |                                         |
                |-----------------------------------------|
                |                                         |
                |                                         |
                |                                         |
                |               (not used)                |
                |                                         |
                |                                         |
                |                                         |
                |-----------------------------------------|
                |                                         |    H.PKSR
                |--         Pack Serial Number          --|
                |                                         |
                |-----------------------------------------|
                |-----------------------------------------|
                |                                         |
                |--                                     --|
                |                                         |
                |--                                     --|
                |               (not used)                |
                |--                                     --|
                |                                         |
                |--                                     --|
                |                                         |
                |--                                     --|
                |                                         |
                |-----------------------------------------|
                |                                         |    H.INDN
                |--                                     --|
                |                                         |
                |--                                     --|
                |              Volume Name                |
     |--                                     --|
     |                                         |
     |--                                     --|
     |                                         |
     |--                                     --|
     |                                         |
     |-----------------------------------------|
     |                                         |     H.INDO
     |--                                     --|
     |                                         |
     |--                                     --|
     |               Volume Owner              |
     |--                                     --|
     |                                         |
     |--                                     --|
     |                                         |
     |--                                     --|
     |                                         |
     |-----------------------------------------|
     |                                         |     H.INDF
     |--                                     --|
     |                                         |
     |--                                     --|
     |               Format Type               |
     |--                                     --|
     |                                         |
     |--                                     --|
     |                                         |
     |--                                     --|
     |                                         |
     |-----------------------------------------|
     |               (not used)                |
     |-----------------------------------------|
     |             Second Checksum             |     H.CHK2
     |-----------------------------------------|
```

5.1.3 Index File Bitmap

The index file bitmap is used to control the allocation of file numbers (and hence file headers). It is simply a bit string of length n, where n is the maximum number of files permitted on the volume (contained in offset H.FMAX in the home block). The bitmap spans over as many blocks as is necessary to hold it, i.e., max number of files divided by 4096 and rounded up. The number of blocks in the bitmap is contained in offset H.IBSZ of the home block.

The bits in the index file bitmap are numbered sequentially from Ø to n-1 in the obvious manner, i.e., from right to left in each byte, and in order of increasing byte address. Bit j is used to represent file number j+1: if the bit is 1, then that file number is in use; if the bit is Ø, then that file number is not in use and may be assigned to a newly created file.

The index file bitmap starts at virtual block 3 of the index file and continues through VBN 2+m, where m is the number of blocks in the bitmap. It is located at the logical block indicated by offset H.IBLB in the home block.

5.1.4 File Headers

The rest of the index file contains all the file headers for the volume. The first 16 file headers (for file numbers 1 to 16) are logically contiguous with the index file bitmap to facilitate their location; the rest may be allocated wherever the file system sees fit. Thus the first 16 file headers may be located from data in the home block (H.IBSZ and H.IBLB) while the rest must be located through the mapping data in the index file header. The file header for file number n is located at virtual block 2+m+n (where m is the number of blocks in the index file bitmap).

5.2 Storage Bitmap File

The storage bitmap file is File ID 2,2,Ø. It is listed in the MFD as BITMAP.SYS;1. The storage bitmap is used to control the available space on a unit. It consists of a storage control block which contains summary information about the unit, and the bitmap itself which lists the availablilty of individual blocks.

5.2.1 Storage Control Block

Virtual block 1 of the storage bitmap is the storage control block. It contains summary information intended to optimize allocation of space on the unit. The storage control block has the following format for disks with less than 4Ø96126, (516,Ø96 blocks):

```
(3 bytes) Not used (zero)
(1 byte)  Number of storage bitmap blocks (less than 127)
(2 bytes) Number of free blocks in 1st bitmap block
(2 bytes) Free block pointer in 1st bitmap block
                    .
                    .
                    .
(2 bytes) Number of free blocks in nth bitmap block
(2 bytes) Free block pointer in nth bitmap block
(4 bytes) Size of the unit in logical blocks
```

For larger disks the following format is used:

```
(3 bytes) Not used (zero)
(1 byte)  Number of storage bitmap blocks (greater than 126)
(4 bytes) Size of the unit in logical blocks
(246 bytes)         Not used (zero)
```

Note: Current implementations of Files-11 do not correctly initialize the word pairs containing number of free blocks and free block pointer for each bitmap block, nor are these values maintained as space is allocated and freed on the unit. They are therefore best looked upon as 2n garbage words and should not be used by future implementations of Files-11 until the disk structure is formally updated.

5.2.2 Storage Bitmap

Virtual blocks 2 through n+1 are the storage bitmap itself. It is best viewed as a bit string of length m, numbered from 0 to m-1, where m is the total number of logical blocks on the unit rounded up to the next multiple of 4096. The bits are addressed in the usual manner (packed right to left in sequentially numbered bytes). Since each virtual block holds 4096 bits, n blocks, where n = m/4096, are used to hold the bitmap. Bit j of the bitmap represents logical block j of the volume; if the bit is set, the block is free; if clear, the block is allocated. Clearly the last k bits of the bitmap are always clear, where k is the difference between the true size of the volume and m, the length of the bitmap.

The size of the bitmap is limited to 256 blocks. In fact, due to existing implementations on all RSX systems, the retrieval pointers must be in one of the following two forms:

1. A single retrieval pointer mapping the entire BITMAP.SYS file.

2. Two retrieval pointers, the first mapping the storage control block only, and the second mapping the entire bitmap proper.

This restriction limits ODS-1 to a volume of 4096255 blocks (1,044,480 blocks or about 500 megabytes).

5.3 Bad Block File

The bad block file is File ID 3,3,0. It is listed in the MFD as BADBLK.SYS;1. The bad block file is simply a file containing all of the known bad blocks on the volume.

5.3.1 Bad Block Descriptor

Virtual block 1 of the bad block file is the bad block descriptor for the volume. It is always located on the last good block of the volume. This block may contain a listing of the bad blocks on the volume produced by a bad block scan program or diagnostic. The format of the bad block data is

identical to the map area of a file header, except that the first four entries (M.ESQN, M.ERVN, M.EFNU, and M.EFSQ) are not present. The last word of the block contains the usual additive checksum. (See section 3.4.3 for a description of the map area.) This block is included in the bad block file to save the data it contains for future re-initialization of the volume.

Bad Block Descriptor Layout

```
|--------------------|--------------------|
|  LBN Field Size    | Count Field Size   |
|--------------------|--------------------|
| Map Words Avail.   | Map Words in Use   |
|--------------------|--------------------|
|                                         |
|                                         |
|                                         |
|           Retrieval Pointers            |
|                                         |
|                                         |
|                                         |
|-----------------------------------------|
|             Block Checksum              |
|-----------------------------------------|
```

5.4 Master File Directory

The master file directory is File ID 4,4,0. It is listed in the MFD (itself) as 000000.DIR;1. The MFD is the root of the volume's directory structure. It lists the five known files, plus whatever the user chooses to enter. In the two level UFD structure described in section 4.1.1, the MFD contains entries for all user file directories.

5.5 Core Image File

The core image file is File ID 5,5,0. It is listed in the MFD as CORIMG.SYS;1. Its use is operating system dependent. In general, it provides a file of known File ID for the use of the operating system, for use as a swap area, for example, or as a monitor overlay area, etc.

6.0 FCS File Structure

File Control Services (FCS) is a user level interface to Files-11. Its principal feature is a record control facility that allows sequential processing of variable length records and sequential and random access to fixed length record files. FCS interfaces to the virtual block facility provided by the basic Files-11 structure.

6.1 FCS File Attributes

FCS stores attribute information about the file in the file's user attribute area (H.UFAT - see section 3.4.1.9). It uses only the first 7 words; the rest are ignored by FCS, but are reserved by DEC. (RMS uses an additional 3 words, 10 words in all, for relative and indexed file attributes.) The following items are contained in the attribute area; they are identified by the usual symbolic offsets (relative to the start of the attribute area). The offsets may be defined in assembly language programs by calling and invoking the macro FDOFF$ DEF$L. Flag values and bits may be defined by calling and invoking the macro FCSBT$. These macros are in the system macro library of any operating system that supports Files-11. Alternatively, all these values are defined in the system object library of any system that supports Files-11, and may be obtained at link time.

6.1.1 F.RTYP 1 Byte Record Type

This byte identifies which type of records are contained in this file. The following three values are legal:

R.FIX	Fixed length records.
R.VAR	Variable length records.
R.SEQ	Sequenced Variable Length records

6.1.2 F.RATT 1 Byte Record Attributes

This byte contains record attribute bits that control the handling of records under various contexts. The following flag bits are defined:

FD.FTN	Use Fortran carriage control if set. The first byte of each record is to be interpreted as a standard Fortran carriage control character when the record is copied to a carriage control device.
FD.CR	Use implied carriage control if set. When the file is copied to a carriage control device, each record is to be preceded by a line feed and followed by a carriage return. Note that the FD.FTN and FD.CR bits are mutually exclusive.
FD.PRN	Used to indicate that the two byte sequence number field for R.SEQ record format is to be interpreted as print control information (see Section 6.2.3.1 for format of print information).
FD.BLK	Records do not cross block boundaries if set. Generally, there will be dead space at the end of each block; how this is handled is explained in the description of record formats in section 6.2.

6.1.3 F.RSIZ 2 Bytes Record Size

In a fixed length record file, this word contains the size of the records in bytes. In a variable or sequenced variable length record file, this word contains the size in bytes of the longest record in the file.

6.1.4 F.HIBK 4 Bytes Highest VBN Allocated

This 32 bit number is a count of the number of virtual blocks allocated to the file. Since this value is maintained by FCS, it is usually correct, but it is not guaranteed since FCS is a user level package.

6.1.5 F.EFBK 4 Bytes End of File Block

This 32 bit number is the VBN in which the end of file is located. Both F.HIBK and F.EFBK are stored with the high order half in the first two bytes, followed by the low order half.

6.1.6 F.FFBY 2 Bytes First Free Byte

This word is a count of the number of bytes in use in the virtual block containing the end of file; i.e., it is the offset to the first byte of the file available for appending. Note that an end of file that falls on a block boundary may be represented in either of two ways. If the file contains precisely n blocks, F.EFBK may contain n and F.FFBY will contain 512, or F.EFBK may contain n+1 and F.FFBY will contain 0.

6.1.7 S.FATT 14 Bytes Size of Attribute Block

This symbol represents the total number of bytes in the FCS file attribute block.

6.1.A FCS File Attributes Layout

```
|--------------------|--------------------|
F.RATT    |  Record Attr.    |   Record Type    |   F.RTYP
          |------------------|------------------|
          |       Record Size (Bytes)           |   F.RSIZ
          |-------------------------------------|
          |           Highest VBN               |   F.HIBK
          |--                                 --|
          |            Allocated                |
          |-------------------------------------|
          |           End of File               |   F.EFBK
          |--                                 --|
          |               VBN                   |
          |-------------------------------------|
          |         First Free Byte             |   F.FFBY
          |-------------------------------------|   S.FATT
```

6.2 Record Structure

This section describes how records are packed in the virtual blocks of a disk file. In general, FCS treats a disk file as a sequentially numbered array of bytes. Records are numbered consecutively starting with 1.

6.2.1 Fixed Length Records

In a file consisting of fixed length records, the records are simply packed end to end with no additional control information. If the record length is odd, each record is padded with a single byte. The content of the pad byte is undefined. For direct access, the address of a record is computed as follows:

```
Let:      n = record number
          k = record size (in bytes)
          m = byte address of record in file
          q = number of records per block
          j = VBN containing the start of the record
          i = byte offset within VBN j

then      h = ((k+1)/2)2 (rounded up record length)
          m = (n-1)h
          j = m/512+1 (truncated)
          i = m mod 512
```

The previous discussion assumes that records cross block boundaries (that is, FD.BLK is not set). If records do not cross block boundaries, they are limited to 512 bytes, and the following equations apply (the variables are defined as above):

```
          h = ((k+1)/2)2 (rounded up record length)
          q = 512/k (truncated)
          j = (n-1)/q+1 (truncated)
          i = ((n-1) mod q)h
```

6.2.2 Variable Length Records

In a file consisting of variable length records, records may be up to 32767 bytes in length. Each record is preceded by a two byte binary count of the bytes in the record (the count does not include itself). For example, a null record is represented by a single zero word. The byte count is always word aligned; i.e., if a record ends on an odd byte boundary, it is padded with a single byte. The content of the pad byte is undefined.

If records do not cross block boundaries (FD.BLK is set), they are limited to a size of 510 bytes. A byte count of -1 is used as a flag to signal that there are no more records in a particular block. The remainder of that block is then dead space and the next record in the file starts at the beginning of the next block.

6.2.3 Sequenced Variable Length Records

The format of a sequenced file is identical to a variable length record file except that a two byte sequence number field is located immediately after the byte count field of each record. This field contains a binary value which is usually interpreted as the line number of that record (see Section 6.1.2 FD.PRN and Section 6.2.3.1). The sequence number is not returned as part of the data when a record is read, but is available separately. Note that the record byte count field counts the sequence number field as well as the data of the record.

6.2.3.1 Format of Two Byte Print Control Field in R.SEQ Records

If the FD.PRN bit is set in the record attribute then the two byte "sequence number" field is used to contain carriage control data for the record. Byte 0 is print control information to act upon before the record data is output to a unit record device; byte 1 is print control information to act upon after the record data has been output to a unit record device.

The format of each byte is as follows:

Bit 7	Bits 6-0	Meaning
0	0	No carriage control
0	count(1-127)	"count" new lines (CR/LF)

Bit 7	Bit 6	Bit 5	Bits 4-0	Meaning

1	Ø	Ø	ASCII CØ set	ASCII char to output (CR,FF etc.)
1	Ø	1	ASCII C1 set	ASCII char (8 bit code) to output
1	1	Ø	CODE (Ø-63)	Device specific code
1	1	1	-	Reserved

NOTE

The print control field is not currently supported by FCS or RMS-11.