LOCAL INFORMATION RETRIEVAL FOR THE SIMULATION AND MODELING SYSTEM

by

HAROLD LEVIN

August 1970

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN • URBANA, ILLINOIS
DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.
DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.
LOCAL INFORMATION RETRIEVAL FOR THE SIMULATION AND MODELING SYSTEM*

by

HAROLD LEVIN

August 1970

Department of Computer Science
University of Illinois
Urbana, Illinois 61801

* This report was supported in part by the Atomic Energy Commission under grant US AEC AT(11-1)1469 and submitted to the Graduate College of the University of Illinois at Urbana-Champaign in partial fulfillment of the requirements for the degree of Master of Science in Computer Science, 1970.
ACKNOWLEDGMENT

The author wishes to express his appreciation to Professor C. W. Gear, who advised this thesis; and to Martin J. Michel, who provided many interesting ideas during its development.

Thanks are also due to Miss Barbara Hurdle for her excellent typing, and to the author's wife for her endless patience.
TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>LIST OF FIGURES</th>
<th>v</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2. IR FACILITIES</td>
<td>3</td>
</tr>
<tr>
<td>3. LIBRARY FUNCTIONAL DESCRIPTION</td>
<td>7</td>
</tr>
<tr>
<td>LIST OF REFERENCES</td>
<td>17</td>
</tr>
</tbody>
</table>

APPENDICES

| A. IR INTERNAL ROUTINES | 18 |
| B. DISK STORAGE ALLOCATION | 23 |
| C. DISK MONITOR SYSTEM | 26 |
### LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Simplified System Architecture</td>
<td>1</td>
</tr>
<tr>
<td>2.</td>
<td>IR Function Table</td>
<td>5</td>
</tr>
<tr>
<td>3.</td>
<td>File Structure</td>
<td>9</td>
</tr>
<tr>
<td>4.</td>
<td>Free Block Lists</td>
<td>11</td>
</tr>
<tr>
<td>5.</td>
<td>IRREQ Flow Diagram</td>
<td>12</td>
</tr>
<tr>
<td>6.</td>
<td>Read and Save Flow Diagrams</td>
<td>13</td>
</tr>
<tr>
<td>7.</td>
<td>Delete Flow Diagram</td>
<td>14</td>
</tr>
<tr>
<td>8.</td>
<td>Write Flow Diagram (Part 1)</td>
<td>15</td>
</tr>
<tr>
<td>9.</td>
<td>Write Flow Diagram (Part 2)</td>
<td>16</td>
</tr>
<tr>
<td>B-1.</td>
<td>Disk Storage Map</td>
<td>25</td>
</tr>
<tr>
<td>C-1.</td>
<td>SWAPTAPE Format (Block Numbers)</td>
<td>28</td>
</tr>
</tbody>
</table>
1. INTRODUCTION

A group in the Department of Computer Science at the University of Illinois is currently developing a general purpose simulation and modeling system. Figure 1 shows a simplified picture of the system architecture.

Figure 1. Simplified System Architecture
The PDP-8 is used to handle all drawing functions, and together with the terminal control and the graphics terminals, allows a user to draw various networks, describe their functional characteristics, save the networks for later use, or request network analysis. The PDP-8 also maintains a temporary library of user pictures and networks on the local disk, while a permanent library is maintained in the 360. Extensive network analysis is also performed in the 360.

This paper describes one component of the PDP-8 software system, namely, the IR (Information Retrieval) routines. These routines are responsible for:

1. Performing all disk operations for GLASP.\(^2,3\)
2. Maintaining the local library on the disk.
3. Communicating with IR in the 360.

Chapters 2 and 3 describe the functional characteristics of IR. Appendix A contains descriptions of the internal routines in IR, Appendix B describes the disk layout for IR, while Appendix C describes the Disk Monitor System that was implemented for program development on the PDP-8.
2. IR FACILITIES

A. GLASP Utilities

Two programs are contained in IR to perform utility operations for GLASP.

1. BNKSWP

This program is used to save and restore core banks 0 and 1 on the disk. Eight such core images can be saved, one for each of the eight terminals that GLASP supports. Due to the nature of the disk, two banks of core may be transferred in one disk access. Transfer time (including a half rotation average latency) is 61 ms. in either direction.

2. DSKGET

A collection of numbered program segments is maintained on the disk. Each program segment is $1024_{10}$ words long and always resides at $34000_{8}$ in the PDP-8. GLASP may read or write any particular segment by calling DSKGET. Transfer time (including latency) is 22.3 ms.

B. Library System

The principle function of IR is to build and maintain the local library. Programs running under GLASP will need facilities for saving and retrieving subpictures, textual data, and network descriptions created by users at the graphics terminals. IR is insensitive to the type of data, and merely handles blocks of PDP-8 words along with a name and type supplied by the calling program. The system is coded in such a way as to reduce disk references and to reduce latency times when reading and writing files.
Communications between programs and the IR library is through parameters stored in the console vector, and via ROUTX. Specifically, a program describes the desired operation by setting up words 5-12 in the console vector of the current console, and goes to ROUTX with a request for IR. GLASP keeps track of all system resources and will transfer control to IR if IR is not busy. A location in core points to the current console vector. IR will service the request, put a return code indicating the degree of success into the console vector, and return to the program via GLASP utility OPDONE.

The console vector parameters for IR are shown below:

<table>
<thead>
<tr>
<th>WORD</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>FUNCTION</td>
</tr>
<tr>
<td>6</td>
<td>6 BIT FILE TYPE</td>
</tr>
<tr>
<td>7-9</td>
<td>6 CHARACTER FILE NAME (packed ASCII)</td>
</tr>
<tr>
<td>10</td>
<td>CORE LOCATION OF DATA</td>
</tr>
<tr>
<td>11</td>
<td>LENGTH OF FILE</td>
</tr>
<tr>
<td>12</td>
<td>PROTECTION</td>
</tr>
</tbody>
</table>

The FUNCTION word decodes as:

- BIT 0: 0 - write
  1 - read
- BIT 1: 0 - communicate with the 360 where appropriate
  1 - no 360 communications
- BIT 2: 0 - save file locally
  1 - do not save locally
- BIT 3: 0 - if a file to be written already exists, overlay the old copy
  1 - do not overlay an existing file
BIT 4:

0 - no action

1 - delete the named file

BITS 5-6:

core bank of data

BITS 7-11:

return code placed here

Common combinations of the function bits are tabulated below

(X is a "don't care").

<table>
<thead>
<tr>
<th>FUNCTION BIT</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4</td>
<td></td>
</tr>
<tr>
<td>X 0 X X 1</td>
<td>Delete locally and in 360.</td>
</tr>
<tr>
<td>X 1 X X 1</td>
<td>Delete locally only.</td>
</tr>
<tr>
<td>1 0 0 X 0</td>
<td>Read locally or from 360, save a copy locally.</td>
</tr>
<tr>
<td>1 0 1 X 0</td>
<td>Read from either one, do not save a copy locally.</td>
</tr>
<tr>
<td>1 1 X X 0</td>
<td>Read locally only.</td>
</tr>
<tr>
<td>0 0 0 0 0</td>
<td>Write a copy both locally and in the 360, overlay if necessary.</td>
</tr>
<tr>
<td>0 0 0 1 0</td>
<td>Write a copy in both; do not overlay an existing file.</td>
</tr>
<tr>
<td>0 0 1 0 0</td>
<td>Write in 360 only; overlay allowed.</td>
</tr>
<tr>
<td>0 0 1 1 0</td>
<td>Write in 360 only; no overlay allowed.</td>
</tr>
<tr>
<td>0 1 0 0 0</td>
<td>Write locally only, overlay allowed.</td>
</tr>
<tr>
<td>0 1 0 1 0</td>
<td>Write locally only, no overlay allowed.</td>
</tr>
<tr>
<td>0 1 1 0 0</td>
<td>No operation.</td>
</tr>
<tr>
<td>0 1 1 1 0</td>
<td>No operation—sets a return code of 3 if file exists locally.</td>
</tr>
</tbody>
</table>

Figure 2. IR Function Table
The calling program sets word 11 during writing to indicate the length of the file. This may be between 2 and 77418 words. For reading, the program sets the amount of core available to receive the file. If the file does not fit, IR indicates that with an appropriate return code. In any case, this word will always be set to the actual length of the file.

Return codes are:

0: operation successful.
1: file to be read does not fit in core.
2: file to be read not found.
3: attempt to illegally overlay a file.
4: file to be deleted not found in 360.
5: file to be deleted not found locally.
9: file to be deleted not found anywhere.
3. LIBRARY FUNCTIONAL DESCRIPTION

This chapter describes the inner workings of the various routines that comprise the library system. Appendix A contains the names and calling sequences of internal routines in the entire system.

A. Directory

The library directory is stored on the disk as a contiguous block of 8192 words. Each directory entry is 8 words long and contains the following information:

- WORD 1: TYPE
- WORD 2: USER ID
- WORDS 3-5: 6 CHAR FILE NAME
- WORD 6: FIRST BLOCK NUMBER OF FILE
- WORD 7: FILE LENGTH
- WORD 8: PROTECTION

The TYPE word contains the user file type in bits 6-11. Bits 2-5 are currently unused. Bits 0 and 1 decode as follows:

- BIT 0: 0 - entry is not currently in use
          1 - entry is in use
- BIT 1: 0 - this entry has never been used
          1 - entry has had a name in it

The USER ID is obtained from GLASP and uniquely identifies the user that created the file. The FILE LENGTH is coded as:

- BITS 0-4: number of full blocks
- BITS 5-11: length of last block
The PROTECTION word is currently unused. At a future date it will be used to allow users limited access to each other's files.

A file name is placed in the directory as follows:

1. A hash-coding is performed on the 6 character name and USERID, mapping it into the range \((0,177)\).
2. A 64 word (8 entry) block of the directory whose number is obtained from (1) is read into core (the access routines check to see if that block is already in core to avoid re-reading).
3. A sequential search is started at the beginning of the block for a free entry (BIT 0 of the TYPE equal to 0). The next sequential blocks are read into core if necessary.
4. An entry is made, and the block is rewritten onto the disk.

An entry is found in a fashion similar to insertion, except that searching stops when:

1. The correct name is located, or
2. An entry that has never been used (i.e., BIT 1 of TYPE equals 0) is found.

The reason for criterion (2) is best illustrated by an example. Assume that names A, B, and C all map into block N that has never been used, and are inserted in the above order. File B is then deleted. The search for file C should not give up at the first free entry (i.e., that formerly occupied by B), but should continue until an entry is reached that has never been used. This guarantees that searches never terminate prematurely.
The directory has room for 1024 entries. Current estimates are that not more than half of the directory will ever be in use which implies that the hashing technique will rarely have to access more than one block.

B. **File Storage and Allocation**

The bulk of the disk is dedicated to actual file storage. This area of the disk is divided into blocks of 128 words each. A file is a linked string of blocks, and has the following structure:

![Figure 3. File Structure](image)

All blocks except the last contain 127 words of data, followed by a one word link pointing to the next block. The last block in the file, which may also be the only block, contains anywhere from 2 to 128 words without a link. The reason that a last block never contains one word is that the previous block may be made into the last block of length 128 words, thereby including the one word.

A file's position on the disk is fully described by:

1. The first block number of the file.
2. The number of full blocks (i.e., blocks with links).
3. The number of words in the last block.

The disk contains 6144 words per physical track, or exactly 48 blocks of 128 words. Since the disk is fixed head, blocks M and N are equivalent with respect to head position if \( M = N \mod 48 \). One block
takes approximately 820 μsec. to pass under the heads. It can be seen that if two successive blocks in a file, M and N, are related such that
\begin{equation}
N = M + 2, \mod 48
\end{equation}
then the software has approximately 820 μsec. from the time block M is transferred before block N is in a position to be read or written. Software response time for initiating a transfer is considerably less than this, so if all blocks in a file satisfy the above relationship, a file can be read or written at the rate of 24 blocks every revolution. The total transfer time $T$ for a file of $L$ blocks, including an average half-revolution latency of 16.7 ms. is
\begin{equation}
T = 16.7 + 1.62L \text{ ms.}
\end{equation}

The block allocation scheme must therefore allocate block numbers that satisfy (1) in order to achieve this reduced transfer time. This is achieved by maintaining two lists of unused blocks on the disk. One list contains the block numbers in sorted order (i.e., successive blocks differ by 2, \mod 48 where possible), while the other list contains unsorted block numbers. Blocks are allocated from the first list, and deallocated into the second list. When the list of sorted blocks is exhausted, the unsorted list is sorted out, emptied, and transferred back into the first list.

To minimize accessing the disk for free block numbers, a copy of the top 64 words of the sorted list is kept in memory. Blocks are allocated from here until the list is emptied, at which point it is refilled from the disk. The disk layout is shown in Figure 4.
Sorting of free blocks takes place infrequently as there are 1359 blocks available (block 0 is not used). Sorting is done by the following algorithm:

1. An IR program segment for sorting is loaded into 340008 by DSKGET.
2. Banks 0 and 1 are saved by BNKSWP.
3. The unsorted list is read into bank 0.
4. The block numbers are sorted into one of 48 slots in bank 1 where block N is placed into slot N, mod 48.
5. A scan is made of the slots starting at slot 0. A block number is removed and placed back into a list in bank 0. The scan examines every other slot and keeps removing blocks. If a slot is empty, the next sequential slot is examined. Note that slot 0 follows slot 47.
6. Scanning stops when all slots are empty.
7. The free block lists are restored to the disk.
8. Banks 0 and 1 and the program segment are restored.

This procedure takes about 0.5 seconds, and guarantees that where possible, successive blocks differ by S, mod 48 where S is > 2 and minimal.

Flowcharts for the main IR functions are shown in Figures 5-9.
IRREQ

INITIALIZE, GET PARAMETER, RETURN CODE = 0

FUNC BIT 4 =

DELETE

0

WRITE

1

READ

IRDONE

RETURN LENGTH AND RETURN CODE

RETURN VIA ODPONE

Figure 5. IRREQ Flow Diagram
Figure 6. Read and Save Flow Diagrams
DELETE

FUNC
BIT = 1

REQUEST
DELETION
IN 360

RETURN
CODE = 4

FILE
IN 360?

SEARCH
LOCAL
DIRECTORY

INCREMENT
RETURN
CODE BY 5

IRDONE

DELETE
FROM LOCAL
DISK

Figure 7. Delete Flow Diagram
Figure 8. Write Flow Diagram (Part 1)
Figure 9. Write Flow Diagram (Part 2)
LIST OF REFERENCES


APPENDIX A

IR INTERNAL ROUTINES

This appendix contains excerpts from the IR system that define the names and calling sequences of most of the internal routines. It should prove helpful if it is ever necessary to expand the system.
GODISK

Performs a disk transfer using locations defined in page 0 of bank 3.

BNKSWP

TRANSFERS WORDS 4-17777

CALLING SEQUENCE:

TAD (BLK+DIR /O=WRITE, 4000=READ)
JMS BNKSWP
(RETURN)

BLOCK NUMBERS ARE: 0, 2, 4, 6, 10, 12, 14, 16

USES DISK WORDS DKcl to DKcl+17777

DSKGET

READS AND WRITES PROGRAM SEGMENTS AT 34000-35777

CALLING SEQUENCE:

TAD (BLK+DIR /BLK<40(8)
JMS DSKGET
(RETURN)

USES DISK WORDS DKC2 to DKC2+77777

BLKIO

READS AND WRITES STANDARD LIBRARY BLOCKS

FORMAT:

WORDS 1-177: DATA
WORD 200: LINK TO NEXT BLOCK

THE LAST BLOCK OF A FILE CONTAINS FROM 1 TO 200 DATA WORDS, AND NO LINK WORD
CALLING SEQUENCE:

JMS BLKIO
FUNC
BLOCK /BLOCK NUMBER
MEMADD /LOCATION OF DATA
LINK/LENGTH /CONTAINS LENGTH IF LAST BLOCK
(RETURN)

THE LINK/LENGTH WORD IS 0 FOR 200 WORDS IN
THE LAST BLOCK

THE LINK/LENGTH WORD IS SET TO 0 WHEN THE
LAST BLOCK IS TRANSFERRED

FUNC:
BIT 0: 1 FOR READ, 0 FOR WRITE
BIT 1: 1 IF THIS IS LAST BLOCK
BITS 6-8: FIELD FOR DATA

READF
READ A LINKED FILE FROM THE DISK

CALLING SEQUENCE:

JMS READF
BLOCK /FIRST BLOCK NUMBER
NBLOCKS /NUMBER OF FULL BLOCKS
LENGTH /LENGTH OF LAST
MEMADD /LOC OF DATA
BANK /FIELD IN BITS 6-8

WRITF
WRITE A LINKED FILE

CALLING SEQUENCE:

JMS WRITF
BLKLST
NBLOCKS /NUMBER OF FULL BLOCKS
LENGTH /LENGTH OF LAST
MEMADD /LOC OF DATA
BANK /FIELD FOR DATA
(RETURN)

BLKLST POINTS TO A BANK 3 LIST
OF BLOCKS TO BE USED
GETBLK

GETS A POINTER TO A LIST OF SORTED BLOCKS

CALLING SEQUENCE:

\[
\begin{align*}
\text{TAD } N & \quad \text{/NUMBER OF BLOCKS NEEDED} \\
\text{JMS GETBLK} \quad & \quad \text{(RETURN) } \quad \text{/AC CONTAINS POINTER TO LIST}
\end{align*}
\]

FREBLK

FREE A LIST OF BLOCKS BY WRITING THEIR NUMBERS ONTO THE DISK

CALLING SEQUENCE:

\[
\begin{align*}
\text{TAD } N & \quad \text{/NUMBER OF BLOCKS} \\
\text{JMS FREBLK} \quad & \quad \text{(RETURN) } \\
\text{BBUF} & \quad \text{/LOCATION OF LIST}
\end{align*}
\]

DIRIO

READ AND WRITE DIRECTORY BLOCKS

CALLING SEQUENCE:

\[
\begin{align*}
\text{TAD } (N+\text{DIR}) & \quad \text{/N BETWEEN 0 and 177} \\
\text{JMS DIRIO} \quad & \quad \text{(RETURN) } \\
\text{DIR=4000 FOR READ} & \\
\text{BLOCK IS IN DIRBUF}
\end{align*}
\]

DIRNXT

GETS POINTER TO NEXT DIRECTORY ENTRY

CALLING SEQUENCE:

\[
\begin{align*}
\text{TAD } N & \\
\text{JMS DIRNXT} \quad & \quad \text{(RETURN) } \\
\text{AC HAS POINTER}
\end{align*}
\]

IF \( N \) IS BETWEEN 0 and 177, THAT BLOCK IS READ, AND THE POINTER IS AT ITS BEGINNING.

IF \( N \) IS NEG., THE NEXT SEQUENTIAL ENTRY IS OBTAINED
DIRSRC

SEARCHES FOR NAME IN PARAMETER LIST

CALLING SEQUENCE:

JMS DIRSRC
(RETURN) /AC CONTAINS POINTER IF FOUND, /ELSE 0

DIREMP

FIND AN EMPTY SLOT

CALLING SEQUENCE:

JMS DIREMP
(RETURN) /AC HAS POINTER

SAVE

SAVES A FILE ON THE DISK IF FUNC BIT 2=0

IF DIRPNT=0, GETS NAME, ETC. FROM PARAMETER AREA, IF DIRPNT IS NON-0, THEN USES THAT DIRECTORY ENTRY.

GETLEN

COMPUTE DISK PARAMETERS FROM CORE LENGTH

CALLING SEQUENCE:

TAD LENGTH /NUMBER OF WORDS
JMS GETLEN
LENGTH /LENGTH OF LAST HERE
NBLOCKS /# OF FULL BLOCKS HERE
(RETURN)

COMLEN

COMPUTE CORE LENGTH OF A FILE

CALLING SEQUENCE:

TAD LEN /PACKED DIRECTORY FORMAT
JMS COMLEN
(RETURN) /LENGTH IN AC
APPENDIX B

DISK STORAGE ALLOCATION
The IR system is implemented using the Data Disk that is currently attached to the PDP-8. A new interface is currently being developed for the disk, and will have the following characteristics:

1. A total capacity of 294,912 16-bit words (12 bits data, 1 bit parity, 1 mark bit, 2 timing bits).
2. Bi-directional capability of from 1 to 8192 words per transfer.
3. The ability to begin and end a transfer to any point on the disk.
4. Transfer speed of 5.4 usec. per word.
5. Rotational speed of 30 revolutions per second.

The IR system logically divides the disk into six areas as follows (see Figure B-1):

1. IPL tracks - 12,288 words reserved for the system bootstrap, restart programs, etc. This area is not used by IR since it is write-protected.
2. Core swapping - 65,536 words are set aside for the purpose of saving core images containing data structures and display files associated with each of eight consoles. There is enough room for sixteen banks of core, allowing each console to save two banks of core. GLASP accesses this area via BNKSWP which is provided by IR.
3. Program segments - GLASP may retrieve any of 32 "programs," each of length 1024 words, from this area by calling DSKGET. The total storage allocated here is 32,768 words.
4. Free block lists - 2048 words are set aside to store the sorted and unsorted lists of free blocks.

5. Directory - this area of 8192 words is used to store the library directory. Since each entry uses 8 words, there is room for 1024 entries.

6. Library storage - the remainder of the disk is divided into 1360 128-word blocks for a total of 174,080 words. These blocks are used for actual data and picture storage.

Figure B-1. Disk Storage Map
APPENDIX C

DISK MONITOR SYSTEM
Until recently, all program development for the PDP-8 was done either on the Dectape operating system, or on a PDP-8 assembler that ran in the 360. The former suffered from excessively long periods needed for file-editing, assembly, and listing because of the slow tape units, while the latter system was considered close to useless due to excessive turn-around time. The acquisition of a disk and a high speed printer (Teletype Corporation's Inktronic printer—120 characters per second) for the PDP-8 advised development of a local disk-based monitor system. Rather than start from scratch, it was decided to adapt DEC's Disk Monitor System to run in the Digital Computer Laboratory environment. This involved modifying DEC's programs, and writing some new ones to match the current hardware configuration.

Before reading the following sections, the reader is strongly advised to familiarize himself with DEC's manual "Disk Monitor System," DEC-08-SDAB-D which describes most of the DMS. The material contained below is intended to be a user's guide to the modified system.

**Generating a User SWAPTAPE**

Each user must have a SWAPTAPE (physically a DECTAPE) on which to store permanent copies of his files and programs. A SWAPTAPE is generated by:

1. Bringing up any SWAPTAPE on unit #8.
2. Mounting an unused (but formatted) tape on unit #4.
3. Calling program SYSGEN.*

* Throughout this appendix, programs that run under the DECTAPE system are referred to as "PROG," while DMS programs are referred to as ".PROG." The _ is typed by the DMS, not by the user.
The tape on unit #4 is now a complete copy of the one on unit #8.

The format of a SWAPTAPE is shown in Figure C-1.

<table>
<thead>
<tr>
<th>Standard Dectape</th>
<th>Monitor Head</th>
<th>Image of disk blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0-2177 in that order</td>
</tr>
</tbody>
</table>

Figure C-1. SWAPTAPE Format (Block Numbers)

Loading the DMS

The DMS is transferred from SWAPTAPE to disk, and given control by:

1. Bringing up the SWAPTAPE on unit #8.
2. Calling SWAP.

The DMS responds with a L. The user may now call up any DMS programs by typing the appropriate name, as is described in DEC's publication.

Modifications to DEC's DMS

1. The DMS monitor head was replaced with one appropriate to the Data Disk on the machine. All characteristics are the same except that:
   a. The monitor head uses 7400-7777 in bank 3 in addition to 7600-7777 in bank 0.
   b. SYSIO may be called from any field (the DF points to the caller).
2. .CD. is modified so that Dn:FILE refers to a file on a SWAPTAPE on unit #n. CDIO is modified similarly.
3. .PIF - all functions work with Dn: taken as above, except that copy functions may NOT involve a SWAPTAPE.

4. .PALD - extra symbols are kept in bank 3 instead of .SYM, and OPT-R results in paginated output on the Inktronic.

New DMS Programs

The following are new programs developed for the DMS, along with a description and operating instructions:

1. .LIST - lists a single ASCII file in paginated form. OPT-T sends listings to the ASR-33, anything else uses the Inktronic.

2. .INDX - similar to L option in .PIP except that it is automatic and uses the Inktronic.

3. .COPY - used to copy ASCII files between Disk, SWAPTAPE, and DECTAPE. Disk files are designated as S:FILE, SWAPTAPE files as Dn:FILE. If a carriage return is typed in response to *OUT: or *IN: or both, the DECTAPE I/O routines are brought in from a DECTAPE on unit #8 and will type OUTPUT: or INPUT: or both for the missing files. Disk or SWAPTAPE files are terminated by a double form-feed (automatically placed by the DMS), while DECTAPE files must contain $ in column 1 to properly terminate.

4. .VGET - loads programs from DECTAPE into core, and returns to the DMS. Light-pen the program desired.

5. .RCVR - returns control to DECTAPE system.

6. .VRFY - verifies that no errors exist in the user's files by checking all links.
Saving the Disk on a SWAPTAPE

To save the contents of the user's disk on his SWAPTAPE:

1. Mount the tape on unit #4.
2. Call .EXIT.

Control is returned to the DMS.

IPL

The first two tracks of the disk have a special write-protect feature and are used for Initial Program Loading. The DMS Monitor Head is normally resident there, but it can be restored by:

1. Turning off the IPL Write-Lock.
2. Running WIPL.
3. Turning the Write-Lock back on.

The Monitor Head may be read into core by running program IPL or by:

1. Loading 6557 into location 7776.
2. Loading 5377 into location 7777.
3. Starting the computer at location 7776.
1. AEC REPORT NO.  
   000-1469-0174

2. TITLE  
   LOCAL INFORMATION RETRIEVAL FOR THE SIMULATION AND MODELING SYSTEM

3. TYPE OF DOCUMENT  
   (Check one):
   a. Scientific and technical report
   b. Conference paper not to be published in a journal:
      Title of conference  
      Date of conference  
      Exact location of conference  
      Sponsoring organization  
   c. Other (Specify)  

4. RECOMMENDED ANNOUNCEMENT AND DISTRIBUTION  
   (Check one):
   a. AEC's normal announcement and distribution procedures may be followed.
   b. Make available only within AEC and to AEC contractors and other U.S. Government agencies and their contractors.
   c. Make no announcement or distribution.

5. REASON FOR RECOMMENDED RESTRICTIONS:

6. SUBMITTED BY: NAME AND POSITION (Please print or type)
   C. W. Gear, Professor and Principle Investigator
   Department of Computer Science
   University of Illinois
   Urbana, Illinois 61801

7. AEC CONTRACT ADMINISTRATOR'S COMMENTS, IF ANY, ON ABOVE ANNOUNCEMENT AND DISTRIBUTION RECOMMENDATION:

8. PATENT CLEARANCE:
   a. AEC patent clearance has been granted by responsible AEC patent group.
   b. Report has been sent to responsible AEC patent group for clearance.
   c. Patent clearance not required.

FOR AEC USE ONLY

Date  
August 1970