DOMAIN System
Utilities

Order No. 009414
Revision 00

Apollo Computer Inc.
330 Billerica Road
Chelmsford, MA 01824

Printed in U.S.A.

First Printing: September, 1986

This document was produced using the SCRIBE document preparation system. (SCRIBE is a registered trademark of Unilogic, Ltd.)

Preface

The *DOMAIN System Utilities* is the fourth volume in the four-volume introduction to the DOMAIN Computing System. The first volume, *Getting Started With Your DOMAIN System*, provides a tutorial approach to getting started on your node. The second volume, *DOMAIN System User's Guide*, constitutes a handbook that takes you beyond the introductory stage into practical applications of Display Manager (DM) and Shell operations. The third volume, *DOMAIN System Command Reference* provides reference information on all of the DM and Shell commands that are available to you.

This fourth document details specific commands which are considered utilities. They previously were Appendices to the *DOMAIN System Command Reference* manual. We assume that you are familiar with the material in the first three books before you attempt to use this reference manual. Fundamental concepts like file structure and usage are taken for granted here. We tell you *how* to use the utilities; not *why* you might want to use them.

Organization of this Manual

This manual contains nine chapters:

Chapter 1	Describes the utility CALENDAR which sets the date and time in the hardware calendar, and stores time zone information on a logical volume.
Chapter 2	Describes the utility CHUVOL, a tool for system builders and service representatives, which changes all UIDs on a disk.
Chapter 3	Describes the utility DCALC which provides the features of a desk calculator, evaluating both logical and arithmetic expressions.
Chapter 4	Describes the utility ED which invokes the line editor.
Chapter 5	Describes the utility EDFONT which is a menu-driven program to design your own letters, numbers and/or special characters.
Chapter 6	Describes the utility FMT which is a general purpose text formating program.
Chapter 7	Describes the utility INVOL which initializes physical disk volumes, creates logical volumes and maintains badspot lists.
Chapter 8	Describes the utility ITEST which test the type managers that manage input and output to objects.
Chapter 9	Describes the utility SALVOL which verifies, and if necessary, corrects the tables that describe the allocation of disk blocks to the files stored on the disk.

Problems, Questions, and Suggestions

We appreciate comments from the people who use our system. In order to make it easy for you to communicate with us, we provide the User Change Request (UCR) system for software-related comments, and the Reader's Response form for documentation comments. By using these formal channels you make it easy for us to respond to your comments.

You can get more information about how to submit a UCR by consulting the description of the Shell command CRUCR (CREATE_USER_CHANGE_REQUEST). You can also get more information by typing:

```
$ HELP CRUCR <RETURN>
```

For your comments on documentation, a Reader's Response form is located at the back of this Guide.

Documentation Conventions

Unless otherwise noted in the text, this manual uses the following symbolic conventions.

UPPERCASE	Uppercase words or characters in formats and command descriptions represent commands or keywords that you must use literally.
lowercase	Lowercase words or characters in formats and command descriptions represent values that you must supply.
[ ]	Square brackets enclose optional items in formats and command descriptions. In sample Pascal statements, square brackets assume their Pascal meanings.
{ }	Braces enclose a list from which you must choose an item in formats and command descriptions. In sample Pascal statements, braces assume their Pascal meanings.
|	A vertical bar separates items in a list of choices.
< >	Angle brackets enclose the name of a key on the keyboard.
CTRL/Z	The notation CTRL/ followed by the name of a key indicates a control character sequence. You should hold down the <CTRL> key while typing the character.
...	Horizontal ellipsis points indicate that the preceding item may be repeated one or more times.
	Vertical ellipsis points mean that irrelevant parts of a figure or example have been omitted.

Contents

Illustrations

Tables

Chapter 1
CALENDAR

1.1. Introduction

The calendar utility, CALENDAR, sets the date and time in the hardware calendar, and stores time zone information on a logical volume. You must use this utility at least once on any volume from which the operating system will be started. Its use is unnecessary on other volumes. CALENDAR must be run only on initialized disks.

You may run CALENDAR either from the Shell or from the Mnemonic Debugger. From the Shell, type

```
$ calendar
```

and follow the prompts as described below.

1.2. Running CALENDAR

To run the calendar utility from the Mnemonic Debugger, first shut down the Display Manger by typing SHUT in the DM input window. When you receive the Debugger prompt "> ", enter service mode by flipping the NORMAL/SERVICE switch on your node to the SERVICE position, then type the following command:

```
>EX CALENDAR <RETURN>
```

This command loads and starts the calendar as a stand-alone utility. The utility identifies itself, then requests the type of disk and the logical volume number. Type W for a Winchester disk, S for a storage module disk, or F for a floppy disk. To specify a unit number, append 0 or 1 to the letter. For example, S1 denotes storage module unit 1. Unit 0 is the default. Next, enter the number of the logical volume if it is not 1.

The utility now checks to see if the volume already contains any calendar information. If the calendar has already been set, the utility informs you of the settings and allows you to change them. If the calendar has never been set, the utility prompts you for the required information.

First, the utility requests a time zone. The time zone is necessary because all date/time information is recorded internally in Coordinate Universal Time (UTC). The time zone offset is the value which, when applied to a UTC time, produces a local time. You can specify either a time zone name (see below) or a positive or negative offset from UTC. If you specify an offset, the utility later prompts you for an optional 1 to 4 character identifier to be used as a time zone name. Offsets may be in whole or half-hour increments. Other fractional offsets produce an error message.

After you enter the time zone, the utility prints the date and time stored in the hardware calendar and asks if you want to make changes. If so, specify the date in the following format:

```
[yy]yy/mm/dd
```

Leading zeros are not required, and you can omit the century. For example, 84/6/7 specifies June 7, 1984.

Next, the utility requests the local time. Enter the local time in hh:mm format.

A warning message appears if you attempt to set the time backward, or attempt to set it forward by more than five minutes. If you are sure that what you typed is correct, ignore the message. The utility then sets the date and time in the hardware calendar and exits.

When you have completed the CALENDAR routine, place the NORMAL/SERVICE switch back in its NORMAL setting.

Table 1-1. Valid Time Zones

Name	Time Zone
EDT	Eastern Daylight Time
EST	Eastern Standard Time
CDT	Central Daylight Time
CST	Central Standard Time
MDT	Mountain Daylight Time
MST	Mountain Standard Time
PDT	Pacific Daylight Time
PST	Pacific Standard Time
GMT	Greenwich Mean Time
UTC	Coordinate Universal Time

Chapter 2
CHUVOL (Change UID of Volume)

2.1. Introduction

Every node has an identifier recorded in read-only memory that is internal to the node. If the node has a Winchester disk, this node ID is also stored on the disk as part of the UID (unique identifier) of each object on the disk. For proper system performance, the IDs stored on the disk must match the ID of the node to which the disk is physically attached.

The CHUVOL (Change UID of Volume) utility changes all UIDs on a disk so that the node ID component of each UID matches the ID of the node to which it is physically attached.

NOTE: This procedure is for use by system builders, their designated service representatives, and our service representatives. *It is not intended for end users.*

Always run CHUVOL after replacing the Winchester disk in the mass storage module. If you attempt to boot the operating system without first running CHUVOL, you will receive a warning message. The system checks the ID of the root directory of the boot volume against the node ID, and prints the following warning message when the two do not match:

```
The node number of this node differs
from that stored on the boot volume.
Prom node #nnnnnn, stored node #nnnnnn.
Do you want to proceed?
```

If you respond N(o) to this question, the operating system will shut down and exit to the Mnemonic Debugger. Then, you can run CHUVOL. (It is possible to run CHUVOL on-line; the program resides in the /COM directory (/COM/CHUVOL. To run it, you load the operating system from another node, as if the node were diskless.) We recommend, however, that you run CHUVOL off-line, using the procedure that follows.

NOTE: CHUVOL is intended for use on new disks received from the manufacturer. It is *not* intended as a general-purpose means of moving a disk from one node to another. In particular, if you run CHUVOL on a disk that contains unrecorded badspots or an inconsistent file system, it may become necessary to completely reinitialize and reload the disk.

To change IDs, CHUVOL establishes a source ID and a target ID. The source ID is the node ID component of the UID of the boot volume's root directory. This ID is generated when you run INVOL. The target ID is the node ID, which is stored in the node's read-only memory. CHUVOL changes any ID that matches the source ID to the target ID. The following procedure describes how to run CHUVOL.

2.2. Running CHUVOL

1. If the node is not already under the control of the Mnemonic Debugger, shut down the operating system (use the instructions in Chapter 4, Section SHUTDOWN.SEC, if necessary).

2. Ensure that the Winchester disk is operational. Do not run CHUVOL if you suspect there is a problem with the Winchester disk.

3. Type the following to reset the Mnemonic Debugger and execute CHUVOL:

   RE <RETURN>
   <RETURN>
   EX CHUVOL<RETURN>

   The program header and a warning then appear:

   Change_UID_of_Volume - Version xx, yy/mm/dd

   CAUTION: This program changes the unique indentifier of every file on the volume. It should be run only once after the disk is installed, and after the disk diagnostic has successfully completed. If the node crashes during CHUVOL, re-execute CHUVOL; do not salvage the volume at this point, or files may be lost. If you wish to abort, type Q to the following prompt:

   Controller type (W=winchester,S=storage module, F=floppy)?

   If you wish to continue, enter **W <RETURN>** to show that you are running CHUVOL. Otherwise, follow the program's instructions and abort. If you do so, the program prints the message RUN ABORTED.

4. For each logical volume on the disk, CHUVOL checks to make sure the volume does not need salvaging. If it does, then CHUVOL prints the following:

   LOGICAL VOLUME n REQUIRES SALVAGING
   PLEASE RUN SALVOL AND THEN
   RE-EXECUTE THIS PROGRAM

   Then, it returns to the Mnemonic Debugger. At this point, you should run the SALVOL utility (for more information see the chapter on SALVOL) and execute CHUVOL again.

5. The program then displays the node ID stored on the logical volume and the node ID stored in read-only memory:

   Logical volume 1 was built with node ID nnnnnn,
   to be changed to nnnnnn.
   Is this correct?

   If you wish to abort, reply **N <RETURN>** to this question; the program displays the message RUN ABORTED. Note that this is your *last* chance to abort. If you wish to continue, reply **Y <RETURN>**.

6. The program then starts to update the logical volume, and it reports the percentage of the volume that it has changed, as follows:

Logical volume 1, % complete:

20
40
60
80
100

Logical volume 1 update complete.

If there is more than one logical volume on the disk, CHUVOL proceeds to the next logical volume. If the ID of the next logical volume is the same as the original ID of the logical volume just processed, then CHUVOL uses the same source and target IDs. If the ID of the next logical volume differs from that of the previous volume, the program prompts you, as shown in Step 5.

7. Once it has processed all logical volumes on the disk, CHUVOL prints: **Physical volume update complete**, and returns to the Mnemonic Debugger.

Chapter 3
DCALC (DESK_CALCULATOR)

3.1. Introduction

DCALC mimics the features of a desk calculator, evaluating both logical and arithmetic expressions.

ARGUMENTS

pathname
(optional)

Specify input file containing expressions to be evaluated, one expression per line.

Default if omitted: read standard input; stop with CTRL/Z

OPTIONS

If no options are specified, all operations are decimal-based.

-H

Specify hexadecimal operations.

3.2. Expressions

Input expressions can be simple arithmetic expressions or variable assignment expressions. DCALC writes the value of each evaluated expression on standard output. Variables hold temporary values, which DCALC does not automatically write.

Expressions may include any of the operators listed below (in order of precedence):

1.	+ -	unary plus and negation operators. These may only appear at the start of an expression or within parentheses.
2.	<< >>	logical left and right shift
3.	**	exponentiation
4.	* / %	multiply, divide, modulo (remainder)
5.	+ -	add, subtract
6.	==	equal to
	!=	not equal to
	>	greater than
	>=	greater than or equal to
	<	less than
	<=	less than or equal to

```
7.    !          unary logical not

8.    |          logical or
      &          logical and
      ^          logical xor
```

Relational operators return the value 1 for true and 0 for false. DCALC performs operations in double precision floating point, except for logical operators listed as items 2 and 8 above, which use 32-bit integers.

Variables

Expressions may include previously declared variables. Use this format to declare a variable:

```
name = expression
```

- A variable name must begin with a letter and may consist of any combination of letters and digits.
- DCALC does not automatically print replacement expressions, because they usually contain temporary values.

Radix Control

You can change the default base for input or output using ibase (input base) and obase (output base) statements. For example,

```
ibase = 2
obase = 16
```

causes DCALC to interpret input in binary and print results in hexadecimal.

To set an individual number's radix, precede it with the desired radix and a pound sign. For example,

```
16#100
```

specifies the hexadecimal number 100 (equals 256 in decimal).

3.3. Examples

```
     Your input:          DCALC output:

1.   10 + (-64 / 2**4)    6

2.   temp = 2#101
     temp == 5            1 (true)

3.   ibase = 16
     obase = 2
     11 + 28              111001
     1a + 0f              101001
     (Note that when you type a hexadecimal number that begins with
      a letter, you must precede it with a zero.)
```

```
      Your input:          DCALC output:

4.    ibase = 16
      numa = 100
      numb = 100
      numa + numb          512
```

Chapter 4
ED (EDIT)

4.1. Introduction

ED invokes the line editor. Input text and editing commands are read from standard input. While you may use ED to create text files interactively, it is better suited for use in programs and scripts. Use the <EDIT> key or the DM command, CE, to create and edit files interactively.

NOTE: There is a homonymous DM command: ED -- Delete character preceding cursor. See the ED command description in the DM chapter for details.

ARGUMENTS

pathname
(optional)

Specify file to be edited. ED reads the file into a buffer for editing and remembers its name for future use. ED operates on the buffer copy; changes made there have no effect on the original file until you issue a W (write) command from within ED. Files are limited to 6400 lines.

If the 'pathname' argument is omitted, the edit buffer is empty and no file name is remembered for future use. You will have to specify an explicit file name when you exit the editor.

Default if omitted: see above

OPTIONS

-N

Suppress the printing of line counts by the E (edit), R (read), and W (write) commands.

4.2. Summary of ED Commands

Commands to ED have a consistent format: zero, one, or two line addresses followed by a single-character command, with optional parameters following the command. The general format is:

```
[line,][line]command parameters
```

The [line] specifies a line number or address in the current edit buffer. There is usually a useful default for each command (normally the current line) so that you don't need to specify an address explicitly.

Addresses:

```
17      a decimal number
.       the current line
```

```
   $       the last line of the file
   /pat/   search forward for line containing pat
   \pat\   search backward for line containing pat
   line+n  n lines forward from line
   line-n  n lines backward from line

Defaults:

   (.)     use current line
   (.+1)   use the next line
   (.,.)   use current line for both line numbers
   (1,$)   use all lines

Commands:
 (.)     A               Append text after line (text follows)
 (.,.n)  Bn              Browse over the next n lines (default n is 22).
                         If n is negative, print last n lines before
                         current line.  If 'B.' is specified, print n
                         lines with current line in center of screen.
 (.,.)   C               Change text (text follows)
 (.,.)   D               Delete text
         E file          Discard current text, enter file, remember
                          filename
         F               Print filename
         F file          Remember filename
 (.)     I               Insert text before line (text follows)
 (.,.)   Kline           Copy text to new line after specified line
 (.,.)   Mline           Move text to line after specified line
 (.,.)   P               Print text (can be appended to other commands)
         Q               Quit
 (.)     R [file]        Read file, appending after current line
 (.,.)   S/pat/new/GP    Substitute new for leftmost pat (G implies all
                         occurrences)
 (1,$)   W [file]        Write file, leave current text unaltered (if
                         no file is specified, write to current filename)
 (.)     =[P]            Print line number, current line
 (.+1)   <CR>            Print next line
 (1,$)   G/pat/command   Execute command on lines containing pat
                         (except A, C, I, Q commands)
 (1,$)   X/pat/command   Execute command on lines not containing pat
                         (except A, C, I, Q commands)
         # ...           Comment
         $n              Read or write temporary buffer, "n".
```

The error message "?" is printed whenever a command fails or is not understood.

4.3. Limitations

- Files being edited can contain up to 6400 lines.
- When a global search and substitute combination fails, the entire global search stops.
- Problems sometimes occur when removing or inserting NEWLINE characters (via @n), especially in global commands.

4.4. ED Commands In Detail

ED accepts commands interactively (from the keyboard) and in a batch-like manner (from script files). To use a script file, substitute the script file name for standard input:

```
ED [options] [pathname] <script
```

Command Format

Commands to ED have a consistent format: zero, one, or two line addresses followed by a single-character command, with optional parameters following the command. The general format is:

```
[line,][line]command parameters
```

The [line] specifies a line number or address in the current edit buffer. There is usually a useful default for each command (normally the current line) so that you don't need to specify an address explicitly.

Line addresses are formed from the following components:

```
17          an integer number
.           the current line
$           the last line in the buffer
.+n         n lines past the current line
.-n         n lines before the current line
/pattern/   a forward context search
\pattern\   a backward context search
```

Line numbers can be separated by commas or semicolons; a semicolon sets the current line to the previous address before the next address is interpreted. This feature can be used to determine the starting line for forward and backward context searches (// and \\).

Regular Expressions

ED supports regular expression notation for specifying patterns in line addresses and in the S, G, and X commands. A regular expression represents one or more strings of characters for which to search. For a description of regular expressions, refer to the chapter on DM basics. The notation is summarized below for your convenience. These search and substitute operations are identical in function to their Display Manager counterparts, although the syntax of the DM's S command differs slightly. Summary of Regular Expression Notation

```
c         Literal character
?         Any character (except newline)
%         Beginning of line
$         End of line
[...]     Character class (any one of these characters)
[~...]    Negated character class (all characters except those in brackets)
[c1-c2]   Any single character in the range c1 to c2
@c        Escaped character (e.g.,@%, @[, @*)
@n        Newline
@t        Tab character
*         Closure  (zero or more occurrences of previous pattern)
{...}     Tagged pattern
```

4.5. ED Commands

The following is a list of ED commands. Default line addresses are in parentheses. Commands may be typed in either upper- or lowercase.

(.)A
[text]
.

The append command reads the text and appends it after the addressed line. The current line is left on the last line input, if any. If no lines are input, the current line remains on the addressed line. Signify the end of the text by typing a line with a period as its first and only character.

(.)B[+/./-][screensize]

The browse command is a shorthand command to print out a screen of data. It has three basic forms, any of which may be followed by a screensize. A simple B (or B+) prints the current line and the screen after it. B- prints the screen of text preceding (and including) the addressed line. B. prints a screen of text, centered on the addressed line. Except for the B. command, these commands leave the current line at the last line printed. The default screensize is 23 lines. If you specify a screensize, it becomes the default screensize for the rest of the editing session or until changed.

(.,.)C
[text]
.

The change command deletes the addressed lines, then accepts input text which replaces these lines. The current line is left at the last line input, if there were any, otherwise at the first line not deleted. Signify the end of the text by typing a line with a period as its first and only character.

(.,.)D The delete command deletes the addressed lines from the buffer. The line originally *after* the last line deleted becomes the current line; however, if the lines deleted were at the end of the file, the new last line becomes the current line.

E [filename]

The edit command deletes the entire contents of the buffer and then reads in the named file. When it executes this command, ED sets the current line to the last line of the buffer and displays the number of lines read. Also, it remembers the supplied filename for possible use as a default filename in subsequent R or W commands.

F [filename]

If you specify a filename, the currently remembered filename is changed to that name. Otherwise, ED prints the currently remembered filename.

(1,$)G/regular expression/command

The global command executes the other specified command for every line that matches the regular expression. To execute multiple commands on the lines matched, place each on a separate line and terminate each command except the last with an "at" sign (@). For example,

```
g/foo/s/bar/zot/@          | For all lines containing the string "foo",
s/wazoo/munch/             |  replace "bar" with "zot" and "wazoo" with
                           |  "munch".
```

(.)I
<text>
.

The insert command inserts <text> *before* the addressed line. The current line becomes the last line input, or, if there are no new lines, the addressed line. This command differs from the A command only in the placement of text. Signify the end of the text by typing a line with a period as its first and only character.

(.,.)K<address>

The kopy command copies the addressed lines to the position after the line specified by <address>. The last of the copied lines becomes the current line.

(.,.)M<address>

The move command deletes the addressed lines from their original location, and places them after the line specified by <address>. The last of the moved lines becomes the current line.

(.,.)P The print command prints the addressed lines. The last line printed becomes the current line. The P command can be used as a modifier following any other command, except the A, C, I, or Q commands. When used in this way, it prints the last line affected by the command.

Q The quit command causes ED to exit. If you have not written the file since changing it, ED reminds you once to do so.

(.)R [filename]

The read command reads the named file into the buffer after the addressed line. If you do not specify a filename, ED uses the remembered filename (see E and F commands). The remembered filename is not changed. The address 0 (zero) causes ED to read the file in at the beginning of the buffer. If the read is successful, the number of lines read is displayed. The last line read becomes the current line.

(.,.)S/regular expression/replacement/
(.,.)S/regular expression/replacement/G

The substitute command searches each addressed line for an occurrence of the regular expression. On each line that contains a match, ED replaces the first occurrence of the expression with the replacement string. If the global replacement indicator G follows the command, all occurrences of the regular expression are replaced. The delimiting character for the regular expression and replacement need not be a slash (/); you can use any character except a space or newline. If the substitution fails on all addressed lines, ED prints a question mark (?). The last line substituted becomes the current line. If no regular expression is specified (for example, S//pat/), ED uses the previous regular expression.

An ampersand (&) in the replacement is replaced by the string that matched the regular expression. To suppress this special meaning of &, precede it with an at sign (@). Note that except for &, all characters in the replacement string are inserted literally.

To split or merge lines, use the symbol @n to stand for the NEWLINE character at the end of a line.

(1,$)W [filename]

The write command writes the addressed lines into the file. If the specified file does not exist, it is created. This command does not change the remembered filename. If no

filename is given, the remembered filename is used (see the E and F commands). The current line is left unchanged. If the command is successful, ED displays the number of lines written.

(1,$)X/regular expression/command
The except command is the same as the global command except that commands are executed for every line that does not contain a match for the regular expression.

(.)= The equals command displays the line number of the addressed line. The current line is not changed.

comment
Text following a pound sign (#) in the first column of a line is treated as a comment and is ignored by the editor. This command allows ED scripts to contain comments. Only whole-line comments are permitted (i.e., you can't place comments at the end of other legal command lines).

(.+1)<carriage return>
An address alone on a line causes the addressed line to be displayed. A blank line alone is equivalent to '.+1' and thus is useful for stepping through text.

4.6. Diagnostics

This message is printed whenever the file exceeds 6400 lines.
file size exceeded

A message is printed if you attempt to quit without writing a file that you edited. ED requires you to retype the command as a verification.

The error message "?" is printed whenever a command fails or is not understood.

Chapter 5
EDFONT (Editing a Character Font)

5.1. Introduction

EDFONT is a menu-driven program that allows you to design your own letters, numbers and/or special characters by constructing them, one at a time, on a screen grid. This grid is displayed as part of the initial EDFONT screen image. You can designate any character(s) you have created as a new font, store this font in a file and access it later for editing or printing. EDFONT allows you to do the following:

- Design a character by using grid spaces (pixels), lines, arcs, and filled or outlined circles and boxes
- Change the space size between characters
- Change the space size between lines
- Change the grid/character size
- Rotate a character on the grid
- View an actual size replica of the character you are currently designing
- Replace the character you are currently editing with one from the same or an alternate font file

In addition to designing your own fonts, you can modify existing system fonts. Refer to the FL (FONT_LOAD) command description in the DOMAIN System Command Reference for further information.

What is a Font?

A font is data that graphically describes a set of related character images. Fonts are stored in named files on a node. A font file contains exactly one font.

All of the characters which appear on a display are read from a font file. You can select the font to be used by using the Display Manager command FL (FONT_LOAD) or by calling PAD or GPR system routines from your own program. At least two fonts are always in use by a node: the standard font, contained in /SYS/DM/FONTS/STD (plus the ".19L" or ".COLOR" suffixes for the associated display types), and the window legend font, contained in /SYS/DM/FONTS/LEGEND (plus the ".19L" or ".COLOR" suffixes for the associated display types).

Fonts are generally classified into two categories: mono-spaced and proportionally-spaced. Characters in mono-spaced fonts are all exactly the same size, making these fonts most useful whenever column alignment is important. Character sizes in proportionally-spaced fonts vary from character to character. Proportionally-spaced fonts are generally used in typesetting and other document preparation applications.

See the Glossary of Terms at the end of this chapter for defintions of related terms. The Glossary is also useful for additional background information.

The size of a font file is limited by the amount of hidden display memory available. EDFONT will display a warning when you have exceeded the size limit.

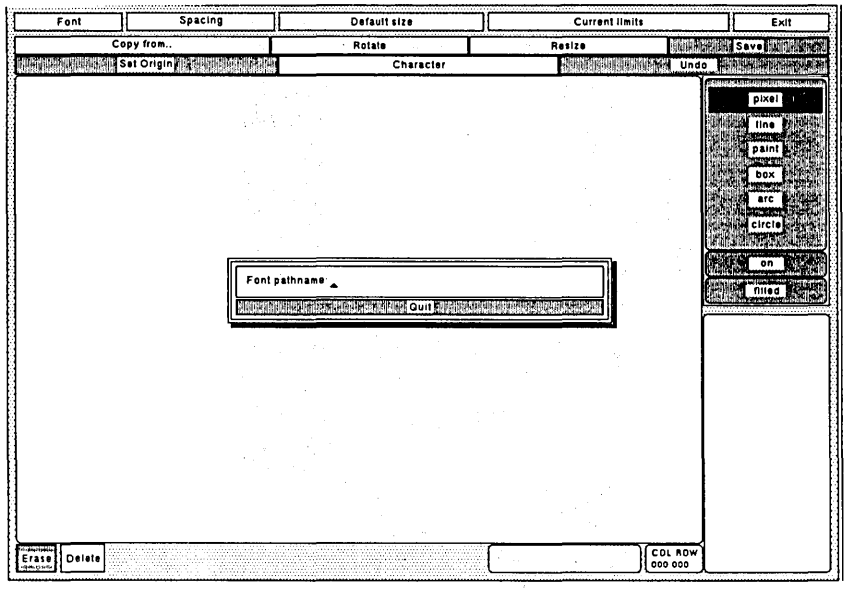
The following is a procedural description of the EDFONT program.

5.2. Invoking EDFONT

This program is designed to be used with either a mouse or a touchpad. To invoke EDFONT, type the following in the Shell input pad.

EDFONT <RETURN>

The initial screen image is displayed.



The font pathname specifies the name of a font file. It can be either a new font name or the name of an existing font. Enter the font name in the following form:

//_____/_____/_____ <RETURN>

You can also invoke EDFONT by entering the name of the font right on the Shell input line. Example: EDFONT /SYS/DM/FONTS/STD

To exit, simply position the cursor on Exit and press either the Function 1 key (F1) or the leftmost Mouse key (M1). See item 5, following, for more information on Exit.

NOTE: The F1 and M1 keys are hereafter referred to collectively as the Select key.

5.3. Sample EDFONT Display

Figure SAMPLE_DISPLAY shows a sample screen display. The following numbered items correspond to the numbered items in Figure SAMPLE_DISPLAY.

NOTE: You can get additional information about any item on the display by pressing the SHIFT and HELP keys at the cursor position where you need help. Move the cursor out of the help box to return to the original display.

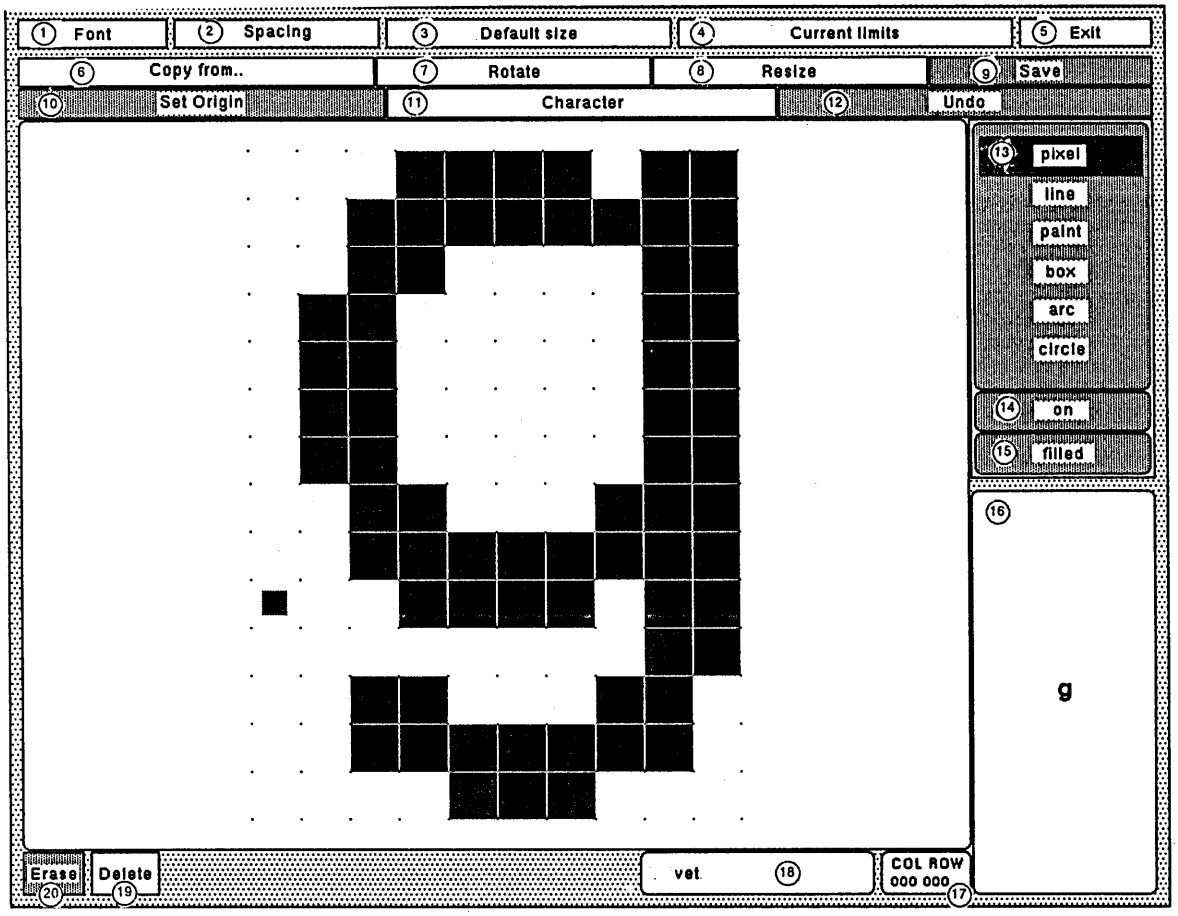


Figure 5-1. Sample EDFONT Display

1. Font

   When you position the cursor here and press the Select key, EDFONT displays the current font pathname. (For a description of how to enter the font pathname, see **INVOKING EDFONT**.)To keep the same name, simply move the cursor out of the pop-up. To change the name, use the LINE DEL key to delete, then enter the new font name and press RETURN. Position the cursor on Load This Font and press the Select key. If you have modified the font, EDFONT asks if you want to discard those modifications. Position the cursor on the appropriate box and press the Select key.

2. Spacing

   Select this option to change font values for vertical spacing, horizontal spacing, or space size. (Refer to the Glossary of Terms for definitions.) Position the cursor at the appropriate line, enter the new information and press the RETURN key.

   NOTE: These values pertain to the whole font, not only to the character being edited.

3. Defaults

   This option allows you to change or create default values for character width and/or height. (Refer to the Glossary of Terms for definitions.) These are the values that EDFONT will assign to the width/height of any newly created character.

4. Info

   To view information about the current font (number of characters in font, maximum height and maximum width) press the Select key at this option. You cannot modify these values.

5. Exit

   Select this option to quit. If you have not modified the font, then EDFONT will quit immediately. If you have made modifications, then EDFONT will ask if you want to keep or discard them. If you decide to keep the modifications, the font file will be updated with the new design information.

6. Copy From

   When you select this option, you can change the character image on the grid to another character in the same or in a different font. Enter the name of the font in which you wish to view this character and press RETURN. Next, type the character you wish to view and press RETURN. The designated character image appears on the grid.

7. Rotate

   Select this option to turn the character image on the grid anywhere from 0 to 360 degrees. When you press the Select key, EDFONT displays a circle showing the degree selection. Press the Select key to indicate the begin point, then move the cursor to the desired number of degrees on the circle. Press the Select key again. The amount of rotation you select appears as the highlighted portion of the circle with the number of degrees printed underneath. The current character image appears on the grid in the designated position of rotation.

8. Resize

   Pressing the Select key here causes EDFONT to display the x-y coordinates for the current font. To change either or both, delete the old number(s) with the CHAR DEL

key, type in the new number(s), and press RETURN. Next, position the cursor on Change Size and press the Select key. The new coordinate(s) are registered. To keep the same coordinates, simply move the cursor out of the pop-up.

9. Save
   Select this option to save your font file, on disk, at any point in its creation or modification without exiting the program.

10. Set Origin
    When you press the Select key here, a crosshair appears on the grid. By moving the cursor, you can move the crosshair to the location you wish to designate as the origin for the current character. When you press the Select key at the desired location, a small box appears on the grid to indicate the new point of origin. Remove the crosshair by moving the cursor completely off the grid.

11. Character
    When you position the cursor here and press the Select key, EDFONT displays a table showing upper- and lowercase letters. To change the selection on the table to special characters and numbers, position the cursor on Special Chars and press the Select key. To regain the letters, position the cursor on Letters and press the Select key. Each time you choose a letter or special character by pressing the Select key, that letter/special character is displayed, actual size, directly under the table.

12. To select a particular character from the table, position the cursor on that character and press the Select key, then move the cursor to Edit This One and press the Select key again. If you select a character from an existing font, EDFONT displays that character on the grid. If you select a character to be created, the cursor moves to its designated point of origin on the grid. EDFONT stores whatever you create, in the font file that you assign it, as the character that you selected it to represent.

13. Undo
    Select this option to cancel the last operation you performed. After you press the Select key, EDFONT displays the next to last version of the character.

14. Design Menu
    This menu offers the following options:

    - Pixel
      Select this option to build a design one grid square at a time. Position the cursor and press Select at the square you wish to fill.

    - Line
      This option refers to a series of pixels with a defined beginning and end point. After selecting Line and pressing the Select key, move the cursor to a begin point and press Select. Now as you move the cursor, "rubber banding" allows you to see the potential length of the line. When the line stretches to the desired length, press the Select key. All the grid squares between the two points are then filled.

    - Paint
      This option allows you to fill a continuous series of squares simply by pressing the Select key at the first square, then moving the cursor. You

don't need to press Select after each subsequent square as in Pixel. Pressing Select again turns this option off.

- Box
  Use this option to draw a box with a series of squares. Set two points on the grid to define the size and shape of the box. The first point is the center and the second point, the radius. After you press Select for the center point, use rubber banding to grow and shrink the box. Press Select at the size box you want. (See items 14 and 15 for additional options.)

- Arc
  Use this option to draw an arc with a series of squares. After setting the first point, use rubber banding as a guide for the size and sweep of your arc.

- Circle
  This option allows you to draw a circle with a series of squares. Set the center of the circle, then use rubber banding as a guide for the radius. (See items 14 and 15 for additional options.)

15. By pressing the Select key or space bar you can switch this option from On to Invert to Off and select any of these by pressing the Select key at the desired option.

On	turns on the pixel.
Invert	turns the pixel from on to off or vice versa, depending on its current state.
Off	turns off the pixel highlight.

16. This option is also switchable, in this case from Filled to Outline.

Filled	fills the space between selected points with pixel squares; available only for Box and Circle.
Outline	fills only a line between selected points with pixel squares.

17. This box shows the character/number you are currently designing on the grid in its actual size.

18. Col Row
    This box indicates the position of the cursor on the grid (in pixels) at any time.

19. This box shows the name of the current font.

20. Delete
    With this option you can delete the current character from the font along with its image from the grid.

21. Erase
    To clear the grid, position the cursor here and press the Select key.

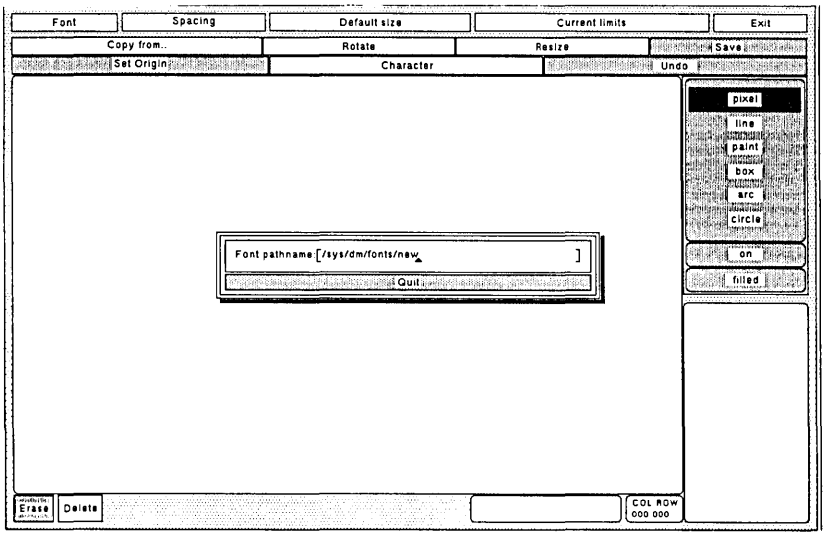
5.4. Using EDFONT

The following exercise is intended to familiarize you with EDFONT. Please follow the instructions below.

1. Invoke EDFONT by typing the following in the Shell input pad.

   $ EDFONT<RETURN>

   The EDFONT menu appears.

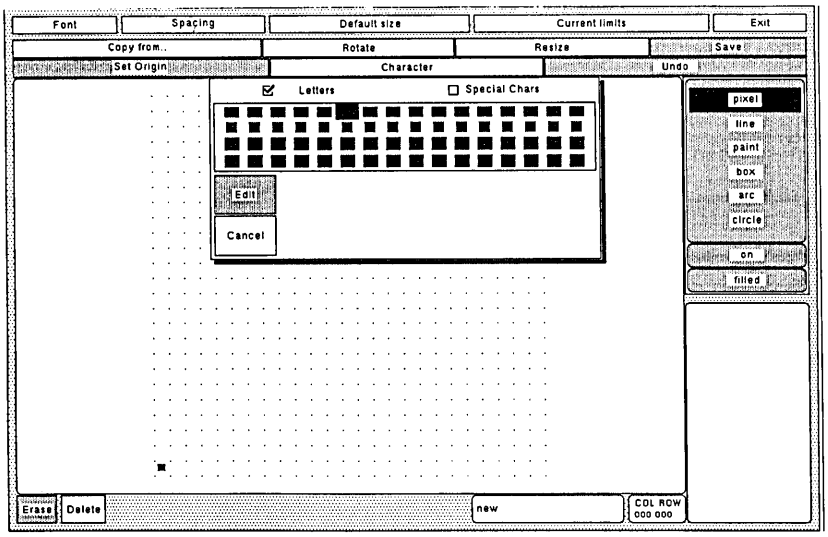


2. At "Font Pathname:" type the following:

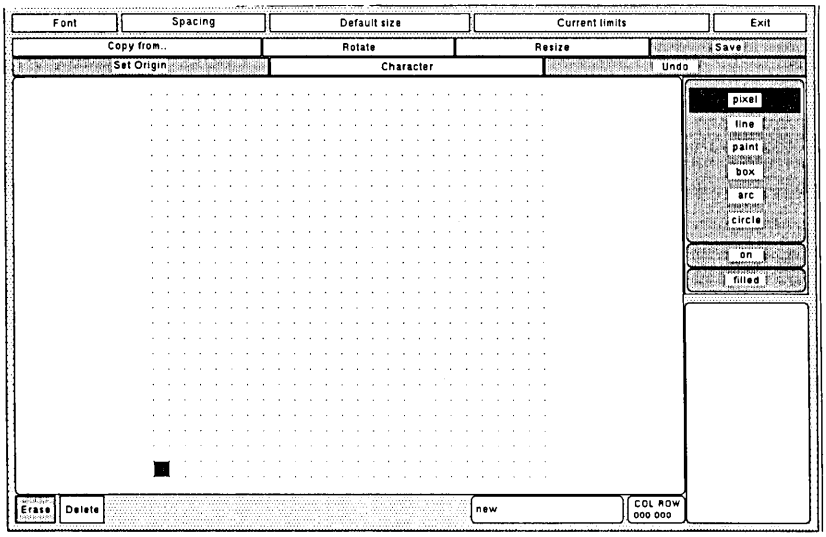
   /SYS/DM/FONTS/NEW<RETURN>

3. Move the cursor to Character. Press the Select key to display the alphabet table.

4. Position the cursor on the letter E and press the Select key. The E on the alphabet table is outlined.

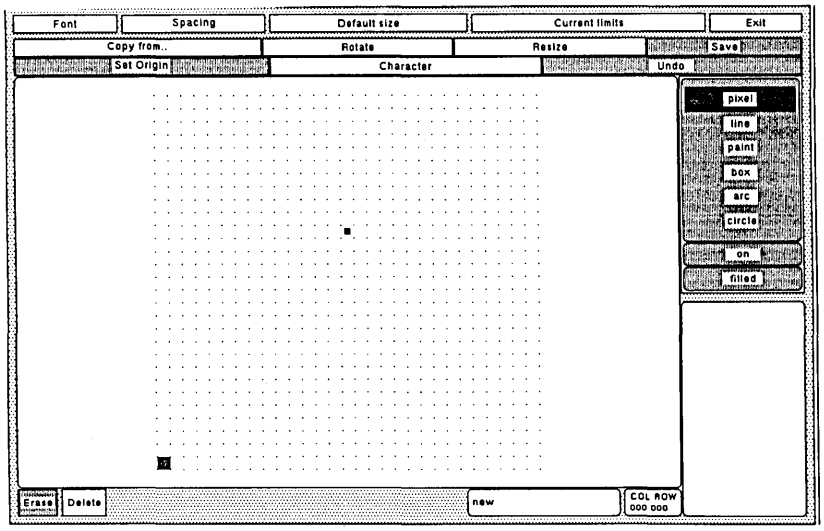


5. Position the cursor at Edit This One and press the Select key. EDFONT displays a grid with a small box on the lower left side. This box indicates the default point of origin. To change the letter's point of origin, see **SAMPLE EDFONT DISPLAY**, item 10.



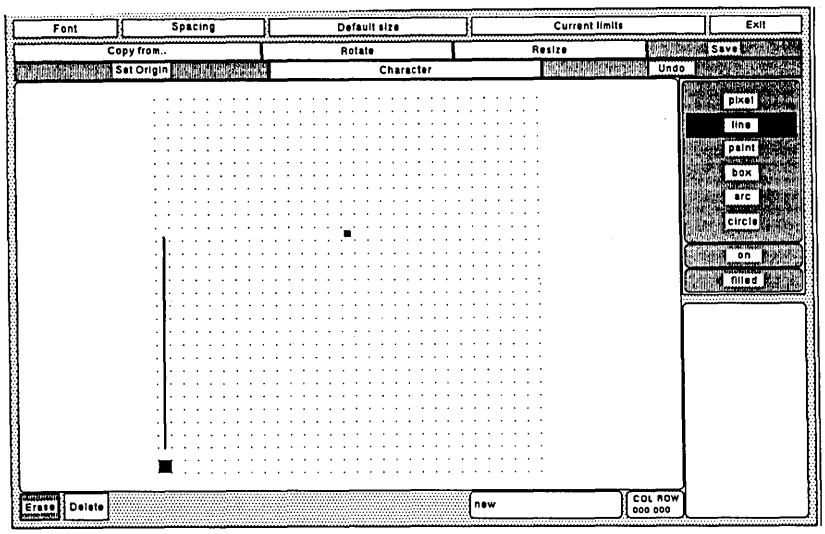
6. Check that the Pixel option in the Design Menu (item 13) and the On option (item 14) are both selected. If not, select them.

7. Begin constructing your new E by pressing the Select key at the grid square shown in the illustration below.

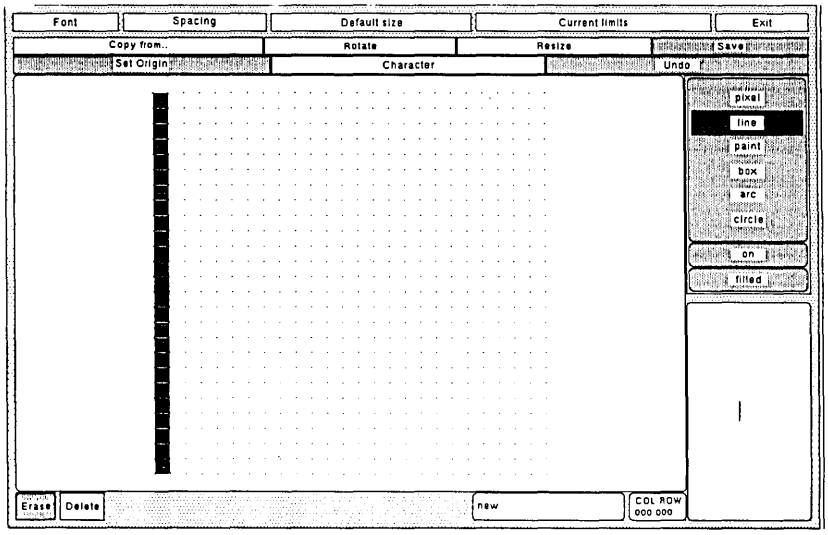


8. At the Design Menu, position the cursor on Line and press the Select key.

9. Return to the grid and press the Select key at the next box up to be filled. As you move the cursor towards the top of the screen, rubber banding allows you to see the potential length of the line.

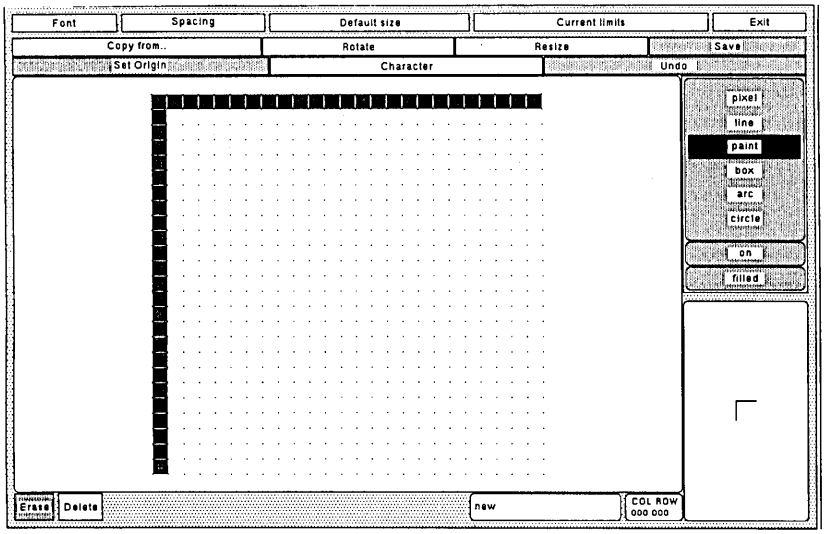


10. Now, move the cursor to the ending box and press the Select key. All squares between these two points automatically fill. You now have an image that looks like this:

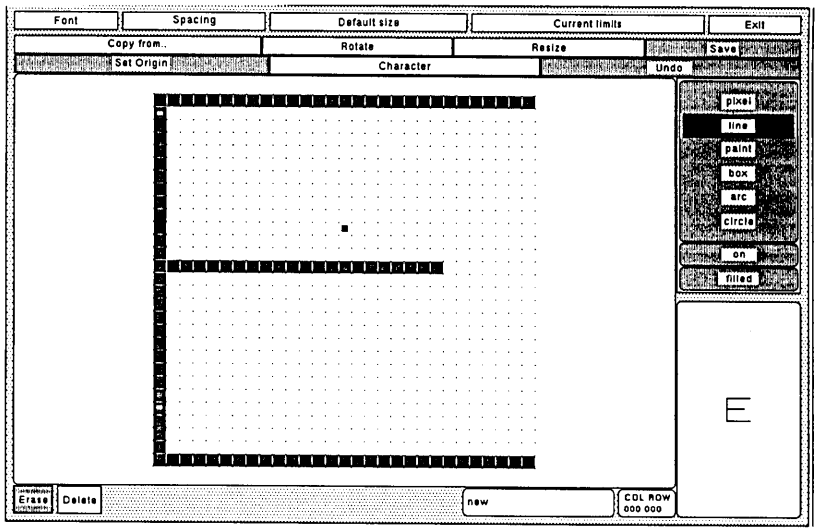


11. Move the cursor to the Design Menu and select the Paint option.

12. Move the cursor to the top square, press the Select key, then move to the right the number of squares shown below. The Paint option allows you to fill in the desired number of squares simply by moving the cursor.



13. Using the Pixel, Line, and/or Paint options, complete the letter "E".



14. Instead of exiting the program, clear the screen for another exercise by moving the cursor to Erase and pressing the Select key.

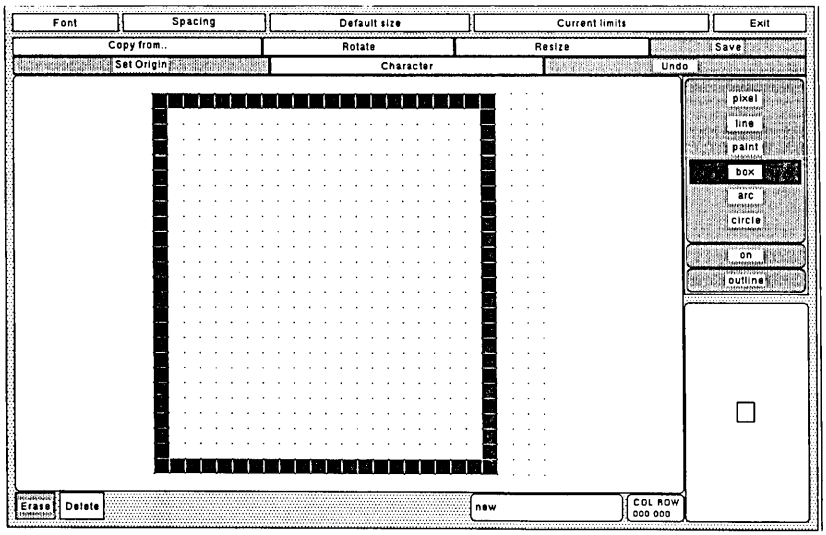
    The next exercise is intended to familiarize you with the remaining EDFONT options.

15. Check that the On and Outline options are selected. If not, select them.

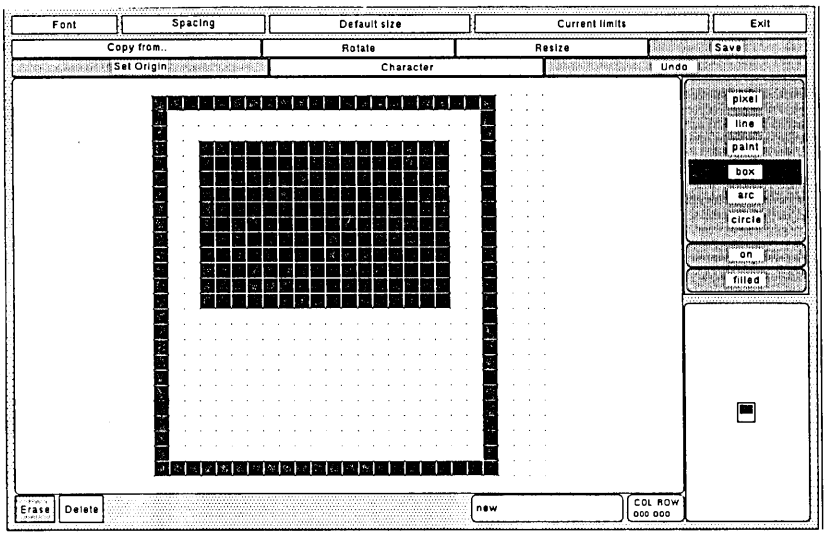
16. Position the cursor on Box and press the Select key.

17. On the grid, set up the center of the box by pressing Select. Now, using rubber banding as a guide, select the size of the box.

18. When you press the Select key, EDFONT automatically outlines the box.



19. Now create a box using the Filled option.



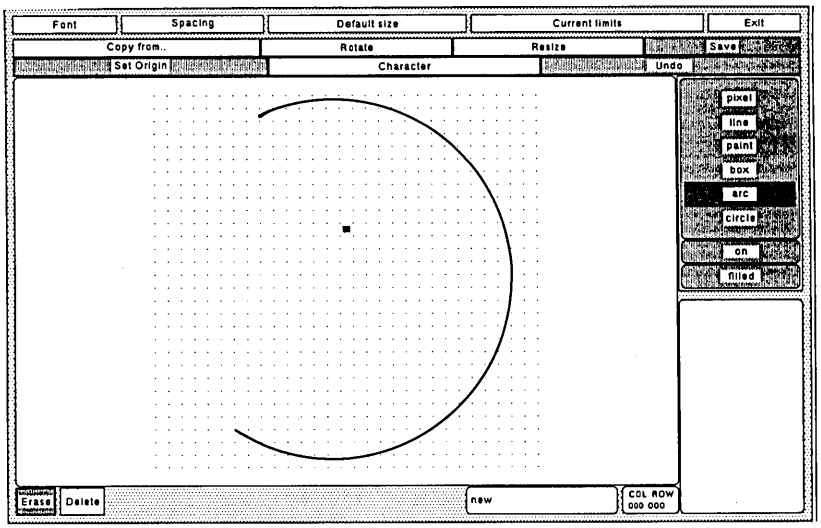
20. Use Erase to clear the screen.

21. A circle is created in exactly the same way as a box and also has Filled or Outline options. Create a circle using both the Filled and Outline options.

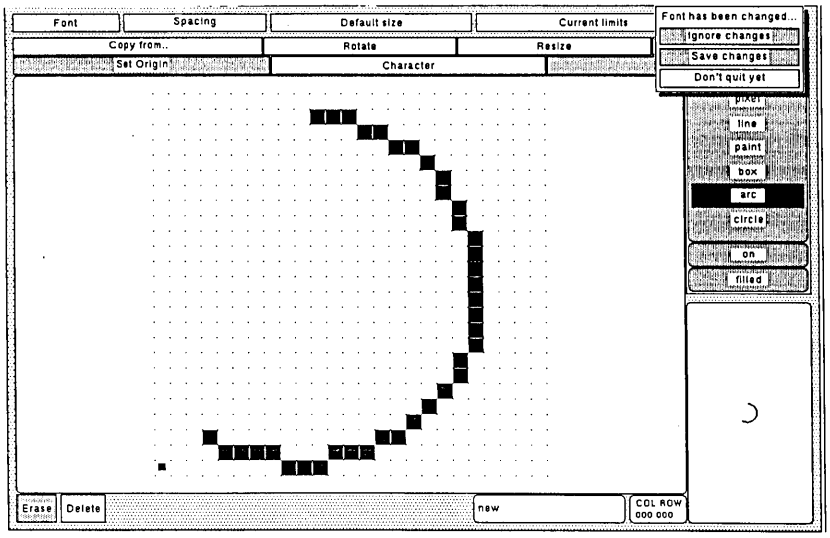
22. Use Erase to clear the screen.

23. Select Arc and return the cursor to the grid.

24. Select two points, pressing the Select key after each one. Rubber banding allows you to see the potential shape of the arc.



25. When you press the Select key a third time, EDFONT draws the arc from point to point.



26. To quit, move the cursor to Exit and press the Select key. A pop-up asks if you wish to keep or discard the modifications. Position the cursor on the desired response and press the Select key.

27. You have now completed the EDFONT exercise.

5.5. Glossary of Terms

ASCII Value — The two-digit hexadecimal number to which a given character corresponds, as follows:

oct	hex		dec	oct	hex		dec	oct	hex		dec
00	00	NUL	00	60	30	0	48	140	60		96
01	01	SOH	01	61	31	1	49	141	61	a	97
02	02	STX	02	62	32	2	50	142	62	b	98
03	03	ETX	03	63	33	3	51	143	63	c	99
04	04	EOT	04	64	34	4	52	144	64	d	100
05	05	ENQ	05	65	35	5	53	145	65	e	101
06	06	ACK	06	66	36	6	54	146	66	f	102
07	07	BEL	07	67	37	7	55	147	67	g	103
10	08	BS	08	70	38	8	56	150	68	h	104
11	09	HT	09	71	39	9	57	151	69	i	105
12	0A	NL(LF)	10	72	3A	:	58	152	6A	j	106
13	0B	VT	11	73	3B	;	59	153	6B	k	107
14	0C	FF	12	74	3C	<	60	154	6C	l	108
15	0D	CR	13	75	3D	=	61	155	6D	m	109
16	0E	RRS	14	76	3E	>	62	156	6E	n	110
17	0F	BRS	15	77	3F	?	63	157	6F	o	111
20	10	RCP	16	100	40		64	160	70	p	112
21	11	XON	17	101	41	A	65	161	71	q	113
22	12	HLF	18	102	42	B	66	162	72	r	114
23	13	XOFF	19	103	43	C	67	163	73	s	115
24	14	HLR	20	104	44	D	68	164	74	t	116
25	15	NAK	21	105	45	E	69	165	75	u	117
26	16	SYN	22	106	46	F	70	166	76	v	118
27	17	ETB	23	107	47	G	71	167	77	w	119
30	18	CAN	24	110	48	H	72	170	78	x	120
31	19	EM	25	111	49	I	73	171	79	y	121
32	1A	SUB	26	112	4A	J	74	172	7A	z	122
33	1B	ESC	27	113	4B	K	75	173	7B	{	123
34	1C	FS	28	114	4C	L	76	174	7C	|	124
35	1D	GS	29	115	4D	M	77	175	7D	}	125
36	1E	RS	30	116	4E	N	78	176	7E	~	126
37	1F	US	31	117	4F	O	79	177	7F	DEL	127
40	20	SP	32	120	50	P	80				
41	21	!	33	121	51	Q	81				
42	22	"	34	122	52	R	82				
43	23	#	45	123	53	S	83				
44	24	$	36	124	54	T	84				
45	25	%	37	125	55	U	85				
46	26	&	38	126	56	V	86				
47	27	'	39	127	57	W	87				
50	28	(	40	130	58	X	88				
51	29	)	41	131	59	Y	89				
52	2A	*	42	132	5A	Z	90				
53	2B	+	43	133	5B	[	91				
54	2C	,	44	134	5C	\	92				
55	2D	-	45	135	5D	]	93				
56	2E	.	46	136	5E	^	94				
57	2F	/	47	137	5F	_	95				

Character Descriptor — That part of a font which contains the information necessary for the system to display a character. This information is in the form of five scalar components: up, down, right, left, and width. The first four describe how many pixels the character image occupies in two directions. The fifth, width, is not related to the width of a character image as described in the font header, but is a spacing device used by EDFONT to tell the system how much space to leave between the origin of one character and the origin of the next.

Character Image — The graphic image stored in memory by the bitmap and displayed on the screen as an array of pixels.

Default Height — A value that designates the height (number of pixels occupied by a character image in the vertical direction) of any character in a font file edited by EDFONT. You can change this value.

Default Width — A value that designates the width (number of pixels occupied by a character

image in the horizontal direction) of any character in a font file edited by EDFONT. You can change this value.

Fixed Width — A font with characters that are all of the same width. These fonts produce displays and documents that are monospaced (no allowance for character width differences). Fixed width fonts are most useful whenever column alignment is important.

Font File — A file that holds one font. All of the characters that appear on a display are read from a font file. The EDFONT current font file format restricts the character image size to a maximum of 224 by 224 pixels, and the character descriptor components to a range of -127 to 127.

Font Header — That part of a font that contains data for the entire font; namely, the number of characters in the font, the maximum height of characters, the maximum width of characters, the amount of space between characters, the amount of space between lines, and the amount of space that will be skipped for an undefined character. All, except the number of characters, are expressed in pixels. The height and width of characters may vary, but no character can be higher or wider than the values listed in the font header for height and width.

Font Pathname — The path that the system must take to find the location of a specific font file. Entered at the initial EDFONT display, the font pathname conforms to the DOMAIN pathname format.

Font — The data compiled by EDFONT relating to a character image created on the grid. A font can consist of as little as one character, but the term usually refers to a group of characters related by size and/or typeface design.

Horizontal Spacing — The amount of space between each character in a font that EDFONT will skip in a horizontal direction. This is user-modifiable data, expressed in pixels, that affects the font as a whole, rather than only one character at a time. (Refer to Figure 5-1, item 2.)

Maximum Height — The value, in pixels, for the tallest character in a font.

Maximum Width — The value, in pixels, for the widest character in a font.

Monospaced Characters — See Fixed Width.

Origin — The x-y coordinate pair set by the system at which a character is displayed, where x is defined as a pixel having a "right" scalar value of 1, and y as a pixel having an "up" scalar value of 1.

Pixel — Any of the tiny picture elements that form a digitalized picture on a tv or crt screen. Pixel also describes the units of space (grid squares) that you use in EDFONT to create or edit a character. Because display pixels are too small to work with comfortably, the EDFONT grid uses enlarged pixels. However, no matter what size pixel you work with, the pixel size on the printed output and on displays other than the grid are set by the particular device you are using and are not modifiable.

Proportionally Spaced Characters
: See Varying Width.

Scalar Components
: See Character Descriptor.

Space Size
: The number of pixels to skip in the horizontal direction when an undefined character is written. When you press a key for a character not in the font, or an arrow key, the cursor moves this number. You can modify this data. (Refer to Figure 5-1, item 2.)

Varying Width
: A font in which character size varies from character to character. These fonts produce displays and documents that are proportionally spaced (with allowance for character width differences). Varying width fonts are most useful in typesetting and other document preparation applications.

Vertical Spacing
: The amount of space between each line of characters that EDFONT will skip. This is user-modifiable data, expressed in pixels, that affects the font as a whole, rather than only one character at a time. (Refer to Figure 5-1, item 2.)

Chapter 6
FMT (FORMAT_TEXT)

6.1. Introduction

FMT is a general purpose text formatting program, allowing you to arrange output text according to formatting directives embedded in the input file or typed on standard input.

By default, formatted text is written to standard output. You may redirect it to a file with the -OUT option.

ARGUMENTS

pathname
(optional)

Specify input file to be formatted. Multiple pathnames and wildcarding are permitted; however, FMT will concatenate multiple files prior to formatting. If FMT cannot find one of the specified input files, control shifts to standard input.

Default if omitted: read standard input

OPTIONS

-F n

Begin output at the first page numbered n.

-T n

Terminate output at the first page numbered higher than n.

-S

Stop before printing each page, including the first. This option is useful for paper manipulation. The prompt "Type return to begin a page" is issued only once, before the first page.

-PO n

Page Offset. Shift the entire document n spaces to the right.

-LF

List names of files as they are processed.

-OUT pathname

Specify output file. If this option is omitted, formatted text is written to standard output.

6.2. Examples

```
$ fmt mary -out mary.formatted -po 9    Format "mary" with a page offset
$                                        of 9 spaces, and write the
                                         results to "mary.formatted".
```

6.3. Using FMT

Input consists of text lines, which contain information to be formatted, intermixed with FMT request lines, which tell the program how to format the text lines. Request lines must begin with a special control character, normally a period. The Request Line Summary that follows this section lists the FMT commands.

Line Endings

If you use the fill (.fi) command, FMT adds as many words to an output line as will fit. FMT determines the line break; the <RETURN> you type when entering text does not establish line breaks on the output.

Justification

You can turn right justification on and off with the .ju and .nj commands. Strings of embedded spaces are retained so that the output line contains at least as many spaces between words as the input line. However, spaces at the beginning of input lines are output without modification.

Line Breaks

In certain cases, FMT generates breaks. A break consists of cancelling right justification for a particular line (even while you are using .ju) and beginning the next input line on a new output line.

Breaks are caused by an empty input line, by an input line that begins with a space, or by certain commands (see the "break" column on the Request Line Summary chart at the end of this section). You can also cause a break by using the .br command. Breaks are used most often to create paragraph breaks.

Tabbing

You may use FMT's tabbing feature as an aid in creating tables, lists, and other items which require multiple columns. The .ta command, followed by a list of integers, sets tabs stops at the given columns (i.e., .ta 5 10 sets tabs at columns 5 and 10.) The .tc command declares the tab character: for instance, .tc \ would cause FMT to move output to the next tab stop whenever a backslash (\) is encountered in the input text. Finally, the .rc command establishes a "replacement character" for use in tabbing operations. The replacement character is printed from the current point to the next tab stop. This can be useful when creating a table of contents, for instance.

The following input file:

```
.ta 50                  Set a tab stop at column 50.
.tc \                   Declare the tab character (\).
Section\Page
.rc                     Declare the replacement character (.) .
.sp
Chapter 1\1
Chapter 2\15
Chapter 3\25
.rc                     Return the replacement character to blank.
```

would produce:

```
Section                                          Page
Chapter 1.....................................1
Chapter 2.....................................15
Chapter 3.....................................25
```

Running Titles

Running titles may appear at the top and bottom of every page. A title line is a single line comprising three textual fields: the first is placed flush with the left margin, the second is centered, and the third is placed flush with the right margin. The first nonblank character in the title is used as the delimiter to separate the three fields. FMT replaces any pound signs (#) in a title with the current page number, and replaces any percent (%) characters with the current date.

Number Registers

FMT provides 26 number registers named a through z. The .nr command defines a number register. The command

```
.nr x m
```

sets number register x to m;

```
.nr x +m
```

increments number register by m; and

```
.nr x -m
```

decrements x by m. To insert the value of number register x in the text or a command line, specify @nx. Number registers are useful for running counts. For example, table numbers in this manual are stored in number register t. The register is set to zero at the beginning of every chapter and is incremented each time a new table appears.

Fixed Spaces

You can insert a fixed number of spaces anywhere in the text except header and footer titles. Fixed spaces are most often used to override the justification command (.ju), which inserts extra spaces to justify a line.

By default, the pound sign character (#) represents a fixed space. For example, placing three pound signs in a string results in exactly three spaces. The phrase

```
Plot###data
```

in the source file would appear in the formatted output as:

```
Plot   data
```

To print an actual pound sign, "escape" it with an at sign. For example:

```
16@#100 appears as 16#100
```

To change the default to another character, use the .sc command.

In-Line Boldfacing and Underlining

In-line flags allow you to mark text to be underlined or boldfaced, without giving the command on a separate line. The flags are:

```
{_underlines these words_}
{!boldfaces these words!}
```

These underlining and boldfacing flags can be used instead of the .ul, .cu, and .bd commands, which must be on lines separate from the text to which they apply.

The underlining flags apply to the entire string they surround; hence embedded spaces (including fixed spaces) will be underlined.

Each of the character sequences {_, {!, _}, and !} is treated as a unit. Therefore you need only one at sign, preceding the sequence, to nullify its special meaning as a flag.

Defining New Commands

You can define your own commands with the .de command. For example, the following sequence defines a paragraph command named .pg.

```
.de pg
.sp
.ti +3
.en
```

Defined commands can be invoked with arguments, separated by blanks or tabs. Within a command definition, arguments are referenced using ^1, ^2, etc., up to a maximum of nine arguments. Omitted arguments default to the null string, and ^0 refers to the command name itself. For example, the following version of the paragraph command uses the argument to determine the amount of indentation.

```
.de pg
.sp
.ti +^1
.en
```

This command could be invoked by:

```
.pg 3
```

to get the same effect as the previous version.

Inserting Files

The .so command,

```
.so filename
```

causes the contents of the named file to be inserted in place of the command line. You can nest .so commands.

6.4. Diagnostics

If your input file contains an invalid FMT request, the following message is displayed.

```
"string": unrecognized command ignored
```

The names of number registers must be a through z, or the following message is displayed.

```
invalid number register name
```

If no name was supplied with the .de command, the following message will be displayed.

```
missing name in command definition
```

If the limit for nesting included source files has been exceeded, the following message will be displayed.

```
.so commands nested too deeply
```

If the buffer holding input characters has been exceeded, the following message will be displayed.

```
too many characters pushed back
```

6.5. Request Line Summary

Request	Initial	Default	Break	Meaning
.#			no	Ignore this line. Precede comment lines with this symbol.
.bd n		n=1	no	Boldface the next n lines
.bp n	n=1	n=n+1	yes	Begin new page and number it n
.br			yes	Break
.cc c	c=.	c=.	no	Control character becomes c
.ce n		n=1	yes	Center the next n input lines
.cu n		n=1	no	Continuously underline next n input lines
.de xx			no	Command xx; ends at .EN
.ef /l/c/r			no	Foots on even pages are l(eft), c(enter), r(ight). '#' and '%' produce page number and date, respectively.
.eh /l/c/r			no	Heads on even pages are l(eft), c(enter), r(ight). '#' and '%' produce page number and date, respectively.
.en			no	Terminate command definition
.fi	yes		yes	Begin filling output lines
.fo /l/c/r			no	Foot titles are l(eft), c(enter), r(ight) '#' and '%' produce page number and date, respectively.
.he /l/c/r			no	Head title is l(eft), c(enter), r(ight) '#' and '%' produce page number and date, respectively.
.in n	n=0	n=0	yes	Set left margin to column n+1
.ju	yes	yes	no	Begin justifying filled lines
.ls n	n=1	n=1	no	Set line spacing to n
.m1 n	n=3	n=3	no	Space between top of page and head

```
Request  Initial  Default  Break  Meaning  (continued)

.m2 n      n=2      n=2      no     Space between head and text
.m3 n      n=2      n=2      no     Space between text and foot
.m4 n      n=3      n=3      no     Space between foot and bottom
.ne n               n=0      y/n    Need n lines; break if new page
.nf        no                yes    Stop filling
.nj        no                no     Stop justifying
.nr x m    m=0      m=0      no     Set number register x to m,
                                    -m, +m for decrement, increment
.of /l/c/r                   no     Foots on odd pages are l(eft), c(enter),
                                    r(ight).  '#' and '%' produce page number
                                    and date, respectively.
.oh /l/c/r                   no     Heads on odd pages are l(eft), c(enter),
                                    r(ight).  '#' and '%' produce page number
                                    and date, respectively.
.pl n      n=66     n=66     no     Set page length to n lines
.po n      n=0      n=0      no     Set page offset to n spaces
.rc c                       no     Tab replacement character is c
.rm n      n=65     n=65     no     Set right margin to column n
.sc c      c=#      c=#      no     Change fixed space character to c
.so file                    no     Switch input to file
.sp n               n=1      yes    Space n lines, except at top of page
.st n               n=0      yes    Space to line n from top; -n spaces to line
                                    n from bottom
.ta n1 n2 ...               no     Set tab stops at columns n1, n2, ...
.tc c                       no     Tab character is c
.ti n               n=0      yes    Temporarily indent next output line n spaces
.ul n               n=1      no     Underline words in the next n input lines

IN-LINE FLAGS

{_c_}                       no     Underline characters enclosed in braces
{!c!}                       no     Boldface characters enclosed in braces
@nc                         no     Replace with value in number register c
#                           no     Insert literal blank

KEY

n   denotes numerical values
t   denotes titles
c   denotes single characters
```

Signed numbers signify relative changes to a quantity; unsigned numbers signify absolute settings. Unless otherwise noted, omitted n fields set the value to 1, omitted t fields are empty, and omitted c fields restore the default character.

Chapter 7
INVOL (Initialize_Volume)

7.1. Introduction

INVOL initializes physical disk volumes, creates logical volumes, and maintains badspot lists. Once initialized, a volume can be mounted with the MTVOL (MOUNT_VOLUME) command, or can be used to bootstrap the operating system, providing it contains the necessary files.

Disks that have been corrupted by system crashes or network failure can be salvaged without having to be reinitialized, thus saving existing data. See Appendix SALVOL for details on SALVOL (SALVAGE_VOLUME).

Logical Volumes

Various logical organizations of a disk are shown in Figure DISK_ORG. Each disk has a physical volume label, created when the disk is initialized. Following the physical volume label are one or more logical volumes. Each logical volume is a logically independent storage area and has a name. A logical volume can span all or part of a physical disk. INVOL can create logical volumes when it first initializes the disk, and can add logical volumes later, by partially initializing a physical volume.

Space on the disk that is not allocated to a logical volume is called a "vacancy." For example, on a partially initialized disk, a vacancy follows the logical volumes. After deletion of a logical volume, a vacancy takes its place. If adjacent logical volumes are deleted, one vacancy is formed. When initializing a partial physical volume, you can fill any vacancy with one or more logical volumes.

Badspots

Badspots are defective blocks that cannot be used for data storage. During manufacturing, we write a list of badspots onto every Winchester disk. Normally, you need not alter the factory-supplied badspot list. However, if you determine that the list is incomplete or inaccurate -- as the result of repeated disk I/O errors at the same location -- you can add new badspots to the list (see operation nine at the end of this appendix).

Floppy disks typically do not have factory badspots. In cases of recurrent floppy disk I/O errors, the simplest solution is to replace the floppy disk.

Examples 1, 2, and 3 in Figure DISK_ORG show three possible disk organizations. In Example 1, a single logical volume occupies the entire disk. Example 2 shows a partially initialized disk where a vacancy follows the logical volumes. Example 3 shows a disk on which one or more logical volumes have been deleted, and vacancies thereby created.

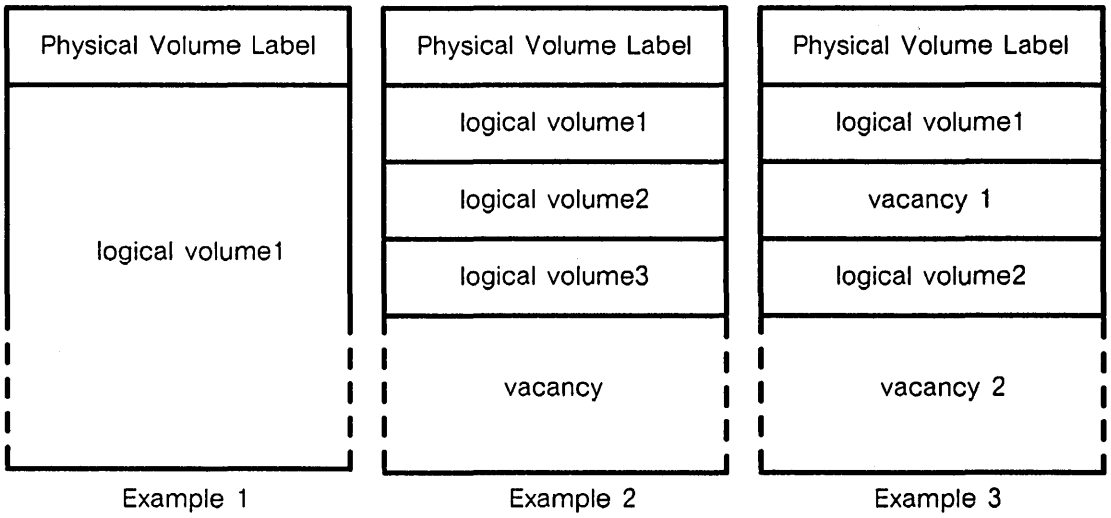


Figure 7-1. Sample Logical Disk Organizations

7.2. Invoking INVOL

INVOL can run on-line under the AEGIS Shell or can run as a stand-alone utility in the Mnemonic Debugger. To invoke the program from the Shell, type:

```
$ INVOL
```

From the Mnemonic Debugger, type

```
> EX INVOL
```

In either case, the program prompts you for all required information. After gathering the required information, INVOL performs the requested operation.

Exiting

INVOL exits normally when you signal completion of a particular operation. To exit from INVOL before an operation is complete, type CTRL/Z or CTRL/Q (under AEGIS) or CTRL/<RETURN> (under the Mnemonic Debugger). Be careful, though. If you exit while INVOL is initializing the disk, the logical volume being initialized will be unusable. If INVOL is initializing a new physical volume, the entire volume is unusable and will still require initialization before it can be used.

7.3. Operations

INVOL can perform ten operations:

1. Initialize a virgin physical volume.

2. Partially initialize a physical volume, preserving existing logical volumes.

3. Reinitialize a logical volume.

4. Delete a logical volume.

5. List logical volumes.

6. List badspots on a physical or logical volume.

7. Input and record badspot information.

8. Create or modify an os paging file on an existing logical volume.

9. Add to existing badspot list.

10. Set or display the sector interleave factor.

The following sections describe these operations.

1. Initializing a Virgin Physical Volume

   Every new disk must be initialized before it can be used. When you initialize a new disk, all existing data on the disk are overwritten. *Do not* initialize a disk that contains any data you need to save. We initialize Winchester disks during the manufacturing process, before installing the system software.

   To initialize a new disk, follow this procedure:

   a. INVOL asks which operation to perform. Type 7 to create or replace the badspot list. (See "Recording Badspot Information"). Type 9 if you want to add to the existing badspot list. Otherwise, type 1 to initialize a new physical volume.

   b. Specify the type of disk to initialize. INVOL prompts with:

```
Disk types are:
     W - Winchester
     S - Storage module
     F - 2-sided double density floppy
```

      Respond by typing W, S, or F. To specify a unit number, append 0 or 1 to the letter. For example, S1 denotes storage module unit 1. Unit 0 is the default.

   c. Choose one of the following verification options:

```
1 - No verification
2 - Write all blocks on the volume
3 - Write and reread all blocks on the volume
```

      If you choose no verification (option 1), INVOL does not read or write to the disk, except to create the volume structure. This option is the fastest, but means that the disk is not verified until it is mounted and read or written by AEGIS.

If you choose the second option, INVOL attempts to write to each block on the disk. The third option, writing and rereading all blocks on the volume, is the safest but also the slowest. For example, to format a complete 33MB Winchester, option 1 requires about five minutes, option 2 requires about fifteen minutes, and option 3 requires about 30 minutes.

A Note on Floppies

If a floppy disk is initialized with INVOL on a busy node, there is a small chance that a format operation will fail, but that the failure will not be reported to INVOL. For this reason, INVOL writes each block during floppy initialization, even for verification Option 1. If a write fails after a purportedly successful format, INVOL will print the message:

```
format failed for daddr <disk_address>:<write status>--retrying format
```

and will reformat (and rewrite) the track in error. This happens whether or not the floppy has been previously initialized.

d. Enter the average file size, when prompted:

```
Expected average file size,in blocks(CR for default-5 blocks):
```

Press <RETURN> to accept the default value of 5 blocks. Supplying a relatively accurate value for the average file size can save space on the disk, because the volume table of contents (a system table) will be allocated more efficiently.

e. INVOL requests the size (in blocks) and name of each logical volume to be created. After each entry, INVOL tells you how much space remains. After entering the size and name of all logical volumes, enter a blank line to terminate input. A physical volume can contain up to 10 logical volumes.

For example:

```
There are 1231 blocks available.

volume 1:  1231,vol1
```

The logical volume size must be at least 30 blocks, and must be a multiple of the track size for the disk. If you specify a logical volume size that is not a multiple of the track size, INVOL rounds it up to the next multiple track size, and informs you. Note that the physical volume label occupies the first block on the volume. Thus, the size of the first logical volume is always one less than a multiple of the track size.

Logical volume names are optional, and are used only when INVOL lists the logical volumes on the disk (option 5). You cannot change the name of a logical volume after creating it.

f. INVOL requests badspot information by asking whether or not you wish to

use the prerecorded badspot list shipped with the disk. ANSWER "YES". To erase the existing list, ANSWER "NO". If you want to initialize the physical badspot list on a virgin disk, use operation seven, not operation one. Use operation nine to add to an exisiting list. You must have a hardcopy of the badspots in order to enter them in. INVOL has retained the badspot prompt in operation one only for compatibility with existing Shell files. After your affirmative response, INVOL displays the badspot list, indicating the physical disk address, cylinder, head, sector, and byte offset range.

If, in later operations, you wish to provide your own badspot information, you can enter badspots for a Winchester disk or storage module in two forms: as hexadecimal physical disk addresses, or as physical cylinder/head/byte addresses, in the form cylinder-head byte. Terminate badspot entry with a blank line.

For a floppy disk, enter badspots as hexadecimal physical disk addresses, one per line. Terminate badspot entry with a blank line.

g. INVOL asks for the name of the physical volume. Then it initializes the disk. As formatting proceeds, INVOL displays milestone messages to report its progress. It also displays a message for each volume it initializes, and another when it completes.

h. INVOL asks if you have any more requests. Type Y to return to step a, or N to return to the calling program (AEGIS command Shell or Mnemonic Debugger).

2. Partially Initializing a Volume

Using operation 2, you can partially initialize a volume, that is, add logical volumes to a physical volume, while preserving the existing logical volumes. Follow this procedure:

a. INVOL asks which operation to perform. Type 2 to partially initialize a disk.

b. Specify the type of disk to initialize. (See operation 1b.)

c. INVOL prints a list of the logical volumes and vacancies on the disk. If the disk has more than one vacancy, INVOL asks where to place the new logical volume by requesting a vacancy number. Indicate the vacancy that you want INVOL to use by entering its number. If there are logical volumes following the vacancy that you choose, INVOL prints a warning message and then automatically increments the volume numbers of those succeeding volumes by one.

d. Choose a verification option for the logical volume being initialized. (See operation 1c.)

e. Enter the expected average file size, in blocks. (See operation 1d.) Press <RETURN> for the default value, 5 blocks.

f. Enter the name and size of each logical volume to be formatted. (See operation 1e.) After each specification, INVOL informs you of how much space is available. Terminate input with a blank line. A physical volume may have up to ten logical volumes.

g. Enter badspot information. (See operation 1f.) Terminate badspot entry with a blank line.

h. Enter the name of the physical volume. (See operation 1g.)

i. INVOL asks if you have any more requests. Type Y to return to step a, or N to return to the calling program (AEGIS command Shell or Mnemonic Debugger).

3. Reinitializing a Logical Volume

You can reinitialize a logical volume, retaining its size and name, with operation 3. All existing data in the volume will be lost. This operation is useful for reinitializing floppy disks, where one logical volume typically occupies the entire physical volume.

To reinitialize a single logical volume, use the following procedure:

a. INVOL asks which operation to perform. Type 3 to reinitialize a logical volume.

b. Specify the type of disk to initialize. (See operation 1b.)

c. Choose a verification option: no verification, write all blocks, or write and reread all blocks. (See operation 1c.)

d. Enter the expected average file size, in blocks. (See operation 1d.) Press <RETURN> for the default value, 5 blocks.

e. INVOL asks if you have any more requests. Type Y to return to step a, or N to return to the calling program (AEGIS command Shell or Mnemonic Debugger).

4. Deleting a Logical Volume

To delete a logical volume, use the following procedure:

a. INVOL asks which operation to perform. Type 4 to delete a logical volume.

b. Specify the type of disk from which the volume will be deleted. (See operation 1b.)

c. Enter the number of the logical volume to delete. You can determine the logical volume numbers present on a disk with operation 5.

d. INVOL asks if you have any more requests. Type Y to return to step a, or N to return to the calling program (AEGIS command Shell or Mnemonic Debugger).

NOTE: INVOL renumbers the logical volumes following the deleted volume.

5. Listing Logical Volumes

To list the logical volumes on a disk, use the following procedure:

a. INVOL asks which operation to perform. Type 5 to list the logical volumes on a disk.

b. Specify the type of disk. (See operation 1b.) INVOL lists the volumes on that disk.

c. INVOL asks if you have any more requests. Type Y to return to step a, or N to return to the calling program (AEGIS command Shell or Mnemonic Debugger).

6. Listing Badspots

To list the badspots in one or more logical volumes, or for the physical volume, use the following procedure:

a. INVOL asks which operation to perform. Type 6 to list badspots.

b. Specify the type of disk. (See operation 1b.)

c. Specify the badspots to be listed, by entering one of the following:

```
a logical volume number
ALL
PHYS
```

Enter a logical volume number to list the badspots in that logical volume. (The logical volumes present on a disk can be listed using operation 5.) Enter ALL to list the badspots in all logical volumes. Enter PHYS to list all badspots on the disk.

d. INVOL asks if you have any more requests. Type Y to return to step a, or N to return to the calling program (AEGIS command Shell or Mnemonic Debugger).

7. Recording Badspot Information

Using operation 7, you can create or replace the badspot list on the disk. (Use operation 9 to add badspots to an existing badspot list.)

a. INVOL asks which operation to perform. Type 7 to enter the disk's badspot list.

b. Enter the location of the badspots, one per line. (See operation 1f for the proper format.) Terminate badspot information with a blank line.

c. After you have typed in the list, INVOL asks you to check for errors. If you made any errors in the list, you must retype the entire list by returning to step a and beginning again.

d. INVOL asks if you have any more requests. Type Y to return to step a, or N to return to the calling program (AEGIS command Shell or Mnemonic Debugger).

8. Creating or Modifying an OS Paging File on an Existing Logical Volume

You can create an operating system (OS) file or modify the size of an existing one. The OS paging file is required if you intend to run the AEGIS operating system off of this logical volume.

To create or modify an OS paging file, perform the following steps:

a. INVOL asks which operation to perform. Type 8.

b. Specify disk type. (See operation 1b.)

c. Specify logical volume number. The logical volumes present on a disk may be listed using operation 5.

d. If an OS paging file already exists on this volume, INVOL displays the file's current size and asks if you want to change it. If you reply Y, INVOL proceeds to step e. If you reply N, INVOL skips to step f.

e. INVOL prompts you to enter the number of pages you want in the OS paging file. Press <RETURN> to use the default, 352 pages. Type "0" (zero) to delete an existing paging file, or specify any number of pages between 1 and 288. If the size you enter is larger than the current OS paging file, INVOL displays milestones as it initializes new disk records.

f. INVOL asks if you have any more requests. Type Y to return to step a, or type N to return to the calling program (AEGIS command Shell or Mnemonic Debugger).

9. Adding Badspot Information

Using operation 9, you can add to the disk's existing badspot list. Run SALVOL when you complete this option.

a. INVOL asks which operation to perform. Type 9 to add to the disk's badspot list.

b. Enter the location of the badspots, one per line. (See operation 1f for the proper format.) Terminate badspot information with a blank line.

c. After you have typed in the list, INVOL asks you to check for errors. If you made any errors in the list, you must retype the entire list by returning to step a and beginning again.

d. INVOL asks if you have any more requests. Type Y to return to step a, or N to return to the calling program (AEGIS command Shell or Mnemonic Debugger).

10. Setting or Displaying the Sector Interleave Factor

Using operation 10, you can set or display the sector interleave factor for a volume. The correct interleave factor is set when a logical volume is created. However, as performance improvements are made, it may become necessary to change it to achieve optimal block read/write rates. Operation 10 displays the current value and the optimal value which we recommend.

a. INVOL asks which operation to perform. Type 10 to set or display the sector interleave factor.

b. Specify disk type. (See operation 1b.)

c. INVOL displays a list of logical volumes for that physical volume. Specify the appropriate logical volume number. INVOL then displays the current sector interleave factor and the value which we recommend.

d. INVOL prompts for the new interleave factor. If you do not wish to change the interleave factor, enter a carriage return.

e. INVOL asks if you have any more requests. Type Y to return to step a, or type N to return to the calling program (AEGIS command Shell or Mnemonic Debugger).

Chapter 8
ITEST (IOS_TEST)

8.1. Introduction

ITEST is a program for testing type managers that manage input and output to objects. ITEST allows you to open a stream to any type of object and then use selected IOS calls on the open stream. With ITEST, you can open streams to existing or new objects. For more information on using ITEST to test type managers, see Using the Open System Toolkit for Extending the Streams Facility.

OPTIONS

-INIT

Call the IOS_$INITIALIZE routine (within a type manager) at startup time.

8.2. Command Summary

ITEST prompts for commands. Any valid, unambiguous prefix of one of the following commands will suffice. Each command calls the IOS call with a similar name. For example, the CLOSE command calls IOS_$CLOSE.

SYNTAX (abbreviation shown in uppercase)	FUNCTION
CHANGE_PATH_NAME stream-id pathname	Changes the pathname of an object.
CLOSE stream-id	Closes a stream.
CREATE create-mode [open-options] pathname typename	Creates an object and opens a stream to it.
DELETE stream-id	Deletes an object and closes the associated stream.
DUP stream-id stream-id	Creates a copy of a specified stream ID.
EQUAL stream-id-1 stream-id-2	Determines whether two stream IDs refer to the same object.

EXPORT stream-id	Simulates stream passing via a PGM_$INVOKE system call. This command tests a type manager's export and import procedures.
FORCE_WRITE stream-id	Forcibly writes an object and the directory containing the object to "stable" storage.
GET [put-get-option] stream-id count	Copies data from a stream into a buffer.
INQ_BYTE_POS [pos-opt] stream-id	Returns the byte position of the stream marker. If you omit a position option, the default is the current position of the stream marker.
INQ_CUR_REC_LEN stream-id	Returns the length of the record at the current stream marker.
INQ_FILE_ATTR stream-id	Returns object usage attributes.
INQ_FLAGS stream-id	Returns the attribute set of an object's type manager.
INQ_FULL_KEY [pos-opt] stream-id	Returns a full seek key. If you omit a position option, the default is the current position of the stream marker.
INQ_PATH_NAME [name-type] stream-id	Returns the pathname of the object to which a stream is open. If you omit a name-type, the default is -ROOT.
INQ_REC_POS [pos-opt] stream-id	Returns the record position of the stream marker. If you omit a position option, the default is the current position of the stream marker.
INQ_REC_REM stream-id	Returns the number of bytes remaining in the current record.
INQ_REC_TYPE stream-id	Returns the record type of an object.
INQ_SHORT_KEY [pos-opt] stream-id	Returns a short seek key. If you omit a position option, the default is the current position of the stream marker.
INQ_TYPE_UID stream-id	Returns the type UID of an object.
LOCATE stream-id count	Reads data from a stream, returning a pointer to the data (rather than copying the data to a buffer).
OPEN [open-options] pathname	Opens a stream to an existing object.

PUT [put-get-options] [-NL] stream-id string	
	Writes data into an object. The -NL option inserts a newline character at the end of the string, the default writes only the data.
REPLICATE stream-id stream-id	Creates a copy of a specified stream ID.
SEEK [-RELATIVE [-MINUS]] [-RECORD] stream-id count	
	Performs an absolute or relative seek using byte or record positioning. If you omit the -RELATIVE option, the default is an absolute seek. If you omit the -MINUS option, the default is to seek forward, towards the end of the file. If you omit the -RECORD option, the default is to seek by bytes.
SEEK_FULL stream-id recadr byteadr	Performs a seek using a full (8-byte) seek key.
SEEK_SHORT stream-id key	Performs a seek using a short 4-byte seek key.
SEEK_TO_BOF stream-id	Positions the stream marker to the beginning of an object.
SEEK_TO_EOF stream-id	Positions the stream marker to the end of an object.
SWITCH stream-id stream-id	Switches a stream from one stream ID to another stream ID.
TRUNCATE stream-id	Deletes the contents of an object following the current stream marker.

Use one of the following to specify 'create-mode'.
These options correspond to the IOS_$CREATE_MODE_T data type.

-NO_PRE_EXIST	Returns an error if object already exists.
-PRESERVE	Saves contents of object, if it exists, opens object, and positions stream marker at BOF.
-RECREATE	Deletes existing object and creates new one of the same name.
-TRUNCATE	Opens object, then truncates the contents.
-MAKE_BACKUP	Creates a backup (.BAK) file when closed.
-LOC_NAME_ONLY	Creates a temporary unnamed object, uses pathname to specify location of object, and locates it on the same volume.

Use one of the following to specify 'name-type'.
These options correspond to the IOS_$NAME_TYPE_T data type.

-LEAF	Specifies leaf name regardless of object's name.
-NDIR	Specifies leaf name if object's name is a name in current naming directory; otherwise, specifies full pathname.
-NODE	Specifies name relative to the root directory if object is a name in boot volume; otherwise, specifies full pathname.
-NODE_DATA	Specifies leaf name if object's name is a name in current 'node_data directory; otherwise, specifies full pathname.
-ROOT	Specifies full pathname, for example, //node/sid/file.
-WDIR	Specifies leaf name if object's name is a name in current working directory; otherwise, specifies full pathname.

Use one or more of the following to specify 'open-options'.
These options correspond to the IOS_$OPEN_OPTIONS_T data type.

-NO_Delay	IOS_$OPEN does not wait for the open to complete before returning.
-Write	Permits writing data to a new object.
-UNREGulated	Permits concurrent writing (unregulated read and write access) to the object.
-END_OF_FILE	Positions stream marker at EOF at open.
-INQuire_only	Opens object for attribute inquiries only.

Use one of the following to specify 'pos-opt'.
These options correspond to the IOS_$POS_OPT_T data type.

-BOF	Returns key for end-of-file (EOF) marker.
-EOF	Returns key for beginning-of-file (BOF) marker.

Use one or more of the following to specify 'put-get-options'.
These options correspond to the IOS_$PUT_GET_OPTS_T data type.

-COND	Gets or puts data conditionally. If the data is not available, returns with a status indicating that condition.
-PREview	Determines if a put/get would succeed, but does not actually perform data transfer.
-PARTial_record	Puts the data, but does not terminate the record.
-NO_REC_bndry	Ignore record (line) boundaries.

8.3. Debugging Managers

Under normal conditions, user-written managers are dynamically loaded into the opener's address space. While you can use ITEST to test such managers, the manager code itself can not be debugged using DEBUG at the present time.

To debug managers using ITEST, you must follow the convention that your manager contains no "main program" (PROGRAM in Pascal, "main" in C). Instead, the initialization for your manager (the part that calls TRAIT_$MGR_DCL, etc.) should be placed in a procedure named "ios_$initialize". To debug your manager module using ITEST, bind all the pieces of your manager together with "/com/itest". Then use DEBUG on the result of the bind and give the -INIT switch. For example:

```
$ bind -b my_itest <<!
my_mgr.bin
my_mgr_uid.bin
/sys/traits/io_traits
/com/itest
!
$ debug -src my_itest -init
```

Chapter 9
SALVOL (Salvage_Volume)

9.1. Introduction

Each logical volume is divided into disk blocks. (See Appendix 7, INVOL, for a detailed description of logical disk volumes.) SALVOL verifies, and if necessary, corrects the tables that describe the allocation of disk blocks to the files stored on the disk. SALVOL also returns all blocks that are no longer in use to the free space pool. (Those blocks allocated to temporary files or to deleted portions of permanent files are included).

The label of every logical volume on every physical disk contains its last mount and dismount times. When the Mnemonic Debugger loads AEGIS into memory, it checks these two times. A system crash or improper dismount may mean that the volume has been mounted since its last proper dismount. If so, AEGIS warns you that the volume needs salvaging, and then proceeds to run SALVOL automatically, provided that the NORMAL/SERVICE switch is set at NORMAL.

Even though SALVOL is not run automatically when your node is in SERVICE mode, you still receive a warning message if some boot volume error is detected. Although you can proceed after AEGIS's "Disk needs salvaging" message, we strongly encourage you to take the time to salvage the logical volume. As part of its normal operation, AEGIS periodically updates the essential data on each disk volume. When the volume is wrested from the system's control by either an improper dismount or a system crash, the information on that volume is at most a few minutes out of date. Proper salvaging can identify and correct almost all the inconsistent block allocation information with no loss of files. Of course, memory-resident data buffers that were not written to the volume cannot be reconstructed.

If you mount the volume without running SALVOL, AEGIS incorrectly assumes that the block allocation information is correct. Extensive damage may occur to files that would otherwise be intact. The need to salvage a volume is satisfied only by using SALVOL on that volume.

9.2. Invoking SALVOL

SALVOL can run under the AEGIS operating system or as a stand-alone utility. To invoke the program under the operating system, type:

```
$ SALVOL
```

If you invoke SALVOL under the operating system, it cannot salvage mounted volumes (you must dismount them), nor can it salvage the boot volume on which the operating system is running.

To invoke the program as a stand-alone utility, type the following command to the Mnemonic Debugger:

```
>EX SALVOL
```

In either case, SALVOL identifies itself and prompts for the type of disk (Winchester, storage module, or floppy) and the number of the logical volume to be salvaged, as indicated below.

```
$ salvol

                Salvage_volume - Version 6.0

Controller type (W=winchester,S=storage module,F=floppy)? w
Salvol options:
     -A : read all blocks in all files
     -V : verify only (don't write anything to disk)
     -U : print uids & vtocx's as well as any filenames
Please input lv_num [-option]... :
```

Controller Types

In response to the "Controller type ?" prompt, you must specify which storage device you wish to salvage. Do this by typing

type[unit]

where

type indicates the device type. Valid types are

- **W** for Winchester disks
- **S** for storage modules
- **F** for floppy disks

unit is an integer indicating the unit number of the device. There must be no blanks between the 'type' and 'unit' specifiers. If this specifier is omitted, the default unit number is 0 (zero). This specifier is useful only when there are two or more of the same storage devices attached to the node (i.e., type "w1" to salvage the second disk in a node with two Winchesters.)

Options

In response to the "Please input lv_num [-option]" prompt, you may specify the number (beginning with 1) of the logical volume you wish to salvage, and one or more of the following three options. (If you omit the logical volume number, logical volume 1 is salvaged.)

-A Direct the salvager to read every block of every file on the disk. This option is useful because AEGIS cannot always provide detailed descriptive information when an uncorrectable disk read or write error occurs online. SALVOL can diagnose these problems, but does not usually correct their cause.

Without the -A option, SALVOL reads only the minimum number of blocks necessary to ascertain the proper allocation of blocks to files; specifically, it reads the volume-table-of-contents (VTOC), block availability table (BAT), volume label, and file index blocks.

-V Instruct the salvager to examine the disk for possible inconsistency, but not to correct any damage it finds; do not write anything to the disk. This option allows you to inspect the disk without changing its current state. Select this option only if you believe that salvaging the volume might cause further damage or destroy "evidence" needed to debug or trace a system problem.

-U Identify problems by UID. Using this option presumes detailed knowledge of the DOMAIN disk structure. It causes the salvager to identify problems not just by filename, but also by UID and VTOC index (VTOCX). The -U option is useful only if you intend to try manually repairing disk problems that SALVOL does not attempt. You should rarely need this option.

Normally, SALVOL collects free blocks, displays a brief summary of free space on the volume, and terminates. If the disk was not properly dismounted, SALVOL may find errors in the block availability table (BAT). Even with the -A option, execution should take at most a few minutes to run on a Winchester disk. The salvager always displays extensive descriptive information about problems, and repairs the volume as necessary, using conservative strategies.

If any file has I/O errors, or if block allocation for a file is detectably wrong (for example, a block is destroyed because some other file wrote over it), SALVOL sets a "file trouble" flag in the file's VTOC entry. Other DOMAIN software (such as the STREAM_$OPEN system routine) checks this flag upon processing the file. If the flag is set, a warning or fatal error status may be returned.

9.3. Salvaging Strategy

The next two sections describe the salvager's processing strategy, the repairs it can make, and the meanings of the more verbose error messages. Refer to these sections whenever the salvager produces unexpected error messages.

Salvager Message Output

When the salvager processes a disk volume on which the logical structure is intact, it collects free blocks, prints a free block count, sometimes finds that the BAT is bad, and terminates. Just before termination, it confirms that the volume contains no multiply-allocated blocks.

Any other messages -- in particular, messages identifying multiply-defined blocks -- deserve careful attention. The salvager detects and describes four common classes of errors:

1. Out-of-range pointers - Any block addresses that point outside the logical volume. These are rare, and are usually the result of hardware or operating system errors. When found, the block pointers are set to zero, leaving a "hole" in the file. In addition, if the pointers are file pointers, the file trouble flag is set.

2. I/O errors - Any read or write attempts that fail at the disk driver level. Most write errors are fatal to the salvager, because it writes only critical blocks that contain space allocation information. For read errors in files, the salvager sets the trouble flag but takes no corrective action, on the assumption that other tools may be able to recover data.

3. Block header validation errors - These are "soft" I/O errors which, like disk driver errors, are fatal on critical system blocks. In normal files, these errors are simply detected and described, and a file trouble flag is set. Block header validation errors are usually the result of hardware or software failure.

4. Multiply-allocated blocks - These blocks are allocated to more than one file, or to a file and to a system structure, such as the VTOC, the BAT, or the badspot list. Multiply-allocated blocks may occur if you ignore the AEGIS warning to run the salvager. Less often, they result from AEGIS errors.

The salvager attempts to repair multiply-allocated blocks, and may display a summary of blocks that have been moved on the volume. In addition, SALVOL displays a list of all files for which it set the trouble flag. Note that the trouble flag is set only if the salvager is sure that a file has problems (an out-of-range pointer that was zeroed, an I/O header error, or a multiply-allocated block that was removed from the file map). For example, if the salvager finds a multiply-allocated block that is owned by two files, and can determine which file the block belongs to, then it sets the trouble flag only for the non-owning file.

Finally, the salvager may report on some uncommon conditions, including:

table overflow — This indicates that the logical structure on the volume is severely damaged. Mount the volume under AEGIS (but not as the boot volume), and move files to other volumes. If the error persists you may need to reinitialize this volume.

disk full — This error prevents the salvager from repairing the disk. Mount the volume under AEGIS (but not as the boot volume) long enough to delete one or more files, then dismount and salvage it.

internal errors — These include nonzero multiply-allocated block count, bad linked lists, etc. Most are fatal to the salvager, and may mean that the volume itself is in trouble.

A number of detailed errors are associated with system blocks, including:

no boot file — Every logical volume from which AEGIS is started must have a boot file. Hence, this error can be fixed only by reinitializing the volume. If this logical volume is not used to start the operating system, ignore the message.

trouble with badspot list
: Usually, the same block address appears twice in the badspot list; just ignore it. It could mean that the badspot list points to a block in the BAT. Ignore the message unless a BAT block really is bad (signified by repeated I/O errors), in which case the volume must be reinitialized.

vtoce with incorrectly hashed uid found
: The VTOCE (VTOC entry) is unusable, so respond "Y" to the salvager's "delete it?" prompt, unless you want to save the VTOCE to trace a probable AEGIS error.

too many bad spots
: Table overflow occurred in salvager. This is okay unless multiply-allocated blocks are found on the volume, in which case repair logic may fail.

label check error — Apparently, some other process is writing to the volume while the salvager is running; this error should never happen.

badspot list in lv label is full
: System limitation exceeded. Either the badspot list has been misused or the physical disk must be replaced.

vtoc chain pointer zeroed
: Some files may have been lost because VTOC chains are used to handle overflow of files hashed (by UID) to a full block.

Salvager Processing Strategy

This section describes some subtleties of salvager processing. This information is not needed for typical use of the salvager. In extreme cases, however, the details provided below may be helpful.

Two high-level observations first: DOMAIN disk volumes are structured so that naming directories and space/location information (in a VTOC) about files are kept separately. Currently, the salvager does not synchronize these on-disk structures. That is, it understands the UID file system and VTOC, but not naming directories. In particular, it cannot detect orphans, that is, files in the VTOC that have no names. Second, disk blocks on floppy disks are not self-identifying, as they are on the Winchester. Hence, the salvager's diagnosis and repair logic is much more limited with floppy disks than with Winchester disks.

The salvager is a 2-pass processor, but the second pass is executed only if multiply-allocated blocks are detected. During the second pass, the salvager completes the list of owners of each multiply-allocated block, if, as in most cases, this cannot be done in the first pass. Both passes consist of a sequential VTOC scan, including descent of any VTOC block chains which result from (UID) hash overflow. For each VTOC entry that points to a permanent file, SALVOL reads the file map or list of block addresses sequentially to construct a new block availability table.

SALVOL reads and verifies the file blocks themselves only if you choose the -A option, or if problems must be resolved. Pass two terminates when the owners of all multiply-allocated blocks are identified, and thus may not be a complete sequential pass. After each pass, SALVOL repairs the volume if multiply-allocated blocks were found, and writes out the updated BAT and logical volume label.

I/O errors that occur on physical and logical volume labels or on the block availability table (BAT) for a logical volume are fatal to the salvager. All other errors are reported, but are non-fatal.

The salvager corrects I/O errors in files only in the following special case. AEGIS may report an I/O error in a file if the block header is incorrect. If the block is multiply-allocated, which could cause such an error report, the salvager allocates it exclusively to the "real" owner, and zeros the pointer(s) in any other file(s), leaving holes but eliminating the I/O errors. On a floppy, because the owner cannot be determined, SALVOL copies the block into all "owner" files, and sets the trouble flag. After the salvager terminates, you can inspect the data to determine which files are really intact.

SALVOL zeros bad block pointers. This loses data, and even whole files when the bad pointers are in the VTOC itself, but currently the salvager cannot locate the correct data even if it is on the disk.

Generally, the salvager always repairs the BAT (except in case of hard I/O errors) and the VTOC. Thus, if AEGIS badly malfunctions, writing normal file blocks over the BAT or VTOC blocks, for example, the salvager repairs the BAT or VTOC and file. To do so, it copies the data into a newly allocated block and reinitializes the overwritten block.

If a block is multiply-allocated to both the badspot list and to a file or a VTOC chain, the salvager tries to copy any potentially valid data to a newly allocated block. If the block is in the badspot list because of persistent device level errors, however, the copy may fail; the salvager then prompts for alternatives. The salvager and badspot listing cannot be used to correct persistent errors in the BAT or VTOC hash space, however. The salvager aborts in the former case, and simply reports the I/O error in the second case. The only solution is to reinitialize the volume around such badspots using INVOL.

Finally, note that table overflow in the salvager is most serious. You can try to rerun the salvager several times, if some repair is apparent on each run, and if the salvager attempts to complete in each case without doing additional damage. However, in some cases, data may be unnecessarily lost. You should never mount the volume under AEGIS while this condition persists, except to copy/dump and delete files, or to reinitialize the volume.

9.4. Limitations

- SALVOL can process only one logical volume at a time.
- SALVOL can correctly process at most 64 multiply-allocated blocks and/or header validation errors on a logical volume.
- Because floppy disks do not have block headers, so that blocks are not self-identifying, the salvager is much less effective in repairing floppy disk problems than it is in repairing Winchester disk or storage module problems.

Index

READER'S RESPONSE

We use readers' comments in revising and improving our documents.

Document Title: *DOMAIN System Utilities*
Order No.: 009414
Revision: 00
Date of Publication: September, 1986

What is the best feature of this manual?

Please list any errors, omissions, or problem areas in the manual. (Identify errors by page, section, figure, or table number wherever possible.)

What type of user are you?

How often do you use your system?

Nature of your work on the DOMAIN System:

Your name Date

Organization

Street Address

City State Zip/Country

No postage necessary if mailed in the U.S. Fold on dotted lines (see reverse), tape, and mail.

cut or fold along dotted line

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 78 CHELMSFORD, MA 01824

POSTAGE WILL BE PAID BY ADDRESSEE

APOLLO COMPUTER INC.
Technical Publications
P.O. Box 451
Chelmsford, MA 01824

FOLD

READER'S RESPONSE

We use readers' comments in revising and improving our documents.

Document Title: *DOMAIN System Utilities*
Order No.: 009414
Revision: 00
Date of Publication: September, 1986

What is the best feature of this manual?

Please list any errors, omissions, or problem areas in the manual. (Identify errors by page, section, figure, or table number wherever possible.)

What type of user are you?

How often do you use your system?

Nature of your work on the DOMAIN System:

Your name Date

Organization

Street Address

City State Zip/Country

No postage necessary if mailed in the U.S. Fold on dotted lines (see reverse), tape, and mail.

cut or fold along dotted line

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 78 CHELMSFORD, MA 01824

POSTAGE WILL BE PAID BY ADDRESSEE

APOLLO COMPUTER INC.
Technical Publications
P.O. Box 451
Chelmsford, MA 01824

FOLD