Part No. ~~061-3200-00~~ 070-5925-00
Product Group 07

NEW #

4400 SERIES OPERATING SYSTEM REFERENCE

35.00	070-5733-00	4400 OPT. 10 LAN INTERFACE
35.00	070-5603-01	4404 USERS
150.00	070-~~5604~~ 5925-00	4404 REFERENCE
———	070-5609-02	4404 FIELD SERVICE
20.00	070-5612-00	4404F20 40 Mbyte DRIVE W/ STREAMING TAPE
NOT AVAILABLE	070-5607-00	4404P30 LISP PROGRAMMING
20.00	070-5606-00	4404 INTRO TO SMALLTALK-80
NOT AVAILABLE	070-6755-00	SMALLTALK REFERENCE
70.00	070-6454-00	SMALLTALK USERS

Please Check for CHANGE INFORMATION at the Rear of This Manual

'st Printing MAR 1986

Tektronix
COMMITTED TO EXCELLENCE

WARRANTY FOR SOFTWARE PRODUCTS

Tektronix warrants that this software product will conform to the specifications set forth herein, when used properly in the specified operating environment, for a period of three (3) months from the date of shipment, or if the program is installed by Tektronix, for a period of three (3) months from the date of installation. If this software product does not conform as warranted, Tektronix will provide the remedial services specified below. Tektronix does not warrant that the functions contained in this software product will meet Customer's requirements or that operation of this software product will be uninterrupted or error-free or that all errors will be corrected.

In order to obtain service under this warranty, Customer must notify Tektronix of the defect before the expiration of the warranty period and make suitable arrangements for such service in accordance with the instructions received from Tektronix. If Tektronix is unable, within a reasonable time after receipt of such notice, to provide the remedial services specified below, Customer may terminate the license for the software product and return this software product and any associated materials to Tektronix for credit or refund.

This warranty shall not apply to any software product that has been modified or altered by Customer. Tektronix shall not be obligated to furnish service under this warranty with respect to any software product a) that is used in an operating environment other than that specified or in a manner inconsistent with the Users Manual and documentation or b) when the software product has been integrated with other software if the result of such integration increases the time or difficulty of analyzing or servicing the software product or the problems ascribed to the software product.

TEKTRONIX DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. TEKTRONIX' RESPONSIBILITY TO PROVIDE REMEDIAL SERVICE WHEN SPECIFIED, REPLACE DEFECTIVE MEDIA OR REFUND CUSTOMER'S PAYMENT IS THE SOLE AND EXCLUSIVE REMEDY PROVIDED TO CUSTOMER FOR BREACH OF THIS WARRANTY. TEKTRONIX WILL NOT BE LIABLE FOR ANY INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES IRRESPECTIVE OF WHETHER TEKTRONIX HAS ADVANCE NOTICE OF THE POSSIBILITY OF SUCH DAMAGES.

PLEASE FORWARD ALL MAIL TO:

Artificial Intelligence Machines
Tektronix, Inc.
P.O. Box 1000 M.S. 60-405
Wilsonville, Oregon 97070
Attn: AIM Documentation

MANUAL REVISION STATUS

PRODUCT: **4400 SERIES OPERATING SYSTEM REFERENCE**

This manual supports the following versions of this product: **4404 Version 1.5, 4405 Version 1.1, and 4406 Version 1.1 .**

REV DATE	DESCRIPTION
MAR 1986	Original Issue

Table Of Contents

Figures

Examples

Tables

Section 1
INTRODUCTION

ABOUT THIS MANUAL

This manual is the primary user's and programmer's reference to the 4400 operating system and hardware support. This manual contains summaries of the commands and utilities included with your 4400 as standard software, and a summary of how to invoke and use each command. This manual does not attempt to show you how to put commands together to perform a task; that information is covered in the *4400 User's Manual.* The User's Manual also contains a complete list of the other manuals available for the 4400 series.

This manual has the following sections:

- User Commands
- System Utilities
- Text Editor
- Terminal Emulation

In addition, the appendices contain information about the hardware of the 4400 series of products.

WHERE TO FIND INFORMATION

You have several important sources of information on the 4400:

- This manual, the *4400 Series Operating System Reference* manual, contains the syntax and details of commands and utilities. This manual also contains details about a text editor and a remote terminal emulator.
- The *4400 Series Assembly Language Reference* manual contains the details of the assembler and linking loader.
- The *4400 Series C Language Reference* manual contains detail about the "C" programming language.
- The *4400 Users* manual contains basic information on system installation, startup, installing software, and the other "how to put commands together" discussions. See the index of the *User's* manual to find how to perform particular tasks.
- The on-line "help" utility contains a brief description of the syntax of user commands.
- The *Introduction to Smalltalk-80(tm)* manual contains details and a short tutorial on the Smalltalk-80 programming language.
- The reference manuals for the optional languages for the 4400 product family are also available.

MANUAL SYNTAX CONVENTIONS

Throughout this manual, the *4400 User's Manual*, and in the on-line help files, the following syntax conventions apply:

1. Words standing alone on the command line are *keywords*. They are the words recognized by the system and should be typed exactly as shown.
2. Words enclosed by angle brackets ("<" and ">") enclose descriptions of variables that are replaced with a specific argument. If an expression is enclosed *only* in angle brackets, it is an essential part of the command line. For example, in the line:

   ```
   addusr <user_name>
   ```

   you must specify the name of the user in place of the expression <user_name>.
3. Words or expressions surrounded by square brackets ("[" and "]") are optional. You may omit these words or expressions if you wish.
4. If the word "list" appears as part of a term, that term consists of one or more elements of the type described in the term, separated by spaces. For example:

   ```
   <file_name_list>
   ```

   consists of a series (one or more) of file names separated by spaces.

Invoke and use each command. This manual does not attempt to show you how to put commands together to perform a task; that information is covered in the *4400 User's Manual.*

Section 2
USER COMMANDS

You can use the commands and utilities in this section from any user account. Some options, however, require special privileges. These options are mentioned in the detailed description of each command or utility.

addpath

Add the specified directories to the search path of the shell. This is a shell command.

SYNTAX

```
addpath <dir_name_list>
```

DESCRIPTION

This *addpath* command, which is part of the shell program, adds the specified directories to the search path of the shell. This is done by altering the shell environment variable *PATH*.

ARGUMENTS

<dir_name_list> list of directory names to add to the search path.

EXAMPLE

```
addpath /etc
```

This example adds the directory */etc* to the shell search path, by adding the directory to the environment variable *PATH*.

SEE ALSO

rmpath
set
shell
unset

alias

Defines or reports the list of alternate names (aliases) for a command sequence.

SYNTAX

```
alias [<alias_name>] [<string>]
```

DESCRIPTION

The *alias* command, which is part of the shell program, defines or reports the list of alternate names (aliases) for a command sequence. With no arguments *alias* outputs the list of aliases defined. If one argument is given the associated alias is printed. If two arguments are given then the first is defined to be an alias for the second. Command line arguments are extracted via the *shell* conventions.

ARGUMENTS

<alias_name>	name of the alias.
<string>	may consist of combinations of utility commands and environment variables surrounded by either single or double quotes (i.e. "copy $*").

EXAMPLES

```
alias long 'dir +l $* | page +30'
```

This example will create an alias *long* that will invoke the command *dir +l*, and pause every 30 lines until you press the space bar.

```
alias
```

This example will display the currently defined aliases.

SEE ALSO

shell
unalias

asm

The *asm* command is the MC68000/68010 relocating assembler.

SYNTAX

```
asm <file_name_list> [+befFlLnsStu] [+o=<file_name>]
```

DESCRIPTION

The *asm* command is used to assemble a program written in the standard 68000 instructions set. The assembler accepts most of the standard mnemonics for instructions, and fully supports the 68000/68010/68020 instruction set. For more information, refer to *4400 Series Assembly Language Reference.*

ARGUMENTS

<file_name_list>	List of the names of files and directories to process. Default is the working directory.

OPTIONS

b	Suppress binary output.
e	Suppress summary information.
f	Disable field formatting.
F	Enable *fix* mode. (Comments that begin with a semicolon, ";", are assembled.)
l	Produce a listing of the assembled source.
L	Produce listing of input file during the first pass.
n	Produce decimal line numbers with the listing.
o=<file_name>	Specifies the name of the binary file.
s	Produce a listing of the symbol table.
S	Limit symbols internally to 8 characters.
u	Classify all unresolved symbols as external.

EXAMPLES

```
asm asmfile
```

Assembles the source file *asmfile* and produces the relocatable binary file *asmfile.r*. The assembler sends summary information to standard output, but produces no source listing. Any errors detected are sent to standard output.

```
asm test.a +euo=test.r
```

Assembles the file *test.a* and produces the relocatable file *test.r*. No summary information is produced, and all unresolved references are classified as external. If the assembler detects no errors during the assembly, the user sees no output from this command.

```
asm test.a test2.a test3.a +blns
```

Assembles the three files, but produces no binary output. A listing with a symbol table is sent to standard output. The listing includes decimal line numbers.

SEE ALSO

4400 Series Assembler Language Programmer's Reference

backup

Copy files from the file system to the floppy device or streaming tape device.

SYNTAX

```
backup [+AbBCdlpr] [+a=days] [+t[=file_name]] [+T[=<length>]]
       [<file_name_list>] [<dir_name_list>]
```

DESCRIPTION

The *backup* command is used to create and maintain archival backups of files or directories on the system. It can operate in three distinct modes, selected by options: catalog mode, create mode, and append mode. Catalog mode prints a list of the files on an existing backup. Create mode copies the specified files or directories to the backup device, and destroys any data that is already on the backup device. Append mode adds the specified files or directories to existing files on the the backup device. Thus, it is possible to append, to an existing backup device, a file whose path name is identical with one already backed up.

The *backup* command stores files and directories on the diskette (*/dev/floppy*) by default or on the optional streaming tape drive (*/dev/tapec*). The *backup* command uses a unique file structure, which is completely different from the standard operating system file structure. Therefore, */dev/floppy* or */dev/tapec* must not be mounted onto the file system using the *mount* command. The only way to read devices written by *backup* is to use *restore*. The only other command that you should use on a backup device is *devcheck*.

The backup diskette should be formatted before the back up operation begins. Although the file structure created by the format command is destroyed by *backup*, the raw track formatting is essential. During the back up process, you can request that *backup* formats diskettes before writing to them. Do this by pressing *f* followed by *Return*, rather than *Return* when backup prompts you to *Hit C/R to continue:*.

The backup tape may not be formatted, but the retensioning option "r" may be specified to avoid reading-errors.

Backups may extend over more than one volume of the backup medium. There are no restrictions on the sizes of files copied. If necessary, *backup* breaks files into segments and stores each segment on a different volume.

As files are backed up, *backup* also stores the file owner ID number, permissions, and time/date stamp of the file. This is used by *restore* when retrieving the files. After the files are restored, they appear just as they were at the time of the backup. The user should be aware of several potential problems.

First, it is possible for users with identical ID numbers to exist on different systems with different names. Since only the owner ID number is saved with the file, not the owner's name, when the file is restored, the apparent owner will be the name of the user in the password file that matches the ID number. If the user ID number does not exist in the restoring system password file, the owner of the file will be the ID number enclosed in double angle brackets, for example, <<14>>. Second, file permissions are respected during restore. If the restoring user does not have write permission for a file, it will not be restored. One method to facilitate easy movement of files

among many machines is to always backup and restore the files from the public user, which exists on all machines. In any event, the user system can backup and restore any file as well as change ownership and permissions.

ARGUMENTS

<file_name_list>	List of the names of files to process. Default is the working directory.
<dir_name_list>	List of the names of directories to process.

If you specify a directory name as an argument in create or append mode, the program processes only the files within that directory. If you also specify the "d" option, the program restores all files within the given directory and its subdirectories.

OPTIONS

a=<days>	Copy only those files that are no older than the specified number of days. A value of 0 specifies files created since midnight on the current day; a value of 1 specifies files created since midnight of the previous day, and so forth.
A	Append to an existing backup.
b	Print sizes of files in bytes.
B	Do not back up files that end in ".bak".
C	Print a catalog of the files on an existing backup. If you specify "C", all of the names in the <file_name_list> are ignored.
d	Back up entire directory structures.
e	Erase entire streaming tape before any action.
l	List file names as they are copied.
p	Prompt user with each file name to determine whether or not the backup procedure should be performed on that particular file.
r	Retension streaming tape cartridge before any action. Using this option may avoid reading errors from the streaming tape drive. This option must be used in conjunction with the *+T* option.
t[=<file_name>]	Back up only files that have been created or modified since the date in the specified file. When the backup is finished, update the date in the file. If you do not specify a file, the default is *.backup.time*.
T[=length]	Backup to the streaming tape instead of the floppy. The default parameter for the tape length is 450 feet. To backup to a 300 foot tape, use *+T=300*.

With no options, *backup* is quiet. The "l" option allows you to see what the program is actually doing.

If you specify the "t" option, but the *.backup.time* file specified as its argument does not yet exist, *backup* copies all the files and directories listed on the command line. Thus, a user may obtain a full backup (either without the "t" option or with a nonexistent *backup time* file) or a partial

backup, which includes only those files created or modified since the last backup.

EXAMPLES

```
backup +l
```

Backs up all files in the working directory to the device */dev/floppy*. The file names are listed as they are copied to the device.

```
backup +ld file1 file2 dir1 dir2
```

Copies (in order) the files *file1* and *file2*, then all files and sub-directories contained in the directories *dir1* and *dir2*, listing the file names as they are copied.

```
backup +ld file1 file2 dir1 dir2 +a=5
```

Performs the same function as the previous example, except it copies only those files that are five days old or less.

```
backup +lt
```

Creates the same backup as the first example, but only copies the files created or modified after the time contained in the file *.backup.time*. If this file does not exist, all the files are copied and the file *.backup.time* is created.

```
backup +lAt=backup_time
```

Adds a set of files to an existing backup. In particular, it adds exactly the files that were created or modified since the creation of the file *backup_time*. This is the most direct way to create incremental backups of your files. The length of time between backups should reflect the amount of activity you spend developing programs, etc.

```
backup +lT
```

Backs up all files in the working directory to the device */dev/tapec*. The file names are listed as they are copied to the device.

NOTES

- When using append mode, the program appends files to the last volume, requesting additional volumes as necessary. If there are many volumes in an existing set of diskettes, place the last volume (diskette) in the backup device. In this case a message is issued indicating the voulme is not the first and prompts for permission to continue. Respond with a "y" and a *C/R* to the prompt. The program then appends files to that volume, requesting new volumes as necessary.
- As files are backed up, *backup* makes an indication of the path name for each file. When files are restored, the program uses the path name to place the file in its proper directory location. If the path name is relative (i.e., does not begin with "/"), the path name of the restored directory is also relative. Thus, files backed up with a relative path name may be restored to a directory location different from the one in which they were created.

  An example should make this clear. If the working directory is backed up, either by specifying no source files or by using the directory name ".", the files are backed up with a relative path of ".". When these files are restored, they are placed in the directory ".". This directory might not be the same directory they originally came from. This feature allows the manipulation of entire file systems in a general fashion. To specify a unique directory location for a file, you should specify its entire path name, starting with "/".

MESSAGES

```
Backup to <file_name>
Update backup on <file_name>
```

These messages are printed when *backup* begins. They notify you of the function about to be performed.

Several of the following messages prompt you for a positive or negative response. The program interprets any response that does not begin with an upper or lowercase "n" as a positive response.

```
Copy <file_name> (y/n)?
```

If you specify the "p" option, the program prints this prompt before it takes any action. A response of "n" or "N" indicates that the operation should not be performed for the given file. Any other response is interpreted as *yes*.

```
Device model name?
```

You should respond to this prompt with *TEK4404*.

```
Do you wish to abort append function and create a new backup?
```

This message is printed at the initiation of the *append* operating mode if an invalid header (indicating a bad backup format) is detected. You can now abort from *append* mode and switch to *create* mode.

```
Format program name?
```

This prompt is issued in response to a *format* request for the next diskette volume. It indicates that the program could not find a format program name in the file */etc/format.control*. You should respond with *format* since you are backing up on a diskette. You can not format a streaming tape cartridge.

```
Insert next volume - Hit C/R to continue:
```

This prompt is issued when the program needs a new backup diskette or tape cartridge. You should type a carriage return only when the next device has been placed in the drive. When creating new backups or when appending to an old backup, with diskettes, you may enter the character "f", followed by a carriage return. If the program is in append mode, it automatically switches to create mode and starts a new backup. The "f" indicates that the diskette has been inserted in the drive, but that it must be formatted before continuing. In this case the program first checks the file */etc/format.control* for a format program name, and if found formats the diskette. If it cannot find this file, it then prompts you for the format program necessary to format the diskette. Subsequent format operations during this backup operation use the same information; thus, all diskettes that were not previously formatted must have the same characteristics (e.g. double-sided, double-density).

The program prints these messages as it takes the corresponding action during a creation operation.

```
This is Volume #<number_1> -- Expected Volume #<number_2> -- Continu
```

The program expects you to insert volumes in sequential order. If a volume appears out of order, *backup* prints this message. If you type anything except an "n" or an "N" as the first character of the response to the message, *backup* ignores the fact that the volumes are out of order and continues with the backup. Otherwise, it prompts you for another volume.

```
Volume name?
```

Each set of backup volumes has a name. You should enter a name that describes the contents, in response to this prompt. The name may contain as many as 126 characters.

```
Volume <number> of <vol_name>
```

When you are printing a catalog, whenever a new volume is inserted and properly validated the program prints this message, which indicates the name of the backup volume and its sequence

number.

ERROR MESSAGES

```
*** Invalid Volume Header -- Not a "backup" disk ***
```

The program validates each backup volume before using it. If this validation fails, the program prints this message to indicate that something is wrong. You then have another chance to insert the proper volume and continue. If validation fails while the program is in append mode, you may abort from append mode and create a totally new backup instead.

```
Write error! - file <file_name>
```

An I/O error occurred during the transfer of a file to the backup. An auxiliary message is printed indicating the nature of the error. The program tries to recover from any error and continue.

```
backup: unknown options: '+<char>'
```

The option specified by <char> is not a valid option to the *backup* command.

```
** Warning: directory <dir_name> is too large!
** Some directories were ignored
** Warning: directory <dir_name> is too large!
** Some files were ignored
```

The program uses some internal tables during the back up process. If the limits of these tables are exceeded (highly unlikely), these messages are printed.

SEE ALSO

format
restore

CC

Invoke the "C" compiler.

SYNTAX

```
cc <file_name_list> [+acDfiIlLmMnNoOpPqrRsStUvwx] [+i=<dir_name>]
[+l=<lib_name>] [+o=<file_name>]
```

ARGUMENTS

<file_name_list>	List of the names of files and directories to process. Default is the working directory.

OPTIONS

a	Produce as output assembly language source files with an *.a* extension and stop.
c	Put comments in the assembly language file.
D<name>[=<defn>]	Command line "#define". This option must appear by itself.
f	Produce an output module suitable for firmware.
i=<dir_name>	Specify a directory for "#include" files. This option must appear by itself.
I	Produce as output intermediate language files with an ".i" extension and stop.
l=<lib_name>	Specify a library name to be passed to the loader. This option must appear by itself.
L	Produce a source listing and write it to standard output.
m	Produce load and module maps from the loader.
M	Leave the combined output as one ".r" file.
n	Run the first pass only, do not produce any output.
N	Produce a listing without expanding *#include* files.
o=<file_name>	Specify the output file name.
O	Run the assembly language optimizer.
p	Use stand alone pre-processor.
P	Produce intermediate (.p) files and stop.
q	Produce code that does calculations on *char* and *short* variables without first converting to *int*.

r	Produce as output relocatable modules with an ".r" extension and stop.
R	Produce as output relocatable modules with an ".r" extension, and continues to produces an executable module.
t	Produce a shared-text, executable output module.
U	Produce a line-feed character ($0A) for "\n" rather than the default of carriage return ($0D).
v	Show each phase of the compilation process (verbose mode).
w	Warn about duplicate "#define" statements.
x=<ldr_option>	Pass the options to the loader for processing.

For a full discussion of the "C" compiler, refer to the manual *4400 Series C Language Programmer's Reference.*

NOTE

The "C" stand-alone pre-processor is the file /bin/cpasses/cprep. *If you want to use it with another program, it takes its input from stdin and writes its output to stdout.*

EXAMPLES

```
cc blocks.c +O +l=graphics
```

Compiles the program *blocks.c*, requesting the assembly language optimizer and passing the library *graphics* to the loader.

```
cc labels.c +vLNr
```

Compiles the program *lablels.c* in verbose mode. The compiler produces a source listing, without expanding any *"#include"* files, creating only a relocatable module *labels.r.*

```
cc access.c labels.r +o=access
```

Compiles the source program *access* and the relocatable module *labels.r* producing a single binary output file *access.*

```
cc rand.c +i=/mark/include +DTHROWS=300 +t +o=dice
```

Compiles the program *rand.c*, specifing a directory */mark/include* for *#include* files and specifying a command line define of *THROWS* to equal *300.* A shared-text binary output file *dice* is produced.

SEE ALSO

headset
load

chd

Change the user's working directory.

SYNTAX

```
chd [<dir_name>]
```

DESCRIPTION

The *chd* command, which is part of both the shell and script programs, changes the user's working directory to the directory specified on the command line. If no directory is specified, the default is the user's home directory (the directory entered on logging in). The user must have execute permission in the directory specified.

ARGUMENTS

<dir_name>	The name of the directory to use as the working directory. Default is the user's home directory.

EXAMPLES

```
chd /mark
```

Changes the working directory to the directory */mark*.

```
chd book
```

Changes the working directory to the directory *book*, which resides in the current working directory.

```
chd
```

Changes the working directory to the user's home directory.

ERROR MESSAGES

```
Cannot change directories.
```

The operating system returned an error when the script program tried to change directories. This message is preceded by an interpretation of the error produced by the operating system.

SEE ALSO

shell
script
perms

commset

Set or display the configuration of the communications port.

SYNTAX

```
commset [<options_list>]
```

DESCRIPTION

This utility allows you to examine or set certain I/O options on the RS-232 communications port. With no argument, it reports the current setting of the options.

OPTIONS

The option strings are selected from the following set:

baud=nnn

=external
=nnn/mmm
=default

Set the transmit and receive baud rates. Valid values are 50, 75, 110, 134, 150, 300, 600, 1200, 1800, 4800, 9600, 19200 and 38400. The keyword *external* specifies that the external clock should be used for the baud rate. The default of 9600 is used if the keyword *default* is entered. If two values are entered, then the first specifies the transmit rate and the second specifies the receive rate, otherwise both rates are set to the same value.

flag=dtr

=input
=output
=inout
=none
=default

Set the type of flagging to be used. The keyword *dtr* specifies that the DTR and CTS signals should be used to flag input and output full conditions. The keywords *input* and *output* specify that DC3/DC1 (CTL-S/CTL-Q) flagging should be used for input or output, respectively. The keyword *inout* specifies that DC3/DC1 (CTL-S/CTL-Q) flagging should be used for both input and output. The keyword *none* disables flagging. The default is inout flagging.

parity=even

=odd
=high
=low
=none
=default

Set the type of parity to be used. The keyword *even* specifies that even parity be used. The keyword *odd* specifies that odd parity be used. The keyword *high* specifies that the parity bit should always be a one. The keyword *low* specifies that the parity bit should always be a zero. The keyword *none* specifies that the parity bit is treated as data. The default is low parity.

stop=n

=default

Set the number of stop bits to be used. Valid values are 1 and 2. The default is one stop bit.

CTS=disable

=enable

Select whether to use the Clear-to-Send (CTS) data signal for communication protocol. Disabling CTS means to ignore the signal condition. With CTS enabled, a CTS signal must be received before transmission is enabled.

reset

Reset the communications port, flushing any pending data and setting all options to their default values.

show

Display the current settings for the options. This is the same as if no option is specified.

C IMPLEMENTATION NOTES

The *commset* command uses the *ttyset* and *ttyget* system calls to communicate option settings to the communications port device driver.

SEE ALSO

conset
Assembler Language Reference manual

compare

Compare two text files line by line and prints the differences.

SYNTAX

```
compare <file_name_1> <file_name_2> [+<window_size>]
```

DESCRIPTION

The *compare* command compares two text files and indicates how they differ. The information provided is usually sufficient to allow the user to change one file into the other. By default, the *compare* command considers that it is in the same place in each of the files if three lines match.

The output from the command reports sets of lines which have been deleted from, added to, or changed in either file. These messages are written from the point of view of how to change the first file into the second file. For instance, the message

```
***** File <file_name_1> lines deleted *****
```

means that if the lines following the message are deleted from *<file_name_1>*, the two files will be the same.

The program also reports the presence of additional lines in a file with the following message:

```
***** File <file_name_1> lines inserted *****
```

This message means that if the lines following the message are inserted to <file_name_1>, the two files will be the same.

If a set of lines is deleted from one file and the following line is changed as well, *compare* reports all those lines as lines that have been changed rather than inserted or deleted.

The *compare* command can handle files of any size, but can only process 250 lines at a time. If the files differ in any spot by 250 lines, the program reports 250 lines changed in each file and continues comparing them.

ARGUMENTS

<file_name_1>	The name of the first file to use.
<file_name_2>	The name of the file to compare to <file_name_1>

OPTIONS

<window_size>	Use the integer <window_size> as the number of matching lines required before considering the files synchronized. The number specified must be between 1 and 250, with a default of 3.

EXAMPLES

```
compare /michael/test /cathy/test
```

Compares the file *test* in the directory */michael* to the file *test* in the directory */cathy*.

```
compare test test.bak +5
```

Compares the two files *test* and *test.bak* in the working directory. The window size for the comparison is five lines.

ERROR MESSAGES

```
Syntax: compare <file_name_1> <file_name_2> [+<window_size>]
```

The *compare* command expects two or three arguments. This message indicates that the argument count is wrong.

conset

Set or display the configuration of the console port.

SYNTAX

```
conset <options_list>
```

DESCRIPTION

The utility *conset* allows you to examine and set certain I/O options on the console port. With no argument, it reports the current setting of the options.

OPTIONS

The option strings are selected from the following set:

+raw -raw	Set or clear the raw mode.
+echo -echo	Enable or disable character echoing.
+tabs -tabs	Enable or disable automatic tab expansion.
+becho -becho	Enable or disable space/backspace to erase on backspace.
+schar -schar	Enable or disable single character mode.
+xon -xon	Enable or disable CTRL-S/CTRL-Q (DC3/DC1) flagging to suspend output.
+any -any	Enable or disable any character to restart suspended output.
+crnl -crnl	Enable or disable RETURNs, to be displayed as return/line-feed.
chardel=<n>	*n* is a hex number specifying a character to be used as the delete character.
linedel=<n>	*n* is a hex number specifying a character to be used as line delete character.
+screensave -screensave	Enable or disable screen blanking after 10 minutes of inactivity.

+video -video	Select normal video (black on white) or inverse video.
+cursor -cursor	Select make graphic cursor visible or invisible.
+track -track	Enable or disable graphic cursor tracking the mouse.
+mousepan -mousepan	Enable or disable mouse panning of the viewport.
+diskpan -diskpan	Enable or disable joydisk panning of viewport.
show	Display the current settings for the options. This is the same as if no option is specified.
default	Restore default settings.

C IMPLEMENTATION NOTES

The conset command uses the *ttyset* and *ttyget* system calls to communicate the raw, echo, tabs, becho, schar, xon, any, crnl, chardel and linedel option settings to the console port device driver and it uses system traps to implement the screensave, video, cursor, track, mousepan, and diskpan options.

SEE ALSO

commset

copy

Copy a file or directory to the specified file or directory, or copy one or more files to the specified directory.

SYNTAX

```
copy <file_name_1> <file_name_2> [+bBcdDlLnopPt]
copy <file_name_list> <dir_name_2> [+bBcdDFlLMnopPt]
copy <dir_name_1> <dir_name_2> [+bBcdDFlLMnopPt]
```

DESCRIPTION

Three forms of the *copy* command exist. The first form makes a copy of a file and gives it the specified name. The second form makes one copy of each specified file and places all copies in the specified directory. The last component of each file name is preserved in the new directory. The third form copies the contents of one directory to another.

In any case, if no file exists which has the same name as the name specified for the new copy, the *copy* command creates one. If a file with that name already exists, it is deleted and recreated before copying takes place. Thus, the original contents of the file is lost and replaced by the contents of the file being copied. In addition, any links to the original file are broken.

The new file has the same permissions as the original file. The owner of the new file is always the user who executes the command. The user must have execute permission in the directory in which copies are to be made. He or she must also have write permission for the file being copied to and, unless the "o" option is specified, in the directory that is to contain the new copy.

ARGUMENTS

<file_name_1>	The name of the file to copy.
<file_name_2>	The name of the new copy of the original file.
<file_name_list>	A list of the names of the files to copy to the specified directory.
<dir_name_1>	The name of the source directory.
<dir_name_2>	The name of the directory in which to place all copies.

OPTIONS

b	Do not copy a file unless it already exists in the destination directory.
B	Don't copy files ending in *.bak*.
c	Do not copy a file if it already exists in the destination directory. Cannot be used with *n*.
d	Copy directory structure for all named directories.
D	Implicity specify the high level directory names. This option works properly only in conjunction with the *+d* option. When used together with *+d*, *+D* preserves the source directory structure within the destination directory.
F	Copy/convert a directory to a regular file.
l	List the name of each file as it is copied and the name of the new copy.
L	Do not unlink the destination file.
M	Convert RETURN/new-line to LINE-FEED/new-line
n	Copy a file if it is newer than the copy in the destination directory. If no copy exists, perform the copy.
o	Retain original file ownership.
p	Prompt for permission to copy each file.
P	Preserve all the characteristics of the file - the modification time and the ownership of the source file.
t	Do not copy source directory unless destination directory exists.

EXAMPLES

```
copy parts parts.bak
```

Copies the file named *parts* to a file named *parts.bak*. If a file named *parts.bak* already exists, it is deleted and recreated before copying takes place.

```
copy letter /mark/letter +p
```

Copies the file *letter* in the working directory to the file */mark/letter*. The *copy* command prompts for permission to copy before proceeding. If the user denies permission, no copy is made. For the command to succeed the user must have both write and execute permission in the directory */mark* as well as write permission for the file */mark/letter*.

```
copy test_1 test_2 memo /mark +lo
```

Copies the files *test_1*, *test_2*, and *memo* to the directory */mark*. The names of the new files are */mark/test_1*, */mark/test_2*, and */mark/memo*. If a file with one of these names already exists, the *copy* command overwrites its contents without warning (the user does not need write permission in the directory */mark*). The name of each file and the name of the new copy are listed as copying takes place. The command aborts immediately if it encounters an error (e.g., one of the files listed does not exist).

Each copy created by these commands has the same permissions as the original file. The owner of all copied files is the user executing the command.

```
copy dir_1 /mark +dnolDLP
```

Copies the directory *dir_1*, and any sub-directories, to the directory */mark*. For source files in the destination directory a copy is made only if the source file is newer. the files are listed as they are copied; preserving ownership, links, and modification times. The source directory structure *dir_1* will be preserved exactly in the directory */mark*.

ERROR MESSAGES

```
Entry does not exist: <file_name>
```

The user asked for a copy of a nonexistent file.

```
<file_name_1> and <file_name_2> are the same file
```

A file may not be copied onto itself. Both *<file_name_1>* and *<file_name_2>* refer to the same file. (If their names are not the same, they are links to the same file.)

```
May not copy a directory: <dir_name>
```

The user asked for a copy of a directory. Directories may not be copied.

```
May not copy a special file: <file_name>
```

The user asked for a copy of a block or character file. Such files may not be copied.

```
Must be a directory: <file_name>
```

The form of the *copy* command being used requires the last argument to be an existing directory; <file_name> is not an existing directory.

```
Path cannot be followed: <file_name>
```

One or more of the directories which make up the name of the file do not exist.

```
Permissions deny access to file: <file_name>
```

The permissions associated with *<file_name>* or with the path leading to *<file_name>* prevent the user from accessing the file.

```
Read error on file: <file_name>
```

A physical read error occurred while reading *<file_name>*.

```
Syntax: copy <file_name_1> <file_name_2> [+bBcdDlLnopPt]
        copy <file_name_list> <dir_name> [+bBcdDFlLMnopPt]
```

The *copy* command expects at least two arguments. This message indicates that the argument count is wrong.

```
Write error on file: <file_name>
```

A physical write error occurred while writing to *<file_name>*.

SEE ALSO

link
move
rename

crdir

Create a directory.

SYNTAX

```
crdir <dir_name_list>
```

DESCRIPTION

The *crdir* command creates a directory for each name listed as an argument to the command. The user must have write permission in the directory in which the new directory is created. Each new directory contains the entry ".", which represents the directory itself, and the entry "..", which represents its parent directory.

By default, *crdir* creates a directory with *rwxrwx* permissions. However, any default permissions set by the *dperm* command override these permissions. The owner may, of course, change the permissions at any time by using the *perms* command.

ARGUMENTS

<dir_name_list>	A list of the names of directories to create. All of the components of the directory name (path name), except the last component, must already exist.

EXAMPLES

```
crdir book
```

Creates the directory *book* in the working directory.

```
crdir /sarah/book
```

Creates the directory *book* in the directory */sarah*. If the directory */sarah* does not already exist, the command fails.

ERROR MESSAGES

```
Error creating <dir_name>: <reason>
```

The operating system returned an error when *crdir* tried to create the specified directory. This message is followed by an interpretation of the error returned by the operating system.

```
Error linking <dir_name> to its . file: <reason>
```

The operating system returned an error when *crdir* tried to link the "." entry to the directory itself. This message is followed by an interpretation of the error returned by the operating system.

```
Error linking .. to parent of <dir_name>: <reason>
```

The operating system returned an error when *crdir* tried to link the newly created directory to its parent. This message is followed by an interpretation of the error returned by the operating system.

```
Error setting owner for <dir_name>: <reason>
```

Initially, the *crdir* command creates the new directory with the owner *system.* It then changes the owner to the user who executed the command. In this case, the operating system returned an error when *crdir* tried to change the owner of the directory. This message is followed by an interpretation of the error returned by the operating system.

```
Syntax: crdir <dir_name_list>
```

The *crdir* command expects at least one argument. This message indicates that the argument count is wrong.

SEE ALSO

dperm
perms
remove

create

Create an empty file for each file name on the command line.

SYNTAX

```
create <file_name_list>
```

DESCRIPTION

The *create* command creates an empty file for each name specified on the command line. If the file does not exist, it is created with *rw-rw-* permissions (unless altered with the *dperm* command), and the owner is the user who executes the command. If the file already exists, the owner and permissions remain intact. However, the file is truncated to a length of 0. You need write permission in the directory that you are creating a new file.

ARGUMENTS

<file_name>	The name of the file to create. The last component of a file name may not contain more than 55 characters. The *create* command ignores any additional characters.

EXAMPLES

```
create test
```

Creates the file *test* in the user's working directory.

```
create /julie/test
```

Creates the file *test* in the directory */julie*.

ERROR MESSAGES

```
Error creating <file_name>: <reason>
```

The operating system returned an error when *create* tried to create <file_name>. This message is followed by an interpretation of the error returned by the operating system.

```
Syntax: create <file_name_list>
```

The *create* command requires at least one argument. This message indicates that the argument count is wrong.

SEE ALSO

```
edit
```

date

Display or set the time and date.

SYNTAX

```
date [ [<mm>-<dd>[-<yy>]] <hr>:<min>[:<sec>] ] [+s]
```

DESCRIPTION

The *date* command has two forms: with arguments and without. Any user may execute the *date* command without any arguments. In response, the system returns the current date and time. The user *system* may also use the *date* command with arguments to set the system date and time. If the user *system* uses the *+s* option, the system reads the hardware clock and sets the date and time accordingly.

ARGUMENTS

<mm>	A number from 1 to 12 inclusive representing the month.
<dd>	A number from 1 to 31 inclusive representing the day.
<yy>	A two-digit number representing the last two digits of the year.
<hr>	A number from 0 to 23 inclusive representing the hour. (Time must be expressed as 24-hour-clock time.)
<min>	A number from 0 to 59 representing minutes past the hour.
<sec>	A number from 0 to 59 representing seconds past the minute. The default is 0.

OPTIONS

s	The *s* option tells the system to set the system date from the internal hardware clock.

EXAMPLES

```
date 7-13-84 15:47:28
```

Sets the date to July 13, 1984, and the time to 3:47:28 P.M.

```
date 11:53
```

Sets the time to 11:53 A.M. The date defaults to the date stored in memory and the value for seconds defaults to zero.

```
date 7-13 17:5
```

Sets the date to July 13 and the time to 5:05 P.M. The value for the year defaults to the stored value, and the value for seconds defaults to zero.

```
date
```

Displays the date and time currently stored in memory.

```
date +s
```

Sets the date and time to correspond to that in the system hardware clock.

ERROR MESSAGES

```
Invalid <arg> specified.
```

The value specified for the argument shown in the error message is not within the acceptable range.

```
Only the system manager may change the date!
```

The user who tried to change the date is not the system manager.

```
Syntax: date [ [<mm>-<dd>[-<yy>]] <hr>:<min>[:<sec>] ]
```

The syntax of the command line is incorrect. Most probably, the arguments specifying the time are missing or mistyped.

debug

debug invokes a machine-language debugging system.

SYNTAX

```
debug [<image_file_name>]
```

DESCRIPTION

The *debug* command is used to aid in the testing and debugging of machine-language programs. Because all programs are ultimately translated into machine language, any program may be debugged using *debug*.

The *debug* command is used to examine or modify the image of a machine-language program. This image can be (1) a post-mortem memory dump of a program which has been aborted by the operating system, (2) a program image file, or (3) a program which is currently executing under the control of *debug*. If no image file is specified on the command line, the default is the file *core* in the working directory. The *debug* command examines the file to determine whether it is a *core* image or an executable image file. If it is neither, *debug* issues the message *Invalid image type* and terminates. The third type of image may be created only by specifying the name of an executable image on the command line, followed by executing "x" command to create the controlled task.

The commands available with *debug* allow the user to examine memory locations within the program image, to modify memory locations, to set breakpoints, to execute single instructions (to single step through the program), to examine and change registers, and more. Some commands, such as single step, are applicable only when *debug* is being used to control the execution of a task. However, most commands are available for use with all image types.

ARGUMENTS

<image_file_name> The name of the file to debug. The default is the file *core* in the working directory.

OPTIONS

The *debug* command normally works in an interactive environment. The basic command structure is designed to be simple to use and to remember. In general, each command name is a single character, which may be followed by one or more expressions.

Expressions may include the operators "+" and "-", which are evaluated from left to right unless parentheses are used. Expressions may also include any of the following terms:

$<num>	The hexadecimal value of <num>.
<num>	The hexadecimal value of <num>. If this form is used, the number must start with a digit. If it starts with a character, *debug* interprets it as a symbol.
#<num>	The decimal value of <num>.
<symbol>	The value of the specified symbol. Symbol names must be completely specified — that is, all char characters are significant.
<register>	The contents of the specified register. The register may be D0 through D7, A0 through A7, SR, or PC. The letters used in specifying a register may be either uppercase or lowercase. A "." means, the last memory address accessed.

debug includes these commands:

+	Execute a shell command.
=	Display the value of an expression in multiple formats.
?	Display the *help* menu.
b	Set a breakpoint.
B	List the breakpoints that are currently set.
c	Clear one or all breakpoints.
d	Dump a section of memory.
g	Continue execution of a program.
G	Execute the program until reaching a branch or a breakpoint.
i	Disassemble instructions.
I	Initialize symbol table.
k	Terminate the currently executing task.
K	Remove any pending signals for the controlled task.
m	Modify bytes in memory.
M	Display the current memory map.
n	Display the command line for the task.
q	Terminate *debug*.
r	Display the contents of all registers.

R	Set the contents of a register.
s	Execute a single instruction.
S	Set a temporary breakpoint at the instruction following the current instruction and execute the current instruction.
T	Trace instructions until reaching a branch or a breakpoint.
x	Create a task to be executed under the control of *debug*.
<CR>	A carriage return performs the same as the "i" command, but with no address.

The following paragraphs describe *debug* commands in more detail:

```
+ <shell_command>
```

This command allows the user to execute a single shell command without exiting *debug*.

```
= <expression>
```

This command displays the value of the expression symbolically, in hexadecimal, and in decimal.

```
?
```

This command displays a menu of commands available from *debug*.

```
b <location> [<count>]
```

The "b" command sets a breakpoint at the given location. When the program is executed, the instruction at the given location is replaced by a special instruction which indicates to the operating system that the user wants to break the flow of the program. When this instruction is executed in the program, the operating system suspends the program and notifies *debug*, which prints the location of the breakpoint and returns to command mode. If the user specifies a count, the breakpoint is executed <count> times before execution is halted and *debug* notified. Once the count is exceeded, execution is halted every time the breakpoint is encountered unless it is reset by another "b" command or cleared.

```
B
```

The "B" command lists each breakpoint which is currently set as well as the corresponding <count> if it is nonzero.

```
c [<address>]
```

If the user does not specify an address, the "c" command prompts for permission to clear all breakpoints that are currently set. If the user does specify an address, it clears the breakpoint at that address.

```
d <address_1> [<address_2_or_count>]
```

The "d" command dumps the hexadecimal contents and the ASCII equivalents of a range of memory locations. Memory is displayed sixteen addresses to a line. Nonprintable characters are represented in ASCII by a period.

If the user specifies only one argument, the command displays the contents of the specified address. If the user specifies two arguments and the second one is greater than the first, the command interprets the second argument as an address. It displays the contents of memory from the first specified address to the second, inclusive. If the user specifies two arguments and the second one is less than or equal to the first, the command interprets the second argument as a count. It displays the contents of memory beginning at the first address and continuing for the number of addresses specified by the second argument.

The dump may be aborted by typing the return key during the dump. CTRL-C does not abort the command.

```
g
```

The "g" command continues the execution of a controlled task. Execution continues until the program terminates, receives a signal or encounters a breakpoint. The user may use this command only when executing a controlled task.

```
G
```

The "G" command executes the program until it encounters any branch instruction, any call instruction, or any breakpoint.

```
i [<address_1> [<address_2_or_count>]]
```

The "i" command displays the contents of memory from the first specified address to the second, inclusive. If the user specifies two arguments and the second one is less than or equal to the first, the command interprets the second argument as a count. The "i" command interprets the specified location or range of locations as machine-language instructions and advances the location counter to the start of the last complete instruction within the specified range. If the user specifies no second argument or if the range specified by the second argument is shorter than the complete instruction, the command displays the instruction which begins at the starting address but does not move the location counter. A carriage return by itself is equivalent to the command "i", except that the location counter is advanced to the beginning of the next instruction.

```
I
```

The "I" command initializes debug's internal symbol table. The symbol table is used to interpret symbolic addresses and values. The "I" command prompts for the name of the file containing the symbol table to use. The file must be a binary image file. This command is normally for use with a core image file, because such files do not contain any symbolic information. Once the symbol table is initialized, however, a core image file can be interpreted symbolically.

```
k
```

The "k" command terminates execution of the current controlled task. If no controlled task exists, the command is not allowed. This command need not be used, because the "x" command implicitly kills any controlled task before creating another.

```
K
```

When a task running under the control of *debug* receives a signal, the operating system notifies *debug* and suspends the task. The *debug* program then enters command mode, allowing the user to execute any *debug* command. A user who wishes to ignore the signal may do so by entering the "K" command. A user who wishes the signal to take effect should simply continue the program with the "g" (or a similar) command.

```
m <address>
```

The "m" command modifies the contents of one or more memory locations in the image file. In response to this command, *debug* first displays the specified address and its contents. The user may change the contents by entering any expression, may leave the contents as is by entering a period, or may terminate the command by entering just a carriage return. Unless the user terminates the command, *debug* modifies the contents if appropriate, displays the next address with its contents, and waits for input from the user.

If the image file is a core dump or an executable file, the file itself is modified. If the image file is a controlled task (i.e., an "x" command has been executed), only the memory of that task is altered. The executable file from which *debug* created the task is not changed. Therefore, when patching code the user should be aware that patches are applied only to the executing image file.

```
M
```

The "M" command displays a map of the logical addresses available to the task image. If the image is either a core dump or a controlled task, the map contains the ranges of addresses being used by the program. These ranges may change whenever the program executes a *break* or a *stack* system call. If the image is an executable file, the "M" command displays the ranges of the addresses of the TEXT and DATA/BSS segments.

```
n
```

The "n" command displays the command line which was used to create the task. This is merely a display of the command arguments passed to the program when it was created. In most cases the command line consists of the shell command used to invoke the program. The command line for a controlled task looks just like the command line entered with the "x" command that created it, except that the "x" is replaced by the program name.

```
r
```

The "r" command displays the contents of the registers for the image file, as well as the address of the program counter and the instruction located at that address. For a core dump it displays the contents of the registers at the time the program was aborted by the system and the location of the program counter at that time. The instruction displayed is the instruction that was in progress when the program was aborted. For a controlled task, the "r" command displays the contents of the registers as they will be when execution resumes, the address at which execution will resume, and the instruction at that address. The registers for an executable file are undefined. For an executable file, the "r" command displays the contents of the registers as zeros and the address and contents of the entry point of the program.

```
R <register_name> <expression>
```

The "R" command, which may be used only if the image file is a controlled task, alters the contents of a register. The register may be D0 through D7, A0 through A7, SR, or PC. The letters used in specifying a register may be either upper- or lowercase. The supervisor portion (the upper byte) of the status register may not be altered.

```
s
```

The "s" command executes a single machine-language instruction. When the instruction is complete, *debug* displays the state of the task (including the new program counter) and the next instruction to be executed. The "s" command uses system facilities provided by the operating system. Thus, the user may safely single-step through macro operations such as system calls.

```
S
```

The "S" command sets a temporary breakpoint at the instruction following the current instruction. This breakpoint is removed as soon as it is encountered. If another "S" command is executed before the breakpoint is encountered, it removes the original breakpoint. This command may be used with any instruction, but it is normally used with a call to a subroutine.

```
T
```

The "T" command executes the program until it encounters any branch instruction, any call instruction, or any breakpoint. After the execution of every instruction, *debug* displays the address of the next instruction and the instruction itself.

```
x [<arguments>] [<I/O_redirection>]
```

The "x" command creates a controlled task from an image file. In order to execute this command, the user must first invoke *debug* with the name of an executable image file as the argument. The task is halted before execution of its first instruction, so that *debug* can accept commands to control its execution.

I/O redirection may be accomplished using the character "<" to redirect standard input, ">" to redirect standard output, and "%" to redirect standard error. No provisions are made for using either append mode (>>) or implied mapping (>%).

NOTE

The more breakpoints you set, the longer the program takes to execute.

ERROR MESSAGES

`Breakpoint table full!`

The user has already set the maximum number of breakpoints.

`Can't access core/image <image_file_name>`

The operating system returned an error when *debug* tried to access the specified file. Most probably, either the file does not exist or the user does not have read permission in the file.

`Can't open <file_name>`

The *debug* command was unable to open the file which the user specified as the file containing the symbol table to use. Most probably, either the file does not exist or the user does not have read permission in the file.

`Can't write <image_file_name>`

The user tried to use the "m" command to modify the contents of a memory location in the image file, but *debug* was unable to write to the file. Most probably, the user does not have write permission in the file.

`Command too complicated`

The user tried to use the "+" command to execute a shell command from *debug*, but the command line was too long for *debug* to interpret.

`Error during EXEC - <error_num>`

The operating system returned an error when the user tried to create a controlled subtask using the "x" command. This message is followed by the error number returned by the operating system.

`Error in expression`

The expression used contains a syntax error.

`Illegal address`

The address specified is not in the user's address space.

`Illegal command, <char>, - ignored`

The command specified by <char> is not a valid command for *debug*. The character is ignored, and *debug* prompts the user for another command.

```
Illegal file type
```

The "I" command cannot determine the file type of the image file and, consequently, ignores the file. All previously defined symbols are no longer defined.

```
Illegal register name
```

The register name specified by the user is not a valid register name. The register name must be one of the following: D0 through D7, A0 through A7, SR, or PC. The letters used may be upper- or lowercase.

```
<image_file_name> is not executable
```

The user does not have execute permission in the specified image file.

```
Invalid image file <file_name>
```

The file specified to the *debug* command must be either an executable file or a core dump.

```
No command line
```

The file being debugged is not a core file, and was not invoked with the "x" command. Therefore, no command line exists for the file.

```
Not executing a task!
```

The command specified can execute only if the user has previously executed the "x" command.

```
Sorry, can't execute a core file
```

The "x" command cannot be executed on a core file.

```
** Syntax error
```

The "x" command cannot parse the specified command line.

```
Undefined symbol
```

An expression contains a term which appears to be a symbol (starts with a letter or an underscore character, "_") but is not in the symbol table. Hexadecimal values used in expressions must begin with a digit (a leading 0 is accepted) or a dollar sign,"$".

dir

List either the contents of a directory or information about a file.

SYNTAX

```
dir [dir_name_list ] [<file_name_list>] [+abdflrsSt]
```

DESCRIPTION

The *dir* command is used to list either the names of the files in the specified directory or, if the argument is not a directory, information about the specified file. By default, the names of the files in a directory are listed in alphabetical order with several names per line.

FORMAT OF THE OUTPUT

The information given about a file is presented on one line, which contains several fields. These fields are described here in the order in which they appear.

<fdn_num> — The number of the file descriptor node (fdn) which describes the file in question. This field is not present unless the user specifies the "f" option.

<file_name> — The name of the file being described.

<size> — The size of the file in blocks or bytes. If the file is a device, *dir* places the major and minor device numbers in this field.

<file_type> — A single character specifying the type of file. The character "b" represents a block device; "c", a character device; and "d", a directory. If the field is blank, the file is a regular file.

<perms> — This field, which is composed of six columns, indicates what permissions are associated with the file. The first three columns represent permissions for the user who owns the file; the last three for other users. Permissions are always presented in the order read, write, and execute. They are represented by the letters "r", "w" and "x". A hyphen in a column means that the corresponding permission is denied. For example, if the permission field contains the sequence *rwxr-x*, the user who owns the file may read, write, and execute the file, whereas other users may only read and execute it.

<link_count> — The link count is the number of directory entries which point to a file. The link count for a directory is always at least 2 because the "." entry within the directory itself points to the same fdn as the directory entry for that file in its parent directory.

<owner>	The name of the user matching the user ID number found in the system password file. If no user ID number is found, the user ID is printed surronded by double brackets, i.e. <<12>>.
<last_mod_time>	The time and date at which the file was created or last modified.

ARGUMENTS

<dir_name_list>	A list of directory names to process.
<file_name_list>	A list of the names of files to process. The default is the working directory.

OPTIONS

a	List all files in a directory, including those whose names begin with a period, ".". This option has no effect if the specified file is not a directory.
b	List the file size in bytes rather than blocks. This option implies the "l" option.
d	If the file being processed is a directory, list the names of all files it contains. Continue this process for all descendant directories. This option allows the user to see the entire directory structure.
f	List the number of the file descriptor node for each file. This option implies the "l" option.
l	If the specified file is a directory, give detailed information about each file in the directory. This option has no effect if the specified file is not a directory because in such a case the information is automatically given.
r	If the specified file is a directory, reverse the order in which the files would otherwise be listed.
s	If the specified file is a directory, list one file name on each line. This option is useful for creating a file which contains the names of all the files in a directory.
S	Print a summary of the information after listing all files.
t	This option sorts *all* files in a directory by the time last modified. It cannot be used to sort specific files or groups of files (via wildcard characters). By default, the most recently modified file is listed first.

EXAMPLES

```
dir +l
```

Lists information about each file in the working directory (except those whose names begin with a period).

```
dir /jay +abdfS
```

Lists information about all files, including those whose names start with a period, in the directory */jay* (the "f" and the "b" option both imply the "l" option). In addition, the command displays a list of the files in each subdirectory that is a descendant of */jay*. The information includes the fdn number of each file. The size of each file is shown in bytes. At the end of the output is a summary showing the total number of directories processed, the total number of nondirectory files processed, and the total number of blocks used by all the files.

```
dir memo +f
```

Displays information about the file *memo* in the working directory. The information includes the fdn number of the file.

```
dir /marcy +rt
```

Lists the names of those files in the directory */marcy* which do not begin with a period. The names are sorted by the time of the last modification with the sense of the sort reversed so that the most recently modified file is the last one in the list.

```
dir /marcy +s
```

Lists the names of those files in the directory */marcy* that do not begin with a period. One name appears on each line.

ERROR MESSAGES

```
Unknown option: <char>
```

The option specified by <char> is not a valid option to the *dir* command.

```
Warning: directory <dir_name> is too large!
Some directories were ignored
```

The *dir* command cannot process a file if the total number of directories in every directory between that file and the directory specified on the command line exceeds 50. In order to make the command succeed, the user should start at a lower point in the directory tree.

```
Warning: directory <dir_name> is too large!
Some files were ignored
```

The *dir* command cannot list more than 500 file names from a single directory. In order to make the command succeed, the user should split the offending directory into two or more directories.

dirs

List the current working directory and the directory stack created by the *pushd* command and maintained by the shell.

SYNTAX

```
dirs
```

DESCRIPTION

List the current working directory and the directory stack created by the *pushd* command and maintained by the shell. The directory stack is listed top first.

SEE ALSO

popd
pushd
shell

dperm

Set the default permissions for the creation of files by the current shell program or by tasks generated by the current shell program.

SYNTAX

```
dperm [<perms_list>]
```

DESCRIPTION

Every time a user creates a file, the operating system assigns it a set of permission bits which determines whether the file's owner and other users may read, write, or execute the file. The permissions assigned depend on the command used to create the file. The editor *edit*, for example, creates all files with *rw-rw-* permissions, which allow the user who owns the file, as well as other users, to read and write, but not execute, the file. The default permission for *crdir* are *rwxrwx*; for *create*, *rw-rw-*; for *makdev*, *rw-r—*.

The *dperm* command, which is part of the shell program, is used to set the default permissions for the creation of a file. It allows the user to instruct the system always to deny certain permissions, independent of how the file is created. It is possible to independently turn off any of the permission bits for the file's owner and other users. If the user specifies no arguments, the operating system restores the default permissions.

It is only possible to deny permissions with the *dperm* command. The *perms* command may be used to add permissions to individual files, overriding the defaults set by *dperm*.

ARGUMENTS

<perms_list> A list defining the permission bits to be used as defaults.

FORMAT FOR ARGUMENTS

<perms_list> The first character of an element in a permissions list specifies if the argument applies to the user who owns the file ("u") or to other users ("o"). The second character must be a minus sign, "-", which indicates that the following permissions are to be denied. The minus sign is followed by one, two, or three of the characters "r", "w", and "x" (for read, write, and execute, respectfully).

EXAMPLES

```
dperm o-rwx
```

Sets the default permissions so that the operating system denies all permissions to other users whenever it creates a file.

```
dperm u-w o-wx
```

Sets the default permissions so that the operating system denies write permission to the user who owns the file, and both write and execute permission to other users whenever it creates a file.

```
dperm
```

Removes all default permissions.

NOTE

The dperm *command is only effective while the shell program under which it is invoked is running. The default permissions for files created by the login shell can be permanently altered by placing the appropriate command in the file* .login *in the user's home directory. This file is automatically executed each time the user logs in.*

ERROR MESSAGES

```
Error in permissions specification.
```

The format of the permissions list is incorrect. Most likely, the user has specified a plus sign, "+", instead of a minus sign, or has used an invalid character.

SEE ALSO

perms

dump

Send both a hexadecimal and an ASCII listing of a file to standard output.

SYNTAX

```
dump <file_name> [+i]
dump [<file_name_list>]
```

DESCRIPTION

The *dump* command sends a hexadecimal and an ASCII listing of a file to standard output. The two versions of the file appear side by side. A line of output consists of the address in the file at which that line starts, the hexadecimal contents of the byte at that address and of the following fifteen bytes, and the sequence of characters represented by these bytes. A nonprintable character appears as a period, ".", in the ASCII part of the listing.

The user may interrupt the *dump* command at any time by typing a CTRL-C. Normally, a CTRL-C returns the user to the shell program. However, if the *dump* command is in interactive mode and is actually displaying information when the user types a CTRL-C, *dump* stops the output and prompts for another address.

ARGUMENTS

<file_name>	The name of the file to dump. The default is standard input.
<file_name_list>	The name of files to dump. You can not use this interactivley.

OPTIONS

i — Enter interactive mode. The "i" option may be used only if exactly one file name appears on the command line. If the user specifies the "i" option, the *dump* command prompts for the address at which to begin. The address is relative to the first byte in the file, whose address is 0. An address preceded by a period is a decimal address; otherwise it is a hexadecimal address. The user may specify a single address, a range of addresses (two addresses separated by a hyphen, or an initial address and an offset (an address followed by either a comma or a space, followed by a number). In the first case, the *dump* command displays sixteen bytes of information, beginning with the specified address. In the second case, it displays all the bytes from the first to the second address inclusive. In the third case, it begins displaying bytes at the address specified and continues for as many bytes as the following number dictates.

EXAMPLES

```
dump memo /cynthia/letter
```

Sends both a hexadecimal and an ASCII listing of the file *memo*, which is the working directory, and the file *letter*, which is in the directory */cynthia*, to standard output.

```
dump letter +i
```

Enters interactive mode and prompts the user for the address at which to begin the dumping the file *letter*.

```
dump testprog >test.dump
```

Sends a hexadecimal and ASCII listing of the file *testprog* via redirected I/O to the file *test.dump*.

ERROR MESSAGES

```
Cannot interactively dump multiple files.
```

The "i" option may not be used if more than one file name appears on the command line.

```
Cannot interactively dump standard input.
```

If the user specifies no file name on the command line, the default is standard input. The "i" option may not be used in such a case.

```
Error opening <file_name>: <reason>
```

The operating system returned an error when *dump* tried open *<file_name>*. This message is followed by an interpretation of the error returned by the operating system.

```
Invalid option <char>: ignored.
```

The option specified by *<char>* is not a valid option to the *dump* command. The command ignores it.

echo

Write the arguments on the command line to standard output.

SYNTAX

```
echo [<argument_list>] [+l] [+<hex_num>]
```

DESCRIPTION

The *echo* command writes the arguments in <argument_list> to standard output. A space character appears after each string argument; no space appears after a hexadecimal argument; while the last argument is followed by a carriage return. You can use *echo* to non-destructively show how the *shell* or *script* programs evaluate special characters in the <argument_list>.

ARGUMENTS

<argument_list>	A list of arguments to write to standard output.

FORMAT FOR ARGUMENTS

<argument_list>	Each element in <argument_list> consists either of a string or a hexadecimal number preceded by a plus sign, "+".

OPTIONS

l	Do not write a carriage return after echoing the argument list.
<hex_num>	Send the equivalent hex byte to standard output.

EXAMPLES

```
echo This is a test!
```

Writes the string *This is a test!* to standard output, which defaults to the console.

```
echo This is a test! +7 +l >/dev/console
```

Writes the string *This is a test!*, followed by the bell character (hexadecimal 7), to standard output. Standard output is redirected to /dev/console (the 4400 display). The output is not followed with a carriage return. (The *+l* is the option *plus el*, not the hexadecimal argument *plus one*.)

edit

Invoke the text editor in order to create a new text file or edit an existing one.

SYNTAX

```
edit [<file_name_1> [<file_name_2>]] [+bny]
```

DESCRIPTION

The *edit* command may be used with zero, one, or two arguments. With one argument, *edit* opens the specified file for editing, creating it if necessary, and reads as much of the file as possible into the edit buffer. At the end of an editing session of a pre-existing file, the editor renames the original file by appending the letters *.bak* to its name. If this addition would result in a file name of more than 55 characters (the maximum allowed by the operating system), the editor shortens the original name before adding the suffix. If a backup file already exists, the editor prompts for permission to delete it.

If the user specifies no arguments, the editor prompts for the name of the file at the end of the editing session, before returning control to the operating system. It does not accept the name of an existing file.

If the user specifies two file names, the operating system makes a copy of the first file specified, gives it the name specified by the second argument, and opens it for editing. If a file with that name already exists, the editor prompts for permission to delete it before proceeding. In such a case, the editor creates the new file with the same permissions as the old file.

Files created by the editor have permissions of *rw-rw-*.

ARGUMENTS

<file_name_1>	The name of the file to open for editing, or, if two file names are specified, the name of the file to copy.
<file_name_2>	The name to give to the copy of the file specified by <file_name_1>. It is this copy that is opened for editing.

OPTIONS

b	Do not save the original copy of the file as a backup file at the end of the editing session.
n	Do not read any text into the edit buffer. This option allows the user to make large insertions at the beginning of a file.
y	If only one argument appears on the command line, at end of the editing session automatically replace any existing backup file with the original copy of the file being edited. If two arguments appear on the command line and the second file specified already exists, delete that file at the beginning of the editing session.

EXAMPLES

```
edit test +ny
```

Opens the file *test* in the working directory but does not read any of it into the edit buffer. If the file does not exist, the editor creates it. At the end of the session, *edit* automatically replaces any existing backup file with the original copy of *test*.

```
edit test oldtest
```

Makes a copy of the file *test*, names it *oldtest*, and opens it for editing. If a file named *oldtest* already exists, the editor asks for permission to delete it.

MESSAGES

```
Delete existing copy of new file?
```

The file specified by *<file_name_2>* already exists. If the user responds with a "y", the editor deletes the existing copy of the file and opens the new file for editing. If the user responds with an "n", the editor leaves the existing file intact and returns the user to the operating system.

```
File already exists
File name?
```

The *edit* command was executed with no arguments on the command line. At the end of the editing session, when the editor prompted for the name of the file, the user specified an existing file. Under these circumstances, the editor does not accept the name of an existing file.

ERROR MESSAGES

`Cannot create new file`

The editor cannot open the file specified by *<file_name_2>*. Most probably, either the user specified a path name that could not be followed or the user does not have the permissions necessary to open the file.

`Cannot open edit file`

The editor cannot open the file specified by *<file_name_1>*. Most probably, either the user specified a path name that could not be followed or the user does not have the permissions necessary to open the file.

`Cannot read edit file`

The editor encountered an I/O error trying to read the specified file.

`Edit file does not exist`

The user has specified two file names on the command line, but *<file_name_1>* does not exist.

`New file is the same as the old file`

Both *<file_name_1>* and *<file_name_2>* refer to the same file. (If their names are not the same, they are links to the same file.)

`Too many file names specified.`

The *edit* command requires zero, one, or two arguments. This message indicates that the argument count is wrong.

`Unknown option specified`

An option on the command line is not a valid option to the *edit* command. The command ignores the option and proceeds.

SEE ALSO

dperm
Section 4 *EDIT The Text Editor*

exit

This shell command terminates a subshell.

SYNTAX

```
exit
```

DESCRIPTION

The *exit* command, which is part of the shell program, terminates a subshell. *exit* sounds the bell if the user attempts to exit the login shell.

EXAMPLES

```
exit
```

This is the only valid form of the *exit* command.

SEE ALSO

shell

env

Change or display the environment variables.

SYNTAX

```
env
env <name=><value>
```

ARGUMENTS

<name=>	The name of the environment variable
<value>	The value assigned to an environment variable

DESCRIPTION

The *env* command, which is part of the script program, displays the current values of the environment variables if no argument is given. If an argument is specified, the *env* command assigns, changes, or deletes the value of the named argument.

EXAMPLE

```
env
```

Displays the environment variables.

```
env TERM=
```

Removes the environment variable TERM from the environment variable list.

```
env TERM=4404
```

Assigns to the environment variable list TERM=4404.

SEE ALSO

script

fdup

Duplicate floppies.

SYNTAX

```
fdup
```

DESCRIPTION

The *fdup* command duplicates diskettes by reading the master floppy and then writing/verifying one or more copies of the master. This is the only reliable procedure to duplicate diskettes on the system. This procedure should be used to make a working copy of the software shipped with your 4400 series system.

EXAMPLES

```
fdup
```

This is the only form of this command.

filetype

SYNTAX

```
filetype <file_name_list>
```

DESCRIPTION

This utility attempts to identify the type of the files specified on the command line. Some of the types recognized are:

Directories
Character/Block devices
Many types of binary files
Many types of ascii text files
Many types of smalltalk files

The *filetype* command makes an intelligent guess as to the type of text file based on the first character of each line. Binary files are detected based on known header information.

ARGUMENTS

<file_name_list> The list of file names to process.

EXAMPLE

```
filetype myfile /mark/yourfile
```

This example will attempt to identify the type of the files *myfile* and *yourfile* in the directory */mark*.

find

Search for a string in a file or in standard input.

SYNTAX

```
find [+bcnsu] <str_1>[&<str_2>] [<file_name_list>]
```

DESCRIPTION

The *find* command looks in the specified file for the specified string. By default, lowercase characters and uppercase characters are distinct.

ARGUMENTS

<str_1>	The string to search for.
<str_2>	The second string to search for (only if "&", the *and* operator, is used).
<file_name_list>	A list of the names of files to search. The default is standard input.

SPECIFYING A STRING

The user may completely specify a string or may take advantage of the matching characters recognized by the *find* command. Because some of these matching characters also have special meanings to the shell program, strings which use them must be enclosed in single or double quotation marks.

\	When used just before any matching character, including itself, the backslash character negates the matching ability of the character.
?	The question mark matches any character except a new-line character.
<	A left angle bracket specifies that the following string must be found at the beginning of a line. It loses its matching ability if it is not the first character of the string.
>	A right angle bracket specifies that the preceding string must be found at the end of a line. It loses its matching ability if it is not the last character of the string.
&	The *and* operator may be used between two strings (see the syntax statement). The *find* command reports only those lines on which both strings occur.
[ ]	Square brackets enclose a list or a range of characters from which the *find* command can choose when looking for a string. A list of characters consists of adjacent characters. A range consists of two characters separated by a hyphen.

!	The exclamation point may be used in conjunction with the square brackets. If it is the first character inside the brackets, the *find* command can choose from all characters not specified in the brackets when looking for a string.

OPTIONS

Any options used with the *find* command must appear immediately after the command name.

b	Check file names ending in ".bak".
c	Do not print the lines that contain the specified string to standard output, instead, report the number of lines containing the string.
n	Do not print line number on match.
s	Print skipped filenames.
u	Do not distinguish between upper- and lowercase.

EXAMPLES

```
find +u syntax test
```

Writes to standard output all lines from the file *test* which contain the string *syntax*. The command does not distinguish between upper- and lowercase.

```
find +u "<syntax>" test trial
```

Writes to standard output all lines from the files *test* and *trial* which contain the string *syntax* at the beginning of the line. The command does not distinguish between upper- and lowercase. Because matching characters are used to specify the string, the string must be enclosed in either single or double quotation marks.

```
find +u "syntax&statement" test
```

Writes to standard output all lines from the file *test* which contain both the string *syntax* and the string *statement*.

```
find +c "\<" test
```

Writes to standard output the number of lines in the file *test* which contain a left-hand angle bracket. The matching ability of the angle bracket is negated because of the backslash character which precedes it.

```
find +u "[a-e]nd" test
```

Writes to standard output all lines from the file *test* which contain any of the following strings: *and*, *bnd*, *cnd*, *dnd*, or *end*.

ERROR MESSAGES

```
Error opening <file_name>: <reason>
```

The operating system returned an error when *find* tried to open the specified file. This message is followed by an interpretation of the error returned by the operating system.

```
Error processing <file_name>: <reason>
```

The operating system returned an error when *find* tried to process the specified file. This message is followed by an interpretation of the error returned by the operating system.

```
Invalid option: <char>.  Command aborted.
```

The option specified by <char> is not a valid option to the *find* command.

```
Syntax: find [+bcnsu] <str_1>[&<str_2>] [<file_name_list>]
```

The *find* command expects at least one argument. This message indicates that the argument count is wrong.

SEE ALSO

shell
script

format

Format a diskette for use on the 4400 flexible disk drive.

SYNTAX

```
format [+Fnqv] [+f=<blocks>]
```

DESCRIPTION

The *format* command formats a diskette for use in the 4400's disk drive, */dev/floppy*. The device model name is *TEK4400* which formats the diskettes as double-sided, double-density, 40 TPI, with eight 512-bit sectors per track.

OPTIONS

f=<blocks>
: Establish <blocks> blocks for file descriptor nodes (fdns). Formatted disks use fdn blocks (each fdn block contains eight fdns) to hold information about files on the disk. By default, *format* uses 3% of the total disk space for fdn blocks. You can overide this default value with the "f" option and specify the decimal number of fdn blocks to establish on the disk. At least one block must be allocated for fdns on every formatted disk.

F
: This option does not physically format the diskette. It performs a logical format only and erases all data on the diskette.

n
: Do not issue the input prompts.

q
: Before actually starting to format the diskette, *format* normally sends a prompt to ask if the user is ready to continue. The "q" (quiet) option suppresses this prompt and inhibits all informative messages from *format* if no errors are encountered during formatting.

v
: Verify the disk after formatting. The "v" (verify) option instructs *format* to verify the media after formatting. If this option is specified, *format* individually verifies every sector on the diskette. It first writes an arbitrary pattern to each sector; then reads and verifies each one. It reports any sectors which fail this test to the user.

  The option is often desirable when the user is formatting a diskette because diskettes do not automatically verify all written data.

free

Report the amount of free and used space on the specified devices.

SYNTAX

```
free <dev_name_list> [+d]
```

DESCRIPTION

The *free* command reports the amount of free space remaining on the specified device. It reports both the total number of free blocks available for use in files and the total number of file descriptor nodes (fdns) available. The number of fdns available tells the user how many more files can be created on the device (assuming that sufficient free blocks remain for use in the files). If the number of available fdns drops to 0, no more files can be created on the disk, no matter how many free blocks remain.

The number of used blocks and file descriptor nodes (fdns) is also printed.

ARGUMENTS

<dev_name_list>	A list of the names of the devices to report on. The devices may be either mounted or unmounted.

OPTIONS

d	Provide more detailed information with the output. This extra information is the amount of swap space on the disk.

EXAMPLES

```
free /dev/disk
```

Reports both the number of fdns available and the number of free blocks on the standard winchester hard disk.

```
free /dev/floppy
```

Reports both the number of fdns available and the number of free blocks on the mounted flexible disk.

ERROR MESSAGES

```
Cannot open <dev_name>
```

The specified device does not exist; the specified device exists, but no hardware is connected to it; or the device exists and hardware is connected to it, but no disk is in the device.

```
<dev_name> is not a block device.
```

The specified device must be a block device.

```
Unknown option: <char>
```

The option specified is not a valid option to the *free* command.

headset

Change information in the binary header of an executable file.

SYNTAX

```
headset <file_name_list> [+aAbBcCdSt]
```

DESCRIPTION

The *headset* command can alter certain portions of the binary header of an executable object module. Features such as whether or not the module is shared-text, whether or not the module can produce a core dump, and the initial stack size can be altered without reloading the module.

The characters used for options are identical to those used when invoking the loader with the *load* command. Those options which do not take an argument can be disabled by preceding the character with a minus sign, "-", instead of the usual plus sign, "+".

ARGUMENTS

<file_name_list>	A list of the names of the files to process.

OPTIONS

a=<num>	Specifies the minimum number of pages to allocate to this task at all times. The minimum value for the argument is 0; the maximum, 32767. The default is 0. The operating system tries to honor the specified number, but if it cannot, it uses as many pages as it needs.
A=<num>	Specifies the maximum number of pages to allocate to this task at all times. The minimum value for the argument is 0; the maximum, 32767. The default is 0. The operating system tries to honor the specified number, but if it cannot, it uses as many pages as it needs.
b=<task_size>	Specifies the maximum size to which the task may grow. The argument <task_size> may be *128K*, *256K*, *512K*, *1M*, *2M*, *4M*, or *8M*. The default task size is generated by the loader. The letters "M" and "K" can be either uppercase or lowercase. If the task size specified by the user is not large enough to hold the code from all the modules being loaded, ***headset*** automatically adjusts the size to the smallest value that can contain all the code.
+B/-B	Set or clear a bit in the binary header of the output module which tells the operating system not to zero either the BSS space or any memory allocated while the task is running.

c=<source_type>	Sets a flag in the binary header of the output module which indicates the type of source code from which the module was created. The argument <source_type> may be *ASSEMBLER* or *C*. The names can be specified in either upper- or lowercase.
C=<config_num>	By default, the loader uses the configuration number of the current hardware. The user may, however, use the "C" option to specify a configuration number which overrides the default. This option is useful when loading a module for a machine other than the one on which it is running.
+d/-d	Set or clear the *no core dump* bit in the binary header.
S=<hex_num>	Specifies the initial stack size, which is written into the binary header of the module produced by the loader. The hexadecimal number is the number of bytes to reserve. The default is 0, in which case the system assigns the default stack size of 4K.
+t/-t	Set or clear the shared-text bit in the binary header.

EXAMPLES

```
headset mathtest +t -d +S=2000
```

Makes the executable object module *mathtest* a shared-text module. It turns off the *no core dump* bit, so that the program can produce core dumps, and sets the initial stack size to hexadecimal 2000.

```
headset run_1 run_2 +tB +a=10
```

Changes the headers in the files *run_1* and *run_2*. Both modules become shared-text modules. The operating system will zero neither the BSS space nor any memory allocated while the task is running. The minimum page allocation is set to ten pages.

NOTES

- The user may make a change in a header which results in an inconsistent header. In such a case the *headset* command makes whatever adjustments are necessary in the fields which were not changed to remove the inconsistency. The user is notified of these adjustments.
- For example, if the user alters the initial stack size, the task size might have to be changed. If this change is necessary, *headset* notifies the user and adjusts the task size to the appropriate value. Adjustments may also be made when either the minimum or maximum page allocation is altered.
- If the task size specified by the user is not large enough to hold the code from all the modules being loaded, *headset* automatically adjusts the size to the smallest value that can contain all the code.
- If the user changes either the minimum or the maximum value for page allocation so that the minimum is greater than the maximum, *headset* automatically adjusts them according to the following rules.

- The value for the maximum is always greater than or equal to the value for the minimum.
- The value for the maximum can be 0, but if it is greater than 0, it must be at least 4.

MESSAGES

```
File <file_name>: changed max page allocation to <num>.
```

The user specified a minimum page allocation that was above the current maximum page allocation. The utility set the maximum equal to the minimum.

```
File <file_name>: changed min page allocation to <num>.
```

The user specified a maximum page allocation that was below the current minimum page allocation. The utility set the minimum equal to the maximum.

```
File <file_name>: task size set to <task_size>.
```

The *headset* command had to adjust the task size either because the user specified an initial stack size that made the module larger, or because the task size specified on the command was too small for the calculated size of the module.

ERROR MESSAGES

```
Error opening <file_name>: <reason>
```

The operating system returned an error when *headset* tried to open the specified file. This message is followed by an interpretation of the error returned by the operating system.

```
Error processing <file_name>: <reason>
```

The operating system returned an error when *headset* tried to process the specified file. This message is followed by an interpretation of the error returned by the operating system.

```
Error reading <file_name>: <reason>
```

The operating system returned an error when *headset* tried to read the specified file. This message is followed by an interpretation of the error returned by the operating system.

```
Error seeking in <file_name>: <reason>
```

The operating system returned an error when *headset* tried to seek in the specified file. This message is followed by an interpretation of the error returned by the operating system.

```
Error writing to <file_name>: <reason>
```

The operating system returned an error when *headset* tried to write to the specified file. This message is followed by an interpretation of the error returned by the operating system.

```
File <file_name> is not a binary file.
```

The specified file does not contain a binary header.

```
File <file_name> is not a regular file.
```

The specified file is either a device or a directory.

```
File <file_name> is not executable.
```

The specified file is not an executable binary file.

```
Illegal configuration specified.
```

The configuration type must be between 0 and 255 inclusive.

```
Illegal hex number: <hex_num>.
```

The number specified is not a valid hexadecimal number.

```
Illegal maximum page allocation specified.
```

The maximum page allocation must be between 0 and 32767 inclusive.

```
Illegal minimum page allocation specified.
```

The minimum page allocation must be between 0 and 32767 inclusive.

```
Illegal task size specified.
```

The argument specified is not a valid argument to the "b" option.

```
Invalid option: <char>.
```

The option specified by <char> is not a valid option to the *headset* command.

```
Minimum page allocation greater than maximum.
```

Both the "a" and "A" options appeared on the command line, but the minimum page allocation specified was greater than the maximum.

```
Unknown source type specified.
```

The argument specified is not a valid argument to the "c" option.

SEE ALSO

Load

help

Display a brief description of the use and syntax of the specified command.

SYNTAX

```
help [<command_name_list>]
```

DESCRIPTION

The *help* command displays a brief description of the use and syntax of the specified command. To obtain this information, it looks for a file in the */gen/help* directory with the same name as the specified command. Descriptions of most 4400 commands are available. If you enter *help help* or *help* with no arguments, the *help* command displays a list of all the commands it can help with and prompts for the name of a specific command. Typing a carriage return terminates the command.

ARGUMENTS

<command_name_list>	A list of the names of commands about which the user wants information.

EXAMPLES

```
help copy remove
```

Displays brief descriptions of the use and syntax of the *copy* and *remove* commands.

```
help
```

Displays a list of all the commands that the *help* command can help with, followed by a prompt for the name of a specific command.

NOTES

- The *system* user may add files to */gen/help*. When the *help* command is executed, it simply looks for the specified file in */gen/help*, reads the contents, and writes it to standard output.
- If the file specified is a directory in the */gen/help* directory, the *help* command lists the contents of the directory and asks what command the user would like help with. If the command specified is not in that directory, *help* prompts for permission to search */gen/help*.

ERROR MESSAGES

```
Cannot help with <command_name>.
```

No description of the specified command is available to the *help* command.

```
Error opening <file_name>: <reason>
```

The operating system returned an error when *help* tried to open the file <file_name>, which describes the specified command. This message is followed by an interpretation of the error returned by the operating system.

```
Error reading <file_name>: <reason>
```

The operating system returned an error when *help* tried to read the file <file_name>, which describes the specified command. This message is followed by an interpretation of the error returned by the operating system.

```
Too many files in directory.
```

The *help* command cannot function if the directory */gen/help* contains more than 500 entries.

history

A shell command that displays list of previous commands.

SYNTAX

```
history
```

DESCRIPTION

The *history* command, which is part of the shell program, displays list of previous commands. Scrolling and editing functions are selected by control keys (or function key sequences) and may be used to recall and modify commands. The command history will be saved from one login to the next in the file *is limited to 30 commands.*

EXAMPLES

```
history
```

This is the only valid form of the *history* command. This command lists the previous 30 commands, not including the *history* command itself.

SEE ALSO

shell

info

Display the contents of the information field associated with the specified binary file.

SYNTAX

```
info <file_name_list>
```

DESCRIPTION

A binary file may have an *information field* that stores textual information associated with the file. This information can include things like the version number and release date of the file, as well as other useful information pertaining to the file. The *info* command displays the contents of the information field.

ARGUMENTS

<file_name_list> A list of the names of the files for |which to display the information field.

EXAMPLES

```
info /system.boot
```

Displays the version number, release date, and copyright information for the file */system.boot*, the operating system itself.

```
info /bin/edit /bin/info
```

Displays version numbers, release dates, and copyright information for the text editor (*/bin/edit*) and the *info* command (*/bin/info*).

ERROR MESSAGES

```
Error opening <file_name>: <reason>
```

The operating system returned an error when *info* tried to open the file <file_name>. This message is followed by an interpretation of the error returned by the operating system.

```
Error processing <file_name>: <reason>
```

The operating system returned an error when *info* tried to process the file <file_name>. This message is followed by an interpretation of the error returned by the operating system.

```
Error reading <file_name>: <reason>
```

The operating system returned an error when *info* tried to read the file <file_name>. This message is followed by an interpretation of the error returned by the operating system.

```
Error seeking in <file_name>: <reason>
```

The operating system returned an error when *info* tried to seek to the appropriate location in <file_name>. This message is followed by an interpretation of the error returned by the operating system.

```
Error writing to standard output: <reason>
```

The operating system returned an error when *info* tried to write the output of the *info* command to standard output. This message is followed by an interpretation of the error returned by the operating system.

```
<file_name> has no information field.
```

The optional information field is not present in the specified file.

```
<file_name> is not a binary file.
```

The specified file lacks the header which identifies it as a binary file. The argument to the *info* command must be a binary file.

```
<file_name> is not a regular file.
```

The specified file is a directory or a special file (a block or character device). The argument to the *info* command must be a regular file.

```
Syntax: info <file_name_list>
```

The *info* command requires at least one argument. This message indicates that the argument count is wrong.

SEE ALSO

4400 Series Assembly Language Reference

int

Send a program interrupt to another task.

SYNTAX

```
int <task_ID> [+<int_num>] [+s]
```

DESCRIPTION

The *int* command sends the specified interrupt to the task identified by the task ID on the command line. If the user does not specify an interrupt number, the termination interrupt (SIGTERM) is sent. Task ID's are reported by the shell program whenever the user executes a task in the background. An ID can also be determined by the *jobs* or *status* command.

ARGUMENTS

<task_ID>	The task ID of the task to interrupt. A task ID of 0 specifies all tasks associated with the user's terminal and owned by the user.
+<int_num>	The number of the interrupt the user wishes to send. The plus sign, "+", is necessary to distinguish the number of the interrupt from the task ID. Table 2-1 shows a list of the possible interrupts. The default interrupt number is #11, SIGTERM.
s	Send a *soft* interrupt.

NOTES

A = Default state is *abort* (otherwise, *ignore*)
C = Interrupt can be caught
D = Produces a core dump
I = Interrupt can be ignored
R = Resets to default state when triggered

Table 2-1
POSSIBLE INTERRUPTS

Name	Number	Description	A	C	D	I	R
SIGHUP	1	Hangup	+	+	-	+	+
SIGINT	2	Keyboard	+	+	-	+	+
SIGQUIT	3	Quit	+	+	+	+	+
SIGEMT	4	EMT $Axxx emulation	+	+	+	+	+
SIGKILL	5	Task kill	+	-	-	-	+
SIGPIPE	6	Broken pipe	+	+	-	+	+
SIGSWAP	7	Swap error	+	-	-	-	+
SIGTRACE	8	Trace	+	+	-	+	-
SIGTIME	9	Time limit	+	+	+	-	+
SIGALRM	10	Alarm	+	+	-	+	+
SIGTERM	11	Task terminate	+	+	-	+	+
SIGTRAPV	12	TRAPV instruction	+	+	+	+	+
SIGCHK	13	CHK instruction	+	+	+	+	+
SIGEMT2	14	EMT $Fxxx emulation	+	+	+	+	+
SIGTRAP1	15	TRAP #1 instruction	+	+	+	+	+
SIGTRAP2	16	TRAP #2 instruction	+	+	+	+	+
SIGTRAP3	17	TRAP #3 instruction	+	+	+	+	+
SIGTRAP4	18	TRAP #4 instruction	+	+	+	+	+
SIGTRAP5	19	TRAP #5 instruction	+	+	+	+	+
SIGTRAP6	20	TRAP #6-14 instruction	+	+	+	+	+
SIGPAR	21	Parity error	+	-	+	-	+
SIGILL	22	Illegal instruction	+	-	+	-	+
SIGDIV	23	DIVIDE by 0	+	+	+	+	+
SIGPRIV	24	Privileged instruction	+	-	+	-	+
SIGADDR	25	Address error	+	-	+	-	+
SIGDEAD*	26	Dead child	-	+	-	+	+
SIGWRIT	27	Write to READ-ONLY memory	+	-	+	-	+
SIGEXEC	28	Execute from STACK/DATA space	+	-	+	-	+
SIGBND	29	Segmentation violation	+	+	+	-	+
SIGUSR1	30	User-defined interrupt #1	+	+	-	+	+
SIGUSR2	31	User-defined interrupt #2	+	+	-	+	+
SIGUSR3	32	User-defined interrupt #3	+	+	-	+	+
SIGABORT	33	Program abort	+	-	+	-	+
SIGSPLR	34	Spooler interrupt	+	+	-	+	+
SIGINPUT	35	Input is ready	+	+	-	+	+
SIGDUMP	36	Memory dump	+	+	+	+	+
	37-41	User-defined interrupts					
SIGUNORDERED%	42	FPU branch/set on unordered	+	+	-	+	+

* The operating system does not reset the signalling mechanism after once set (i.e. with an cpint(SIGDEAD,addr) call). The parent task must reset with an cpint(SIGDEAD,addr) call.

% These interrupts are produced only by the MC68881 Floating Point Co-processor.

Table 2-1 (cont.)
POSSIBLE INTERRUPTS

Name	Number	Description	A	C	D	I	R
SIGINEXACT%	43	FPU inexact result	+	+	-	+	+
SIGFPDIVIDE%	44	FPU divide by zero	+	+	-	+	+
SIGUNDERFLOW%	45	FPU underflow	+	+	-	+	+
SIGOPERAND%	46	FPU operand error	+	+	-	+	+
SIGOVERFLOW%	47	FPU overflow	+	+	-	+	+
SIGSNAN%	48	FPU signaling NAN	+	+	-	+	+
	49-61	User-defined interrupts					
SIGMILLI	62	Millisecond alarm	+	+	-	+	+
SIGEVT	63	Mouse/keyboard event interrupt	+	+	-	+	+

* The operating system does not reset the signalling mechanism after once set (i.e. with an cpint(SIGDEAD,addr) call). The parent task must reset with an cpint(SIGDEAD,addr) call.

% These interrupts are recognized only on the 4406 AIM system.

EXAMPLES

```
int 263
```

Sends a termination interrupt (SIGTERM) to task number 263.

```
int +5 149
```

Sends a SIGKILL interrupt to task 149. No program can trap or ignore a SIGKILL interrupt.

```
int 149 +5
```

This example is identical to the previous example. The order of the arguments is irrelevant.

ERROR MESSAGES

```
Error sending interrupt: <reason>
```

The operating system returned an error when *int* tried to send the interrupt. This message is followed by an interpretation of the error returned by the operating system, such as could not find the specified task.

```
Illegal interrupt specified: <int_num>
```

The number specified must be an integer between 1 and the number of signals, inclusive.

```
Illegal task ID specified: <task_ID>
```

The task ID specified contains some characters that are not digits. A legal task ID contains only digits.

```
Syntax: int <task_ID> [+<int_num>]
```

The *int* command expects exactly one task ID and no more than one interrupt number. This message indicates that the argument count is wrong.

SEE ALSO

jobs
status

jobs

Report the task IDs and starting times of all background tasks originated by the user from the current shell program. This is a shell command.

SYNTAX

```
jobs
```

DESCRIPTION

The *jobs* command, which is part of the shell program, reports the task IDs and starting times of all background tasks originated by the user from the current shell program. (If *script* is running as the current shell, the task IDs are preceded by the letter "T" for task. This letter is not part of the task ID.)

EXAMPLES

```
jobs
```

This example is the only valid form of the *jobs* command. It reports the task ID and starting time of all active background tasks originated by the user from the current shell program.

MESSAGES

```
No tasks active.
```

The user has no active tasks in the background.

SEE ALSO

int
status

libgen

Create a new library or update an existing one.

SYNTAX

```
libgen o=<old_lib> n=<new_lib> [u=<update>] [<del_list>] [+al]
```

DESCRIPTION

The *libgen* command creates a new library of relocatable or executable modules or updates an existing library. Each module in a library must have a name. The name is assigned to a module by either the *name* pseudo-op in the relocating assembler or the "N" option of the linking loader. The *libgen* command does not accept a module without a name.

As it runs, *libgen* produces a report describing the action that it takes for each module in the library. The report includes the name of the module and the file from which it was read (the old library or one of the update files).

ARGUMENTS

o=<old_lib>	The name of an existing library file that was previously created by the *libgen* command. *libgen* is being called to update an existing library rather than to create a new one. Either the *o=<old_lib>* or *n=<new_lib>* argument, or both, must appear on the command line.
n=<new_lib>	The name of a new library. If a file with this name already exists, *libgen* deletes it without warning before writing the new library. If the user does not specify a name for the new library, it defaults to the name of the old library. In such a case *libgen* puts the new library in a scratch file, deletes the old library, and renames the scratch file with the name of the old library. Either the *o=<old_lib>* or *n=<new_lib>* argument, or both, must appear on the command line.
u=<update>	The name of a file containing modules to add to the library. Modules of the same name are replaced by modules from the update file.
<del_list>	A list of the names of modules to delete from the old library.

OPTIONS

a	Produce an abbreviated report that contains information only about modules that were replaced, added, or deleted.
l	Suppress the production of a report.

EXAMPLES

```
libgen n=binlib u=one u=two u=three
```

Creates a new library named *binlib* that contains all the modules from the files *one, two,* and *three.*

```
libgen o=binlib u=new +a
```

Updates the library *binlib* by adding or replacing modules from the file *new.* The command produces an abbreviated report.

```
libgen o=binlib u=newmods n=newlib transpose add +l
```

Updates the library *binlib* by adding or replacing modules from the file *newmods* and by deleting the modules named *transpose* and *add.* The updated library is written to the file *newlib.*

ERROR MESSAGES

```
An old or new library name must be specified.
```

Either the *o=<old_lib>* or *n=<new_lib>* argument, or both, must appear on the command line.

```
No index found in <lib_name>
```

The *libgen* command creates every library with an index. This message indicates either that the file specified is not a library or that it is a library, but has been badly damaged, and can no longer be used.

```
Record not found in <module_name>
```

One of the files in the list of modules to delete from the old library was not found in that library. The command ignores that file name and continues.

```
Record with no name found in <module_name>
```

Every relocatable or executable module that goes into a library must have a name. The user should remake the specified module and give it a name.

```
Unknown argument: <str>
```

The argument specified by *<str>* is not a valid argument to the *libgen* command.

```
Unrecognizable record in <module_name>
```

All modules in a library must be either executable or relocatable.

SEE ALSO

libinfo
4400 Series Assembly Language Reference

libinfo

Display information about a library.

SYNTAX

```
libinfo <library_name_list> [+em] [M=<mode_name>]
```

DESCRIPTION

The *libinfo* command lists the entry points and module names contained in a library produced by the *libgen* command. The user can optionally display only the entry points or only the module names. Information about a particular module within a library can also be displayed.

ARGUMENTS

<library_name_list> A list of the names of the libraries to report on.

OPTIONS

e	Display only entry points in the specified library.
m	Display only module names in the specified library.
M=<mod_name>	Display information about module <mod_name>. This option is incompatible with both the "e" and "m" options. If the user specifies incompatible options, *libinfo* uses the "M" option and ignores any others.

EXAMPLES

```
libinfo testlib
```

Lists all entry points and module names in the library *testlib*.

```
libinfo runlib +m
```

Lists all the module names contained in the library *runlib*.

```
libinfo /lib/cmathlib +M=Arctan
```

Displays the entry points and module names in the module *Arctan* in the library */lib/cmathlib*.

ERROR MESSAGES

```
Error opening <file_name> : <reason>
```

The operating system returned an error when *libinfo* tried to open the specified file. This message is followed by an interpretation of the error returned by the operating system.

```
Error reading <file_name> : <reason>
```

The operating system returned an error when *libinfo* tried to read the specified file. This message is followed by an interpretation of the error returned by the operating system.

```
Error seeking to <location> in <file_name> : <reason>
```

The operating system returned an error when *libinfo* tried to seek to the specified location (in hexadecimal) in the specified file. This message is followed by an interpretation of the error returned by the operating system.

```
<file_name> is not a library!
```

The file specified does not have the correct format for a library created with the *libgen* command.

```
** 'M' taken, others ignored ***
```

The "m" and "e" options are incompatible with the "M" option. If the user specifies incompatible options, *libinfo* uses the "M" option and ignores any others.

```
Unknown option <char> ignored.
```

An unknown option was found and ignored.

SEE ALSO

libgen
relinfo

link

Establish a new link to an existing file.

SYNTAX

```
link <file_name_1> <file_name_2>
```

DESCRIPTION

The *link* command establishes a new link to an existing file. If the command is successful, both <file_name_1> and <file_name_2> refer to the same file.

The user must have write permission in the parent directory in which the new link is created, and must have execute permission in the directory containing the original copy of the file. A link cannot cross devices.

ARGUMENTS

<file_name_1>	The name of the existing file to which to establish a link.
<file_name_2>	The name of the link to the existing file.

EXAMPLES

```
link /susan/.editconfigure .editconfigure
```

Creates a file named *.editconfigure* in the user's working directory and links it to the existing file *.editconfigure* in the directory */susan.*

ERROR MESSAGES

```
Cannot link across devices
```

The specified file names reside on different volumes and, therefore, cannot be linked.

```
Entry already exists: <file_name_2>
```

The file specified by <file_name_2> must be a nonexistent file.

```
Entry does not exist: <file_name_1>
```

If the file to which the link is to be made does not exist, it is impossible to link the files.

```
Entry is a directory: <file_name_1>
```

The existing file specified is, in fact, a directory. Only the system manager can link to a directory.

```
Invalid options: +<char>
```

The *link* command supports no options.

```
Path cannot be followed: <file_name>
```

One or more of the directories that make up the name of the file do not exist.

```
Permissions deny access: <file_name>
```

The user does not have permission to access the specified file. If the file is the existing file, <file_name_1>, the user does not have execute permission in the parent directory. If the file is <file_name_2>, the user does not have write permission in the parent directory.

```
Syntax: link <filename> <linkname>
```

The *link* command expects exactly two arguments. This message indicates that the argument count is wrong.

SEE ALSO

copy
move

list

Write the contents of the specified file to standard output.

SYNTAX

```
list [<file_name_list>] [+l] [+<num>]
```

DESCRIPTION

The *list* command writes the contents of the specified file to standard output. If the user specifies more than one file, the files are listed one after the other with no space between them.

The default file name is standard input. A plus sign, "+", may also be used as an argument to indicate standard input.

ARGUMENTS

<file_name_list>	A list of the names of the files to write to standard output. The default is standard input.

OPTIONS

l	Include line numbers in the listing.
<num>	The number of the line at which to begin listing the file.

EXAMPLES

```
list test
```

Writes the file *test* to standard output.

```
list test +120 >>test.out
```

Also writes the file *test* to standard output. Standard output is redirected so that the listing is appended to the contents of the file *test.out*. The listing is accompanied by line numbers and starts at line 20 of the file.

```
list part_1 part_2 + part_3 >whole_thing
```

Writes the files *part_1* and *part_2*, followed by the text entered from standard input (end standard input with a CRTL-D), followed by *part_3*, to the file *whole_thing*.

ERROR MESSAGES

```
Error listing <file_name>: <reason>
```

The operating system returned an error when *list* tried to write <file_name> to standard output. This message is followed by an interpretation of the error returned by the operating system.

```
Error opening <file_name>: <reason>
```

The operating system returned an error when *list* tried to open the file <file_name>. This message is followed by an interpretation of the error returned by the operating system.

```
Error reading <file_name>: <reason>
```

The operating system returned an error when *list* tried to read the file <file_name>. This message is followed by an interpretation of the error returned by the operating system.

```
Invalid option: <char>.  Command aborted!
```

The option specified by <char> is not a valid option to the *list* command.

```
Invalid starting line number.  Command aborted!
```

The string used to specify the starting line of the listing either is not a string of digits or is too large.

SEE ALSO

page
tail

load

The *load* command invokes the linking loader.

SYNTAX

```
load <file_name_list> [+a=<num>] [+A=<num>] [+b=<task_size>]
     [+B] [+c=<module_type>] [+C=<configuration>] [+d]
     [+D[=<hex_num>]] [+ef] [+F[=<file_name>]] [+i]
     [+l=<library_name>] [+Lm] [+M=<file_name>] [+n]
     [+N=<module_name>] [+o=<file_name>] [+P=<hex_num>]
     [+rs] [+S=<hex_num>] [+t] [+T=<hex_num>] [+u]
     [+U=<trap_num>] [+x=<file_name>]
```

DESCRIPTION

The *load* command takes as input one or more relocatable binary modules and produces as output either a relocatable module or an executable module. The relocatable modules used as input should have been produced by the relocating assembler or the linking loader. Options are available for producing load and module maps as well as a global symbol table. Starting addresses for text and data segments can be adjusted for the particular hardware being used. The page size can also be adjusted. The loader can search libraries produced by the *libgen* utility in order to resolve external references.

The user can place all desired options in a file specified with the *load* command's "F" option rather than specifying them individually on the command line. The operating system comes with one such file, the file */lib/ldr_environ*, which describes the hardware environment. The loader always reads this file before processing any other options. It then processes options in the order in which they appear on the command line. If an option is specified more than once (e.g., once in a file and once on the command line), the last specification overrides all others.

ARGUMENTS

<file_name_list> A list of files to load.

OPTIONS

a=<num>	Specifies the minimum number of pages to allocate to this task at all times. The default is 0. The operating system tries to honor the specified number, but if it cannot, it uses as many pages as it needs.
A=<num>	Specifies the maximum number of pages to allocate to this task at all times. The default is 0. The operating system tries to honor the specified number, but if it cannot, it uses as many pages as it needs.
b=<task_size>	Specifies the size of the task, where <task_size> is *128K*, *256K*, *512K*, *1M*, *2M*, *4M*, or *8M*. The default is *128K*. If the argument specified by the user is not large enough, the *load* command adjusts it to the smallest possible size. The letters "M" and "K" can be either upper- or lowercase.
B	BSS space will not be cleared.
c=<module_type>	Specifies the source code of the modules, where <module_type> is *ASSEMBLER*, or *C*. The names can be specified in either upper- or lowercase.
C=<configuration>	By default, the loader uses the configuration number of the current hardware. The user may, however, use the "C" option to specify a configuration number which overrides the default. This option is useful when loading a module for a machine other than the one on which it is running.
d	Sets the *no core dump* bit in the binary header.
D[=<hex_num>]	Specifies the starting address of the data segment. If the user does not specify the option or specifies the option without an argument, the data segment immediately follows the text segment.
e	Prints each occurrence of any unresolved external. By default, the loader prints only the first occurrence.
f	In this format, text pages are loaded into the user's address space when first referenced (through page faulting) rather than at *exec* time.
F[=<file_name>]	Specifies the name of a file of options to process. The default file name is *ldr_opts*. The "F" option may be used repeatedly but may not be nested.
i	Writes all global symbols to the symbol table of the binary file.
l=<library_name>	Specifies the name of a library to search. The loader first searches the working directory, then the *lib* directory in the working directory, and finally the directory */lib*. Libraries are searched in the order specified on the command line. Up to five libraries may be specified in this manner. By default, unless the user specifies five libraries on the command line, the library */lib/Syslib68k* is the last one searched.
L	Does not search any libraries for unresolved externals.

m	Produces load and module maps and writes them to standard output (see the "M" option).
M=<file_name>	Specifies the name of the file in which to put the output of the "m" option (load and module maps) and the "s" option (a global symbol table). This information is purely textual. The user may edit or list the file like any other text file. If the "m" or "s" option is used without the "M" option, the loader sends the information to standard output.
n	Produces an executable module with separate instruction and data space.
N=<module_name>	Specifies the name to give to the file containing the module.
o=<file_name>	Specifies the name to give to the binary output file.
P=<hex_num>	Specifies the page size. The hexadecimal number should always be a power of 2; otherwise, the results are unpredictable. The *load* command uses the page size to determine the starting address of the data segment when it immediately follows the text segment (the data segment starts at the next page boundary). The default is 0 (i.e., the loader rounds the starting address to the next even location after the end of the text segment).
r	Produces a relocatable module as output. Do not search any libraries.
s	Writes the global symbol table to standard output (see the "M" option).
S=<hex_num>	Specifies an initial stack size where the hexadecimal number is the number of bytes to reserve. The default is 0 (the system determines the size of the stack).
t	Produces a shared-text executable module.
T=<hex_num>	Specifies the starting address of the text segment. Default is 0.
u	Does not print any unresolved messages when producing a relocatable module.
U=<trap_num>	Sets the trap number for system calls. The default is hardware-dependent. The user can specify the argument as either *TRAP n* where "n" is a number between 0 and 15 inclusive, or as a string of four hexadecimal digits which represent a bit pattern to use as an instruction instead of the system call.
x=<file_name>	Incremental load file name.

EXAMPLES

```
load *.r +F=/lib/ldr_environ +t +l=clib +o=tester
```

Loads all files whose names end with *.r* in the working directory. The loader reads the file */lib/ldr_environ* and processes the options therein. It uses the library *clib* to resolve externals. The executable output module, which is a shared-text module, is named *tester*.

```
load t1.r t2.r +T=20000 +iN=mod +P=2000 +c=C +o=test
```

Loads the the files specified and produces a binary file named *test*. The internal module-name is *mod*. The text segment begins at 20000 hexadecimal, and the data segment follows it at the next page boundary (page size 2000 hexadecimal). The source code is *C*. All global symbols are inserted in the symbol table of the binary file.

```
load sqrt +msM=loadmap +l=cmathlib +i
```

Loads the file *sqrt* and produces an executable module named *sqrt.o*. The loader searches the library *cmathlib* for unresolved externals. It produces load and module maps, as well as a symbol table, and writes them to the file *loadmap*. All global symbols are added to the symbol table of the binary file.

```
load temp?.r +reo=combined.r
```

Loads the files in the working directory whose names match the pattern *temp?.r* and produces a relocatable module named *combined.r*. The loader prints each occurrence of all unresolved externals rather than only the first occurrence of each. Because the "r" option is specified, the loader does not search any libraries.

```
load t1.r t2.r +a=10 +A=100 +b=2M +l=testlib +do=test
```

Loads the files *t1.r* and *t2.r* and produces the binary file named *test*. The minimum page allocation is set to 10; the maximum, to 100. The task size of the module is set to 2 Megabytes. The executable module does not produce a core dump.

NOTES

- If the file */lib/std_env* contains information about the starting address of the text segment, the data segment, or both, and if the user wishes to override this standard configuration, starting addresses for both text and data segments should be specified.
- If the user specifies page allocation values that don't make sense, the loader automatically adjust them according to the following rules:

  The value for the maximum is always greater than or equal to the value for the minimum. The value for the maximum can be 0, but if it is greater than 0, it must be at least 4.

SEE ALSO

cc
headset
4400 Series Assembly Language Programmer's Reference

login

Give a user access to the operating system.

SYNTAX

```
login <user_name>
login [user_name]
```

DESCRIPTION

The *login* command gives a user access to the operating system. If the user does not have a password, the system automatically honors the command. If the user does have a password, the system requests it. If it is entered correctly, the user is given access to the operating system. Otherwise, the system returns an error message, followed by a login prompt.

ARGUMENTS

<user_name>	The name of the user to put in contact with the operating system. If no <user_name> is supplied, the system prompts for it.

EXAMPLES

```
login leslie
```

This example tells the operating system to give the user whose user name is *leslie* access to the operating system.

ERROR MESSAGES

```
Login incorrect.
```

The combination of the user name specified and the password entered is invalid. This message is followed by a login prompt.

```
No login name specified.
```

When using the script program, the user did not specify a user name on the command line.

SEE ALSO

log
logout
script
shell

logout

This shell command terminates an active session.

SYNTAX

```
logout
```

DESCRIPTION

The *logout* command, which is part of the shell program, terminates an interactive session.

EXAMPLE

```
logout
```

This is the only valid form of the *logout* command.

SEE ALSO

login
shell

move

Rename a file or move a file to another directory.

SYNTAX

```
move <file_name_1> <file_name_2> [+klps]
move <file_name_list> <dir_name> [+klps]
```

DESCRIPTION

The *move* command moves or renames one or more files. The first form of the command renames <file_name_1> to <file_name_2>. The second form moves each file named in <file_name_list> to <dir_name>. In either case, if there is already a file with the same name as the file created by the *move* command, it is overwritten without warning.

Directories and special files (block devices and character devices) may not be moved. The user must have write and execute permissions in the parent directory of each file being moved and in the directory to which the files are moved. Each original file is removed.

A file may not be moved from one device to another unless the user has read permission on the file. A file may not be moved to itself.

Normally the *move* command links the new file to the original file and deletes the original one. Thus, a link between files on different devices is not permitted; if you attempt to *move* a file to a different device, the original file is copied to the new file, then the original file is deleted.

ARGUMENTS

<file_name_1>	The name of the source file to move or rename.
<file_name_2>	The name of the destination file to which to move <file_name_1>.
<dir_name>	The name of the destination directory to which to move all the specified files.

OPTIONS

k	Do not delete the source file.
l	List the name of each file as it is moved.
p	Prompt for permission to replace existing files.
s	Stop as soon as an error is encountered.

EXAMPLES

```
move test oldtest +l
```

Renames the file *test* in the working directory; the new name is *oldtest.* The *move* command issues a message describing the move.

```
move test /elaine
```

Moves the file *test* from the working directory to the directory */elaine.* The last component of the file name is preserved, so the name of the new file is */elaine/test.*

```
move test /elaine/oldtest +kp
```

Moves the file *test* from the working directory to the directory */elaine* and renames it *oldtest.* If the file */elaine/oldtest* already exists, the user is prompted for permission to delete the file. If permission is denied, the move does not take place. Even if the move takes place, the original files remain intact.

```
move * /elaine +s
```

Moves all the files in the working directory to the directory */elaine.* Each file name is preserved. The command aborts if it encounters an error.

MESSAGES

```
<file_name_1> copied to <file_name_2>
```

This message is produced only if both the "l" and "k" options are specified and the two files are on different devices. It means that <file_name_1> has been copied to <file_name_2>, but that the original file remains intact.

```
<file_name_1> linked to <file_name_2>
```

This message is produced only if both the "l" and "k" options are specified. It means that the two files have been linked and the original file remains intact.

```
<file_name_1> moved to <file_name_2>
```

This is the normal message issued by the *move* command. It means that <file_name_1> has been either linked or copied to <file_name_2>, and that <file_name_1> has been deleted.

ERROR MESSAGES

```
Cannot move a block special file: <file_name>
```

The file <file_name> is a block special file (block device) and may not be moved.

```
Cannot move a character special file: <file_name>
```

The file <file_name> is a character special file (character device) and may not be moved.

```
Cannot move across devices: <file_name>
```

The file <file_name> is read-protected and, therefore, cannot be moved across devices.

```
Directory is not accessible: <dir_name>
```

The user does not have the necessary permissions (write and execute) to move a file to <dir_name>.

```
<file_name_1> and <file_name_2> are the same file.
```

The user tried to move a file to itself, which if allowed would destroy the file. If <file_name_1> and <file_name_2> are different, they are links to the same file.

```
Permissions deny access: <file_name>
```

The user does not have write permission in the parent of the specified directory.

SEE ALSO

copy
link

page

Page format a file or files.

SYNTAX

```
page [+lf<n>] [+p<n>] [<file_name_list>]
```

DESCRIPTION

Page format a file or files. The format includes the file name in the upper-left corner, the date and time centered, and the page number in the upper-right corner. May also be used to display lines on a terminal, <n> lines at a time. If no file is specified or if a '+' is specified, then standard input will be listed.

ARGUMENTS

<file_name_list>	The list of file name(s) to display.

OPTIONS

l	issue line numbers
f	use line feeds instead of form feeds
n	<n> is a decimal number representing crt screen length
p<n>	<n> is a decimal number representing printer page length, length must be 10 or greater.

EXAMPLES

```
page myfile +l +22
```

Formats the contents of the file *myfile* including line numbers for a screen with a length of 22 lines.

password

Set or change a user's password.

SYNTAX

```
password [<user_name>]
```

DESCRIPTION

The *password* command sets or changes a user's password. Only the system manager may change another user's password. When a user other than the system manager invokes the command, the operating system prompts for the existing password (if there is one). If the password is entered correctly, the system prompts for the new password. Generally, a password should contain between five and eight random characters. After the new password is entered, the system prompts for it again to verify it. If the second entry agrees with the first, the password is entered in the password file. In order to maintain the secrecy of the password, the operating system does not echo the characters typed in response to the prompts for either the existing or the new password.

To remove a password, enter a carriage return for the new password.

ARGUMENTS

<user_name>	The name of user whose password is being changed. The default is the user invoking the command.

EXAMPLES

```
password
```

Changes the password of the user who invoked the command.

```
password greg
```

Uses the command form that can be used only by the system manager. It changes the password associated with the user name *greg*.

ERROR MESSAGES

```
Cannot find <user_name> in the password file.
```

The file */etc/log/password* does not contain an entry for the user <user_name>.

```
Cannot find your name in the password file.
```

The file */etc/log/password* does not contain an entry for the user issuing the command. This situation is extremely unlikely to occur.

```
Error linking /tmp/pswd to /etc/log/password:<reason>
```

The operating system returned an error when *password* tried to link the new version of the password file to the old password file. This message is followed by an interpretation of the error returned by the operating system.

```
Error opening <file_name>: <reason>
```

The operating system returned an error when *password* tried to open the specified file. This message is followed by an interpretation of the error returned by the operating system.

```
Error unlinking <file_name>: <reason>
```

The operating system returned an error when *password* tried to unlink the specified file. This message is followed by an interpretation of the error returned by the operating system.

```
Error writing <file_name>: <reason>
```

The operating system returned an error when *password* tried to write to the specified file. This message is followed by an interpretation of the error returned by the operating system.

```
Only the system manager may change another's password.
```

Use of the form of the *password* command that takes an argument is limited to the system manager.

```
Password not correct.  Permission denied!
```

The user did not enter the existing password correctly.

```
Retry different password unchanged.
```

The first and second entries of the new password were not identical. The password command aborts, leaving the original password in place.

```
Syntax:  password [<user_name>]
```

The *password* command expects no more than one argument. This message indicates that the argument count is wrong.

```
System busy -  try again later.
```

The file */tmp/pswd*, which must be created by the *password* command already exists. Either someone else is using the command or it was interrupted before it had a chance to delete the temporary file. If no one is using the command, you should login as *system* and delete the file */tmp/pswd*.

path

Write the path name of the working directory to standard output.

SYNTAX

```
path
```

DESCRIPTION

The *path* command writes the path name of the working directory, followed by a carriage return, to standard output. The path name is the unique path from the root directory through the directory tree to the file in question.

EXAMPLES

```
path
```

This is the only valid form of the *path* command. It writes the name of the working directory to standard output.

ERROR MESSAGES

```
Directory structure is corrupt
```

The directory path from the root directory, "/", to the working directory is corrupt. Therefore, the *path* command cannot determine the path name of the working directory.

SEE ALSO

chd

perms

Change the permissions associated with a file.

SYNTAX

```
perms <perms_list> <file_name_list>
```

DESCRIPTION

Every time a user creates a file, the operating system assigns it a set of permission bits which determines whether or not the file's owner and other users may read, write, or execute the file. The permissions assigned depend on the command used to create the file. The editor, for example, creates all files with *rw-rw-* permissions, which allow the user who owns the file, as well as other users, to read and write, but not execute, the file. The default permission for *crdir* are *rwxrwx*; for *create*, *rw-rw-*; for *makdev*, *rw-r—*.

Read permission allows a regular file to be read. A user cannot execute commands such as *list* and *copy* without read permission on the file in question. Write permission allows a file to be modified. Execute permission allows the name of the file to be used as a command.

Permissions for directories are similar to those for normal files. Read permission allows the user to read file names that are actually in the directory. Write permission allows the user to create and delete files in the directory. Execute permission allows the directory to be searched for a name used as part of a file specification or file name. The user must have execute permission to successfully use a directory as the argument to the *chd* command.

In addition to these permissions, each file has associated with it a user ID bit. If this bit is set for a given file, any user executing the file has the same privileges as the file's owner for the duration of the task.

The *perms* command changes the permission bits associated with a file. Only the owner of a file or the system manager may change the permissions associated with it.

ARGUMENTS

<perms_list>	The list of permission bits to alter. Permission bits not mentioned are not changed.
<file_name_list>	A list of the names of the files for which to alter the permissions.

FORMAT FOR ARGUMENTS

<perms_list> The first character of an element in the permissions list specifies whether the argument applies to the user who owns the file ("u") or to others ("o"). The second character specifies whether to add ("+") or remove ("-") the permissions in question. The second character is followed by one, two, or three of the characters "r", "w", and "x" (for read, write, and execute). The user ID bit is set or cleared with one of the following arguments: *s+* or *s-*.

EXAMPLES

```
perms o-wx inventory
```

Removes write and execute permissions for other users from the file *inventory* in the working directory.

```
perms o+x u+x script
```

Gives execute permissions on the file *script* to both the user who owns it and to other users.

```
perms o-rw o+x s+ inventory script
```

Removes read and write permissions for others from the files *inventory* and *script*. It also sets execute permissions for others, as well as the user ID bit. Thus, although other users may neither read from nor write to the files, they may execute them. While they are executing them, they have the same permissions on all files as the owner of these files does.

ERROR MESSAGES

```
Error changing permissions for <file_name>: <reason>
```

The operating system returned an error when *perms* tried change the permissions on the specified file. This message is followed by an interpretation of the error returned by the operating system.

```
Error processing <file_name>: <reason>
```

The operating system returned an error when *perms* tried to determine the original permissions on the file. This message is followed by an interpretation of the error returned by the operating system.

```
Syntax: perms <perms_list> <file_name_list>
```

The *perms* command expects at least two arguments. This message indicates that the argument count is wrong.

```
Unrecognizable character, '<char>', found in permissions list.
Command aborted!
```

A character following a plus or minus sign in an element in the permissions list was not an "r", "w", or "x". The command aborts without altering any permissions.

SEE ALSO

chd
dir
dperm

popd

Changes the working directory to the one whose name is on the top of the directory stack. This is a shell command.

SYNTAX

```
popd
```

DESCRIPTION

The *popd* command is a part of the shell program and changes the working directory to the one whose name is on the top of the directory stack. The directory stack is created by the *pushd* command.

EXAMPLES

```
popd
```

This is the only valid form of the *popd* command.

SEE ALSO

dirs
pushd
shell

pushd

Push the name of the working directory on the directory stack and change to the specified directory. This is a shell command.

SYNTAX

```
pushd [<dir_name>]
```

DESCRIPTION

The *pushd* command, which is part of the shell program, pushes the name of the working directory on the directory stack and changes to the specified directory. With no argument, exchanges the top of the directory stack and the current working directory. The *dirs* command may be used to view the directory stack.

ARGUMENTS

<dir_name>	The name of directory to change to after pushing the current directory onto the stack.

EXAMPLES

```
pushd ~/Lang
```

Pushes the current working directory on the directory stack and changes to the directory *Lang*.

SEE ALSO

dirs
popd
shell

relinfo

Display information about an object file.

SYNTAX

```
relinfo <file_name_list> [+ehrs]
```

DESCRIPTION

The *relinfo* command examines an object file or all the modules in a library and displays information about the binary header, the symbol table, and both the relocation and external records. Normally, *relinfo* displays all the information. The available options restrict the display to the specified information.

ARGUMENTS

<file_name_list>	A list of the names of files to report on.

OPTIONS

e	Display information about external records.
h	Display information about the binary header.
r	Display information about relocation records.
s	Display information about the global symbol table.

EXAMPLES

```
relinfo tester
```

Displays information about the binary header, the symbol table, and both the relocation and external records in the object file *tester* in the working directory.

```
relinfo /lib/cmathlib +h
```

Displays the information about the binary headers from all the modules in the library */lib/mathlib*.

```
relinfo reporter +se
```

Displays the information about both the relocation and external records in the file *reporter* in the working directory.

ERROR MESSAGES

```
Error opening <file_name> : <reason>
```

The operating system returned an error when *relinfo* tried to open the specified file. This message is followed by an interpretation of the error returned by the operating system.

```
Error reading <file_name> : <reason>
```

The operating system returned an error when *relinfo* tried to read the specified file. This message is followed by an interpretation of the error returned by the operating system.

```
Error seeking to <location> in <file_name> : <reason>
```

The operating system returned an error when *relinfo* tried to seek to the specified location (in hexadecimal) in the specified file. This message is followed by an interpretation of the error returned by the operating system.

```
<file_name> is not a binary file!
```

The specified file does not have a valid binary header.

```
Unknown option <char> ignored.
```

An unknown option was found and ignored.

SEE ALSO

libgen
libinfo
load
asm

remote

Communicate with a host computer via the RS-232 port, */dev/comm*.

SYNTAX

```
remote [+l=<filename>] [+n]
```

DESCRIPTION

The utility *remote* allows the Tektronix 4400 series to be used as a terminal to a remote host computer connected to the */dev/comm* port.

Remote allows you to capture both sides of a session with a host into a disk file for later editing and review. In addition, this utility also allows file transfers to and from the host under control of a host program.

OPTIONS

l=filename	Output from the host will be directed to the specified file in addition to being sent to the terminal emulator and appearing on the screen. This function can be toggled on and off using function key F3.
+n	This options specifies that linefeed characters be ignored when directing to the file specified by the "+l" option. The "+l" option must be specified for this option to have any meaning.

FUNCTION KEY ACTIONS

F1	Terminates remote.
F2	Create and enter a subshell. Any executing file transfers will continue uninterrupted.
F3	Toggles output to file specified by the +l option on and off.

FILE TRANSFERS

Remote supports a file transfer protocol which works in conjunction with a program running on the remote host. The "C" source code for a sample of such a program, which will run under the Unix<tm> operating system, may be found in */samples/xfer.c*

CONFIGURING THE COMMUNICATIONS PORT

The *commset* command is used to set the various parameters of the communications port. For example, the baud rate of the part may be set with a command like:

```
commset baud=9600
```

See the documentation on the *commset* command for further information on configuring the communications port.

EXAMPLES

```
remote +l=temp +n
```

Communicates with a remote host through the device */dev/comm/*. When you toggle the capture buffer with the function key F3, all activities are recorded in the file *temp*. Pressing the key F3 again turns off the capture buffer. The "+n" option causes the capture buffer to ignore and not record all linefeeds.

SEE ALSO

commset

remove

Remove the specified file from the system.

SYNTAX

```
remove <file_name_list> [+dklpqw]
```

DESCRIPTION

The *remove* command removes the specified file(s), which may be any type of file(s), from the file system. The user must own the file(s), must have write permission in the parent directory of the file(s) being removed and, by default, must also have write permission for the file(s) itself, unless the "w" option is specified. Restrictions on deleting a directory are discussed with the options.

ARGUMENTS

<file_name_list>	A list of the names of files to remove from the file system. The list may include regular files, special files, and directories.

OPTIONS

d	If the specified file is a directory and it is empty, delete it. By default, the *remove* command does not delete directories.
k	If the specified file is a directory, delete it and all the files it contains.
l	List the name of each file as it is removed.
p	Prompt for permission to remove each file. The file is removed if the user responds to the prompt with a "y" or "Y".
q	Quiet mode. Do not report any errors.
w	Prompt for permission to remove files for which the user does not have write permission. By default, the *remove* command does not delete such files. The file is removed if the user responds to the prompt with a "y" or "Y".

EXAMPLES

```
remove first_file dir_file second_file +w
```

Removes the files *first_file* and *second_file*, prompting for permission to do so if the user does not have write permissions in the file. The file *dir_file* is not removed because it is a directory.

```
remove first_file dir_file second_file +dp
```

Prompts for permission to remove *first_file* and *second_file* (assuming the user has the proper permissions). Also prompts for permission to remove *dir_file* if the directory is empty.

```
remove first_file dir_file +kl
```

Removes *first_file* and *dir_file* from the file system. In addition, descends the directory structure of *dir_file*, deleting the directory itself as well as every file. Lists the name of each file as it is deleted.

CAUTION

The remove command, especially when executed with the "k" option, is an extremely powerful and potentially destructive command.

ERROR MESSAGES

```
Cannot delete the root directory: /
```

The user tried to delete the root directory.

```
Directory <dir_name> is not empty.
```

The *remove* command cannot delete a nonempty directory unless the user specifies the 'k' option.

```
Error deleting <file_name>: <reason>
```

The operating system returned an error when *remove* tried to delete <file_name>. This message is followed by an interpretation of the error returned by the operating system.

```
Error deleting . in <dir_name>: <reason>
```

The operating system returned an error when *remove* tried to delete the "." entry in <dir_name>. This message is followed by an interpretation of the error returned by the operating system.

```
Error getting status for <file_name>: <reason>
```

The operating system returned an error when *remove* tried to read the fdn for <file_name>. This message is followed by an interpretation of the error returned by the operating system.

```
Error removing <file_name>: <reason>
```

The operating system returned an error when *remove* tried to remove <file_name>. This message is followed by an interpretation of the error returned by the operating system.

```
Invalid option: <char>
```

The option specified by <char> is not a valid option to the *remove* command.

```
Syntax: remove <file_name_list> [+dklpw]
```

The *remove* command expects at least one argument. This message indicates that the argument is wrong.

```
You do not own <file_name>.
```

The user may not delete a file that is owned by someone else.

SEE ALSO

deluser

rename

Change the name of the specified file.

SYNTAX

```
rename <file_name_1> <file_name_2>
```

DESCRIPTION

The *rename* command changes the name of the specified file. If a file named <file_name_2> already exists, it is deleted without warning.

ARGUMENTS

<file_name_1>	The name of an existing file.
<file_name_2>	The new name for <file_name_1>.

EXAMPLES

```
rename test oldtest
```

Changes the name of the file *test* in the working directory to *oldtest*. If a file named *oldtest* already exists, it is deleted without warning.

```
rename test /elaine/oldtest
```

Changes the name of the file *test* in the working directory to */elaine/oldtest* if the user has write permissions in the directory *elaine*.

ERROR MESSAGES

```
Error renaming <file_name_1>: <reason>
```

The operating system returned an error when *rename* tried change the name of <file_name_1>. This message is followed by an interpretation of the error returned by the operating system.

```
Error renaming to <file_name_2>: <reason>
```

The operating system returned an error when *rename* tried to assign the new file name. This message is followed by an interpretation of the error returned by the operating system.

```
Error unlinking <file_name_1>: <reason>
```

The operating system returned an error when *rename* tried to unlink <file_name_1> from the new file. This message is followed by an interpretation of the error returned by the operating system.

```
File <file_name_1> does not exist!
```

The first name on the command line must be the name of an existing file.

```
Syntax: rename <file_name_1> <file_name_2>
```

The *rename* command expects exactly two arguments. This message indicates that the argument count is wrong.

SEE ALSO

move

restore

Catalog or restore files from the backup device onto the file system.

SYNTAX

```
restore [+bBCdLlnpr ] [+T[=length]] [<file_name_list>]
        [<dir_name_list>]
```

DESCRIPTION

The *restore* command is used to copy backup files from the backup device onto the file system. Although the program is named *restore*, it can operate in two distinct modes, selected by options: catalog mode and restore mode. Catalog mode lists the contents of the backup device in much the same format as that used by the *dir* command. Restore mode retrieves files or directories from the backup device.

The *restore* command retrieves backup files and directories from */dev/floppy* or */dev/tape*. You should not attempt to *mount* a backup device; the only way to read devices written by *backup* is to use the *restore* command. The only other command that you should use on a backup device is *devcheck*.

As files are backed up, *backup* also stores the file owner ID number, permissions, and time/date stamp of the file. This is used by *restore* when retrieving the files. After the files are restored, they appear just as they were at the time of the backup. The user should be aware of several potential problems.

First, it is possible for users with identical ID numbers to exist on different systems with different names. Since only the owner ID number is saved with the file, not the owner's name, when the file is restored, the apparent owner will be the name of the user in the password file that matches the ID number. If the user ID number does not exist in the restoring system password file, the owner of the file will be the ID number enclosed in double angle brackets, for example, <<14>>. Second, file permissions are respected during restore. If the restoring user does not have write permission for a file, it will not be restored. One method to facilitate easy movement of files among many machines is to always backup and restore the files from the public user, which exists on all machines. In any event, the user system can backup and restore any file as well as change ownership and permissions.

ARGUMENTS

<file_name_list>	List of the names of files to process. Defaults to the working directory.
<dir_name_list>	List of the names of directories to process.

If you specify a directory name as an argument in restore mode, the program processes only the files within that directory. If you also specify the "d" option, the program restores all files within the given directory and its subdirectories.

OPTIONS

b	Print sizes of files in bytes.
B	Do not restore files that end in *.bak*.
C	Print a catalog of the files on an existing backup. If you specify the "C" option, *restore* ignores all the names in <file_name_list>.
d	Restore entire directory structures.

l	List file names as they are restored.
L	Do not unlink files before restoring.
n	Only restore a file if the copy on the back up device is newer than the copy at the destination. If the destination file does not exist, the program restores the file (unless prohibited by another option, such as the "B" option). The "n" option may be used only in restore mode.
p	Prompt you with each file name to determine whether or not the restore procedure should be performed on that particular file.
r	Retension streaming tape cartridge before any action. Using this option may avoid read errors from the streaming tape drive. This option must be used in conjunction with the *+T* option.
T[=length]	Restore from streaming tape instead of floppy. The tape length parameter *L* default is 450 foot tape, eg. (*+T=300* for 300 foot tape).

NOTE

The "+d" option to restore entire directories creates subdirectories only if the original backup *command specified "/" or "." as the directory to back up. Absolute sub-directories will* **not** *be created, although the files contained within them will be restored if the subdirectory already exists. That is, the command*

backup . +dl

saves all subdirectories under the current working directory, and the command

restore +dl

restores these subdirectories and their contents. However, the command

backup +dl /dir1/subdir2

while it saves the subdirectory /dir1/subdir2 *and its contents (including subsequent subdirectories) in absolute format, the command*

restore +dl

will fail if any of the directories or subdirectories do not exist. The error messages are specific enough to allow you to manually create the directory structure necessary for restore *to work. For an example of how this is used to control directory structure, see the script file* restoreFiles *on the* SYSINSTALL *diskette.*

restore normally works in a quiet mode. The "l" option allows you to see what the program is actually doing.

EXAMPLES

```
restore +l
```

Restores all of the files, excluding subdirectories and their contents, from the backup diskettes you are prompted to insert in the flexible disk drive.

```
restore +ln file1 dir2
```

Restores the file *file1* from the backup if the backup copy is newer than any existing copy. It then restores the files contained in *dir2* on the backup, creating the directory *dir2* if necessary. Only files newer than existing copies are restored, and these are listed as they are restored. This example does not restore any subdirectories in *dir2* or any files or directories contained in subdirectories in *dir2*.

```
restore +C >catalog
```

Catalogs the files on the backup set and stores it in a file called *catalog*.

NOTES

- In restore mode, file names or directory names on the command line are used to select the files or directories to be restored. The program searches the entire backup for each argument specified. If multiple files satisfy the restoration criteria, the program restores them all, destroying the older version as the new one is restored. Thus, to ensure proper restoration, you must provide all backup volumes (in order) for each argument.
- When files are restored, they are generally restored to the same directory location as you specified when they were backed up. As files are backed up, *backup* makes an indication of the path name for each file. When files are restored, *restore* uses the path name to place the file in its proper directory location. If the path name is relative (i.e., does not begin with "/"), the path name of the restored directory is also relative. Thus, files backed up with a relative path name may be restored to a directory location different from the one in which they were created.

  An example should make this clear. If the working directory is backed up, either by specifying no source files or by using the directory name ".", the files are backed up with a relative path of ".". When these files are restored, they are placed in the directory ".", which might not be the same directory they originally came from. This feature allows the manipulation of entire file systems in a general fashion. To specify a unique directory location for a file, you should specify its entire path name, starting with "/".
- It is possible to restore backed up data onto the device currently being used as the root device or system disk. Two possible problems arise, however. First of all, if the operating system is restored from a backup, the result is not bootable. In such a case, the file must be copied from the original master diskette and installed in order to allow booting. The second problem occurs if the shell program or the device *tty00* is restored over the current shell or *tty00*. This operation leaves unreferenced files in the file system. Unreferenced files must be corrected with the *diskrepair* command. It is a good idea to run *diskrepair* on the root device after restoring backed-up data to it.

MESSAGES

```
Catalog of backup on <file_name>
Restore backup from <file_name>
```

These messages are printed when *restore* begins. They notify you of the function about to be performed.

Several of the following messages prompt you for a positive or negative response. The program interprets any response that does not begin with an upper or lowercase "n" as a positive response.

```
Restore <file_name> (y/n)?
```

If you specify the "p" option, the program prints one of these prompts before it takes any action. A response of "n" or "N" indicates that the operation should not be performed for the given file. Any other response is interpreted as *yes*.

```
Insert next volume - Hit C/R to continue:
```

This prompt is issued when the program needs a new backup volume. You should type a carriage return only when the next volume has been placed in the device.

```
link <file_name_1> to <file_name_2>
copy <file_name>
Copying from <dir_name>
```

The program prints these messages as it takes the corresponding action during a creation operation.

```
This is Volume #<number_1> -- Expected Volume #<number_2> --
Continue?
```

The program expects you to insert volumes in sequential order. If a volume appears out of order, *restore* prints this message. If you type anything except an "n" or an "N" as the first character of the response to the message, *restore* ignores the fact that the volumes are out of order and continues with the backup. Otherwise, it prompts you for another volume. It is important to insert volumes sequentially because *restore* cannot correctly restore files that are broken across volumes if the volumes are inserted out of order.

```
Volume <number> of <vol_name>
```

Whenever a new volume is inserted and properly validated, the program prints this message, which indicates the name of the backup volume and its sequence number.

ERROR MESSAGES

```
<dev_name> is not a block device
```

The destination device for the backup must be a block device. This message indicates that the specified device (that is always the first argument) is not a block device.

```
<file_name> not located - try again?
```

When using the program in restore mode, you may specify which files or directories to restore. If the program cannot find a specified file or directory after searching the entire backup, it prints this message. If the response is not "n" or "N", the program searches the entire archive again. This option is allowed because volumes need not be inserted in order of their creation when the program is in restore mode. If one volume is left out or if the final volume is inserted before the entire archive has been processed, some files might not be processed. Note that if you specify more than one file name or directory name, the program processes the entire archive for each file before proceeding to the next one.

```
Formatting not allowed during Catalog/Restore
```

You may not format a disk with the *restore* command.

```
Read error! - file <file_name>
```

An I/O error occurred during the transfer of a file either to or from the backup. An auxiliary message is printed indicating the nature of the error. The program tries to continue for all errors except *device full* during restore mode.

```
Unknown option: <char>
```

The option specified by <char> is not a valid option to the *backup* command.

```
** Warning: directory <dir_name> is too large!
**  Some directories were ignored
** Warning: directory <dir_name> is too large!
**  Some files were ignored
```

The program uses some internal tables during the back up process (not during restore or catalog). If the limits of these tables are exceeded (highly unlikely), these messages are printed.

SEE ALSO

backup

script

The script execution shell.

DESCRIPTION

The program named *script* is a command interpreter used primarily to execute commands from a file. It can be run as an interactive interface, but does not support aliases, and history, that are available under *shell.*

If you run *script* as an interactive shell, it collects and interprets your commands and executes some built-in commands (*chd, dperm, jobs, log, login, time,* and *wait)* itself. It passes others to the operating system kernel which, in turn, performs the operations requested.

A *script* command line consists of a command name, which may be followed by arguments, options, or both. All elements of the command line must be separated by either spaces or commas. The command may be one of the commands supplied with the operating system, the name of a binary file produced by either the assembler or a compiler, or the name of a text file (with execute permission turned on) which contains a series of commands to execute. In all cases the script program spawns a child-task which executes the specified command or commands.

ENVIRONMENT VARIABLES

A list of name-value pairs called *environment variables* is available to *script.* When *script* encounters a string that it recognizes as an environment variable, it emits the value stored for that variable. You can define or modify an environment variable by the command:

```
env name=value
```

You can delete an environment variable by omitting the value in the command:

```
env name=
```

SEARCH PATH

The environment variable *PATH* defines the search path for the directory containing the command. Each alternative directory name is separated by a colon. If the command name contains a /, the search path is not used. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission, but is not a binary file, it is assumed to be a file containing script commands. A subshell (i.e., a separate process) is spawned and the program *script* is used to read and execute it. A command contained within parentheses is also executed in a subshell.

BACKGROUND TASKS

If you follow a command with an ampersand, "&", the script program spawns a child-task which executes the command. However, in this case the script does not wait for the task to complete. Thus, you may start another command while the first one is executing. A single script program can support a maximum of five of these *background tasks.* Each time you send a task to the background, the script program reports the task ID assigned to that task, preceding it with a "T", which is not part of the task ID. The user may need the task ID to execute the *wait* or *int* command. The task ID may also be obtained by executing the *jobs* command, which returns the task ID and starting time of all background tasks originated by you from the script program. The ampersand may be used following a single command or separating one command from another on a single line.

MULTIPLE COMMANDS ON A LINE

You may specify more than one command on a command line by separating them with any of several special symbols.

The script program sequentially executes commands that are separated by a semicolon, ";". If a task terminates abnormally, the script program stops executing the command line.

Two additional command separators, the conjunction operator ("&&") and the disjunction operator ("||"), are available. With these separators, execution of the command following the operator is dependent on the outcome of the command preceding it. A command is *true* if it terminates with a termination status of zero, indicating successful completion, and *false* if it terminates with a nonzero termination status, indicating failure. When two commands are separated by the conjunction operator, the script program executes the second one only if it completes the first one successfully (it is *true*). When two commands are separated by the disjunction operator, the script program executes the second one only if the first one fails (it is *false*).

Normally, the command line is evaluated from left to right; however, parentheses may be used to group commands. Commands in parentheses are treated as a single command. Commands separated by a pipe (see REDIRECTED I/O) are also treated as one command.

The processing of the command separators may be summarized as follows:

&&	If the command preceding the conjunction operator succeeds, the script program tries to execute the next command. If the command preceding the conjunction operator fails, the script program looks for a disjunction operator. If it finds one, it tries to execute the command which follows it. If it does not find one, processing of the command line ceases.
||	If the command preceding the disjunction operator succeeds, the script program looks for a semicolon, ";". If it finds one, it tries to execute the command which follows it. If it does not find one, processing of the command line ceases. If the command preceding the disjunction operator fails, the script program tries to execute the next command.
;	If the command preceding a semicolon succeeds, the script program tries to execute the next command. If the command preceding a semicolon fails, processing of the command line ceases.
&	Whether the command preceding a single ampersand succeeds or fails, the script program processes the next command on the command line.

Consider the following example:

```
<task_1> && <task_2> || <task_3> && <task_4>
```

The script program first tries to execute <task_1>. If the task is unsuccessful, the script skips <task_2> and proceeds to <task_3>; if <task_3> succeeds, it tries to execute <task_4>. If, however, <task_1> succeeds, the script program tries to execute <task_2>. If <task_2> also succeeds, the script program skips the rest of the command line. If, after the successful execution of <task_1>, <task_2> fails, the script tries to execute <task_3>. If and only if <task_3> succeeds, it goes on to <task_4>.

The use of parentheses can change the interpretation of the same set of commands separated by the same operators:

```
<task_1> && ( <task_2> || <task_3> ) && <task_4>
```

In this case, the script once again begins by trying to execute <task_1>. If it fails, the script program skips the remaining tasks. If, on the other hand, <task_1> is successful, the script program spawns a subshell (because of the presence of the parentheses). This subshell tries to execute <task_2> and, if and only if it fails, it tries to execute <task_3>. If <task_2> succeeds, it returns a termination status of *true* to its parent script. If <task_2> fails but <task_3> succeeds, it also returns a termination status of *true*. If, however, both <task_2> and <task_3> fail, the termination status returned is *false*. If the termination status returned by the subshell is *true*, the parent script tries to execute <task_4>.

TERMINATION STATUS

Normally, the script program does not report the termination status of a command it executes unless the task terminates abnormally (because of a program interrupt). A list of the possible program interrupts appears in the documentation of the *int* command. The script program does, however, always report the termination status of a background task, even if it terminates normally.

REDIRECTED I/O

The script program associates three files with every command it executes: standard input, standard output, and standard error. Standard input is the file from which a command takes its input. Standard output is the file to which a command sends its output. Standard error is the file to which many error messages are directed. By default, the system uses your keyboard as standard input and your console as both standard output and standard error. However, you can direct the script to use another file for any of these standard files. This process is known as I/O redirection.

The table 2-2, *I/O Redirection*, is a quick summary of the commands for redirection. Following the table are explanations of each of the optional symbols.

Table 2-2
I/O Redirection

Symbol	Meaning
<	Take standard input from file following symbol.
>	Send standard output to file following symbol.
^ or %	Send standard error to file following symbol.
>>	Append standard output to file following symbol.
|	Connect programs so output of one becomes input of next.
>%	Redirect standard error to standard output.
%>	Redirect standard error to standard output.

The symbol "<" tells the script program to take its standard input from the file whose name follows the symbol. Similarly, the symbols ">" and "^" are used to send standard output and standard error respectively to a file. The file to which standard input is redirected must already exist. However, if the file to which standard output or standard error is redirected does not exist, the system creates it. In fact, if the file does already exist, the system deletes the contents of the file before executing the command. To avoid this effect, you may use the ">>" symbol to direct the script program to append data to the file specified as standard error or standard output. For example, you might add the results of the *compare* command to one of the pre-existing files.

It is also possible to redirect standard output or standard error (or both) to another task. This form of redirection is accomplished by using a *pipe*. A pipe is a function that connects programs so that the output from one program becomes the input for another. Standard output is piped from one task to another by using one of the symbols "|" or "^". For instance, the following example lists all the files in the working directory, formats the listing with the *page* command, and prints the listing on the printer */dev/printer*.

```
dir . | page | /dev/printer
```

Similarly, you can redirect standard error with either of the symbols "|" or "^".

Although you can place many pipes on the command line, a single task can support only one pipe. Thus, you cannot pipe standard error and standard output to separate tasks. It is possible, however, to duplicate standard error onto standard output and to redirect them both to the same task. You have a choice of symbols for duplicating standard error onto standard output: ">%" or "%>". Neither of these symbols takes an argument. After duplicating standard error onto standard output, you redirect standard output to a file or a task in the usual way. Whenever standard error and standard output are routed to the same destination, their contents may be intermingled. For instance, you can get a listing of all the files in the working directory, redirect both standard error and standard output to the *page* command, and print the results on the printer */dev/printer* with the following command:

```
ls . %> | page | /dev/printer
```

Finally, the following constructions redirect I/O from or to the null device, */dev/null*: "<-" for standard input, ">-" for standard output, and - for standard error. If either standard output or standard error is redirected to the null device, its contents are lost. If the null device is used as standard input, an end-of-file character is read.

CONTINUATION OF THE COMMAND LINE

Command lines may be continued across more than one physical line by terminating each line, except the last, with a backslash character, "\," immediately followed by a carriage return. As an interactive shell, *script* uses the prompt "+>" to indicate that the line being entered is a continuation of the previous line. When the script program processes the line, it replaces the backslash and the carriage return with a space. Typing a line-delete character (CTRL-U) only affects the physical line being typed. A keyboard interrupt (CTRL-C), deletes the entire command line.

PATTERN MATCHING CHARACTERS

The operating system recognizes several characters, known as pattern matching characters, which allow you to specify multiple files without typing each name individually. The special characters are the asterisk, "*", the question mark, "?" and square brackets, "[ ]". The script program matches these special characters to characters in the filenames in the specified directory. If the matching character appears in the last component of the file name, the script tries to match it to the names of all files in the specified directory (by default, the working directory). If the matching character appears in any other position in the file name, the script tries to match it to the names of directories only.

An asterisk in a command line matches any character or characters, including the null string but not including a leading period. Thus, the command

```
list *.bak
```

lists all files in the working directory whose names end in *.bak* and do not begin with a period.

The question mark matches any single character except the null character or a leading period. For example, the command

```
list chapter_?
```

lists all files whose names begin with the string *chapter_* and end with a single character.

You can use more than one matching character at a time. For instance, the command

```
list *.?
```

lists all files in the working directory whose names end with a period followed by a single character (except those whose names begin with a period).

Square brackets allow you to specify a set of characters to use in the matching process. The set of characters is defined by listing individual characters or by specifying two characters separated by a hyphen. In the former case, the script program looks for all file names which use any one of the enclosed characters in the appropriate place. In the latter, the two characters specify a class of characters containing the two characters themselves and any characters which lexically fall between them in the ASCII character set. For example, if your working directory contains nine files named *chapter1*, *chapter2*, *chapter3*, and so forth, the following command lists the first three chapters, the fifth chapter, and the last three chapters:

```
list chapter[1-357-9]
```

If the script program cannot find a match for any of the arguments containing matching characters, it aborts the command. If it finds a match for at least one argument containing matching characters, it ignores any other arguments containing matching characters for which it cannot find a match.

If a filename actually contains one of the matching characters or either a space or a comma, you must enclose the name in single or double quotation marks. In such a case the script program passes the arguments to the command without performing any character matching.

script SCRIPTS

A *script* script is a file that contains a list of commands for the script program. Such a file might consist of a list of commands that are frequently executed in sequence, or of a single, lengthy command that is often used. If you set execute permissions on such a file, the name of the file can be used as a command.

You may add to the versatility of a *script* script by using arguments within the script. The arguments are specified within the script as *$1*, *$2*, *$3*, and so forth. The argument *$0* specifies the name of the calling program. These arguments may appear anywhere in a command argument.

If an argument being passed to a command actually contains an ampersand, the argument must be enclosed in single quotation marks so that the script program does not try to perform any substitution. Note that single quotation marks prevent both substitution of arguments and the expansion of matching characters, whereas double quotation marks prevent the expansion of matching characters but allow the substitution of arguments.

The script program supports several commands that are used exclusively with *script* scripts. These commands—*verbose*, *exit*, *proceed*, and *sabort*—are discussed in the following paragraphs.

verbose

When the script program executes a script file, it does not normally echo the commands being executed. The *verbose* command causes the script program to echo commands from a script file as they are executed. Each line that is echoed is preceded by two hyphens and a space character.

The *verbose* command may be called without arguments or with one argument, which must be one of the strings *on* or *off*. If called without an argument, the default is *on*. The command may be executed by the login script or may be part of a script script. The verbose attribute is always passed from a parent script program to a child shell, but never from a child to a parent.

exit AND *proceed*

script permits a limited amount of control over the processing of script files. *shell* sequentially processes commands in a script file until one of the commands fails or it reaches the end of the file. If a command fails, *script* begins to search the remainder of the script file for a line that contains one of the commands *exit* or *proceed.* If it encounters one of these commands, *script* resumes processing the script after that command. The only difference between *exit* and *proceed* commands is that during successful execution of a script file *script* stops processing the file if it encounters an *exit* command, whereas it ignores a *proceed* command. The search for both these commands takes place before both the substitution of any arguments and the expansion of any matching characters. Thus, the script program does not see an *exit* or *proceed* command that is created as the result of either of these processes.

Here's an example of the *proceed* command:

```
/etc/mount /dev/floppy /usr2
/usr2/runjob
echo "Successful execution."
proceed
/etc/unmount /dev/floppy
```

In this example, *script* mounts a disk and tries to execute the command */usr2/runjob* on that disk. If the command succeeds, *script* echoes the message, *Successful execution.* and proceeds to unmount the disk. If the command fails, *script* skips all commands between the one that failed and the *proceed* command. It resumes execution with the *unmount* command. Thus, if */usr2/runjob* fails, your disk is unmounted, but no message is sent to standard output.

By adding an *exit* command you can modify this example to notify you of either successful or unsuccessful execution:

```
/etc/mount /dev/floppy /usr2
/usr2/runjob
/etc/unmount /dev/floppy
echo "Successful execution."
exit
/etc/unmount /dev/floppy
echo "Unsuccessful execution."
```

Here, if */usr2/runjob* succeeds, the script program continues execution with the *unmount* command and echoes the string *Successful execution.* to standard output. The *exit* command then causes the script program to stop processing the script because it encounters the *exit* command during normal execution. If */usr2/runjob* fails, the script program skips all commands until it encounters the *exit* command. It then resumes execution with the *unmount* command and echoes the string *Unsuccessful execution.* to standard output.

sabort

The *sabort* command can be used to turn off the search for either an *exit* or *proceed* command, thus forcing execution of every command in the script, regardless of the failure of a command.

sabort may be called without arguments or with one argument, which must be one of the strings *on* or *off*. When *sabort* is *on*, *script* looks for an *exit* or *proceed* command whenever a command in the script fails. When *sabort* is off, *script* processes every command in the script. If you execute the *sabort* command without an argument, it both rescinds the effect of any previous *sabort on* and fails. Thus, if *script* is executing a script, *script* immediately begins looking for an *exit* or *proceed* command.

The *sabort* command may be executed by a login shell (if you use *script* as your shell) or may be part of a *script* script. The attribute is always passed from a parent program to a child shell, but never from a child to a parent.

The system also supports startup files for individual users. Whenever a user logs in using *script* as an interactive shell, the script program looks for a file named *.startup* in your home directory (as defined in the password file). If the file exists and you have *read* permissions in it, *script* executes the file before issuing the system prompt.

The script program can also be used as a command in its own right. This form is used primarily to execute a *script* scriptfile for which execute permissions are not set, to call the script program from another program, or in the password file.

SYNTAX

```
script [+abcnvx] [<argument_list>] [<script_filename>]
```

DESCRIPTION OF THE *SCRIPT* COMMAND

If the *script* command is executed without any options or arguments, the operating system simply spawns another shell for you. This script program functions as a normal shell, but because it is the child of the shell or script program from which the command was executed, it does not know what your home directory is. The *log* command terminates the child shell and returns control to the parent script.

The *script* command can also be executed with options only. This form of the command also spawns a script program that interacts with you.

Finally, the *script* command can be executed with arguments or with both options and arguments. This form may be used, for example, to execute a *script* script for which you do not have execute permissions. Either of the following commands executes the file *script_filename*:

```
script script_filename
script <script_filename>
```

script first checks to see that the file specified as an argument is actually a file that contains commands. If it is not, *script* executes it only if you specify the "c" option (see Options).

ARGUMENTS

<argument_list> — A list of arguments to pass to the script command. Each element in the argument list consists of a command name followed by the appropriate arguments and options. The elements in the list must be separated by a valid command separator (";", "&", "&&", or "||"). If any separator characters are used, the entire argument list must be enclosed in single or double quotation marks.

<script_filename> — The name of a file containing commands to execute.

OPTIONS

Options specified to the script program must appear immediately after the name *script* on the command line, so that they are not confused with options that pertain to the arguments passed to the script.

a — Start execution with the *sabort* attribute off.

b — Ignore CTRL-C and CTRL-\.

c — Process the argument list as a command.

n — Run all background jobs with lower priority.

v — Start execution with the verbose attribute on.

x — On the next command, do not fork unless necessary. This option is used only when calling a script program from another program.

NOTE

It is impossible to specify a null string as an argument to a command because the script program removes null strings from the command line.

ERROR MESSAGES

`Built-in commands may not use pipes.`

Input to or output from the script built-in commands (*chd, dperm, jobs, log, login,* and *wait*) may not be routed through a pipe.

`Cannot execute <cmd_name>.`

The operating system was unable to execute the specified command. Either the command does not exist or you do not have execute permission.

`Cannot initialize tables.`

This error, which should not occur, is usually indicative of a hardware failure. If it does occur, contact your Tektronix field office.

`Cannot open I/O redirection file.`

The operating system returned an error when the script program tried to open the file specified for I/O redirection. Most probably, the path specified cannot be followed (one of the directories does not exist) or you do not have the permissions necessary for opening the file. This message is preceded by an interpretation of the error produced by the operating system.

`Cannot open pipe.`

The operating system returned an error when the script program tried to open the specified pipe. This message is preceded by an interpretation of the error produced by the operating system.

`Error opening a file.`

The operating system returned an error when the script program tried to open the specified file. This message is preceded by an interpretation of the error produced by the operating system.

`Error reading a file.`

The operating system returned an error when the script program tried to read the specified file. This message is preceded by an interpretation of the error produced by the operating system.

`Error writing a file.`

The operating system returned an error when the script program tried to write to the specified file. This message is preceded by an interpretation of the error produced by the operating system.

```
I/O redirection conflict.
```

You tried to redirect standard input, standard output, or standard error to more than one place.

```
I/O redirection error.
```

The operating system returned an error when the script program tried to perform the specified I/O redirection. This message is preceded by an interpretation of the error produced by the operating system.

```
Memory overflow.
```

There is not enough memory available to perform the specified command. Most probably, the expansion of the matching characters used on the command line, for which many matches were possible, caused the error.

```
Missing ] or invalid character range.
```

Either the right square bracket is missing from the specification of a range of matching characters, or the range specified is invalid.

```
No matching file names found.
```

Matching characters appear on the command line, but no file names match the specified pattern.

```
Parenthesis usage error.
```

The parentheses used on the command line are unbalanced.

```
Too many tasks.
```

The script program tried to fork, but too many tasks were running at the time. The limit to the number of tasks allowed either to the individual user or to the operating system as a whole was reached.

```
Unknown error.
```

This error should not occur. If it does, contact your Tektronix field office.

```
Unrecognized argument to built-in command.
```

The argument specified is not a valid argument to the built-in command in question.

```
Unterminated string.
```

The quotation marks used on the command line are unbalanced.

SEE ALSO

chd
dperm
env
jobs
log
login
shell
time
wait

set

Change or display the current state shell and values of the environment variables. This is a shell command.

SYNTAX

```
set [<file_name>]
```

ARGUMENTS

<file_name>	The name of source file from which to read commands.

DESCRIPTION

The *set* command, which is part of the shell program, displays the current state of the shell and the values of the environment variables if no arguments are given. If a file argument is specified then the commands in it are executed as if they were typed from the keyboard.

SEE ALSO

unset
shell

shell

DESCRIPTION

Shell is an interactive command language that gives you many conveniences when working with the 4400 operating system. When using *shell* as the command interpreter, you can perform command line editing.

EDITING AND HISTORY

Shell remembers a limited number of commands. You can use the shell command *history* to retrieve a list of commands that *shell* accepted. You can then use control (or function) keys to recall and modify commands.

You enter commands one character at a time, editing the command line (either with backspace and re-typing or with the command editor) and press the return key to execute the command.

Table 2-3, Shell Editing Keys And Functions, shows the keys or key sequences associated with the *shell* editing functions and a brief description of those functions.

When editing, the characters you insert appear at the cursor position and the characters following the cursor shift to the right.

The editing function most commonly used with *history* is *up*. If you do not have the cursor positioned at the start of a line, successive calls to *up* recall only commands that begin with the same non-blank character string as that preceding the cursor. For example, if you have the cursor after the string *chd* (where you had used the *chd* command earlier) pressing ^P takes you back to the previous command where you used *chd*. The *history* command can only recall a limited number of past commands.

Table 2-3
SHELL EDITING KEYS AND FUNCTIONS

Key	Function	Description
^P	up	Recalls the previous command with the same prefix.
^N	down	Recalls the next command with the same prefix.
^F	right character	Moves the cursor right one character.
^B	left character	Moves the cursor left one character.
^D	erase character	Erases the character at the cursor.
^H or DEL	backspace	Erases the character preceding the cursor.
ESC-F	word right	Moves the cursor to the right to the start of the next word.
ESC-B	word left	Moves the cursor to the left to the start of the nearest word.
ESC-D	erase word	Erases to the end of the word at or following the cursor.
ESC-H or ^W	erase back word	Erases the word before the cursor.
^A	begin line	Moves the cursor to the beginning of the line.
^E	end line	Move the cursor to the end of the line.
^K	erase to end	Erase characters from the cursor to the end of the line.
^U	erase line	Erase (or restore) the entire line.
^T	transpose	Transpose the previous two characters.
^L	redisplay	Redisplay the current line.
^Q	quote	Enters the key value of the following key.
RET or LF	return	Executes the command.

ENVIRONMENT VARIABLES

A list of name-value pairs called *environment variables* is kept by *shell*. When *shell* encounters a string that it recognizes as an environment variable, it emits the value it has stored for that variable. You may define or modify an environment variable by writing a quoted string of the form: *name=value* to *shell*. For example to define the variable COMMAND as /bin, type the string *COMMAND=/bin*. Then, to change your working directory to /bin, type *chd $COMMAND*.

You can delete environment variables with *unset*, used as *unset COMMAND*. The *set* command displays the currently listed environment variables.

SEARCH PATH

The environment variable *PATH* defines the search path for the directory containing the command. Each alternative directory name is separated by a colon. If the command name contains a /, the search path is not used. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission, but is not a binary file, it is assumed to be a file containing shell commands. A subshell (i.e., a separate process) is spawned and the program *script* is used to read and execute it. A command contained within parentheses is also executed in a subshell.

ALIASES

Shell maintains a list of aliases, or command redefinitions. When you enter a command line, *shell* checks the first word of the command to see if it is an alias. If so, *shell* executes the text of the alias and can use argument designators to extract the arguments to the aliased command.

You can create or modify an alias with the *alias* command. You can delete an alias with the *unalias* command. You can see the currently defined aliases by entering the *alias* command without any arguments.

For example, if a Unix<tm> programmer were to want the command *ll* to perform the action of the operating system command *dir +l*, that person could create that alias by typing:

```
alias ll 'dir +l $*'
```

Now typing *ll /bin* will have the same effect as typing *dir +l /bin*.

VARIABLE ARGUMENTS

Variables may contain argument designators to extract arguments from commands (such as used when defining aliases). The argument designators are:

$0	The first word of the command (the command itself)
$n	The nth argument of the command
$^	The first argument of the command (equivalent to $1)
$$	The last argument of the command
$x-y	The range of arguments from x to y (such as $3-5)
$-y	Abbreviation of $0-y
$*	Abbreviation of $^-$ ($1 $2 ...$$)
$n*	Abbreviation of $n-$
$n-	Abbreviation of $n-($-1) (omits last argument)
$-	Abbreviation of $0-($-1) (omits last argument)

When evaluating aliases, these argument designators extract the arguments from the command line to pass to the aliased commands.

FUNCTION KEYS

The function keys and joydisk are represented by special environment variables. By defining these variables, you can cause the joydisk and function keys to perform actions. When you press a function key or the joydisk, *shell* echoes the string defined for that variable.

You can insert special characters into function key and joydisk variable definitions by using the quote character, CTRL-Q. The special-character following a CTRL-Q is stored literally.

The 24 function key variables are $f1 - $f12 and $F1 - $F12. The joydisk variables are $joyup, $joydown, $joyleft, $joyright $JOYUP, $JOYDOWN, $JOYLEFT, and $JOYRIGHT. The *Break* key is bound to the variable $BREAK, and the arrow key (upper right of keyboard) is bound to *arrow* and *ARROW* for the shifted arrow key.

COMMAND SYNTAX

A *command* is either a simple-command or a list.

A *simple-command* is a sequence of non blank words separated by blanks (a blank is a tab or a space). The first word specifies the name of the command to be executed. Except as later specified, the remaining words are passed as arguments to the invoked command. (The command name is passed as argument 0.)

A *pipeline* is a sequence of one or more commands separated by "|". The standard output of each command but the last is connected by a pipe to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate.

A *list* is a sequence of one or more pipelines separated by the characters ";" or "&", and optionally terminated by them. The characters ";" and "&" have equal precedence. A semicolon causes sequential execution; an ampersand causes the preceding pipeline to be executed without waiting for it to finish. Newlines may appear in a list, instead of semicolons, to delimit commands.

COMMAND SUBSTITUTION

The standard output from a command enclosed in a pair of back quotes (` `) may be used as part or all of a command word; trailing newlines are removed.

Wild

Following substitution, each command word is scanned for the characters "*", "?" and "[". If one of these characters appears, the command word is regarded as a pattern. The command word is replaced with alphabetically sorted file names that match the pattern. If no file name is found that matches the pattern, the command word is left unchanged. The character "." at the start of a file name or immediately following a "/", and the character "/", must be matched explicitly.

The special characters match in this manner:

*	Matches any string, including the null string.
?	Matches any single character.
[...]	Matches any one of the characters enclosed. A pair of characters separated by "-" matches any character lexically between the pair.
~	Expands by replacing the tilde with the home directory of the named user. This is valid only if the tilde is the first charaqcter. For example, if user sandra has a home directory (defined in the password file) of */public/sandra*, the filename *~sandra/file* expands to */public/sandra/file*.

An asterisk in a command line matches any character or characters, including the null string but not including a leading period. Thus, the command

```
list *.bak
```

lists all files in the working directory whose names end in *.bak* and do not begin with a period.

The question mark matches any single character except the null character or a leading period. For example, the command

```
list chapter_?
```

lists all files whose names begin with the string *chapter_* and end with a single character.

You can use more than one matching character at a time. For instance, the command

```
list *.?
```

lists all files in the working directory whose names end with a period followed by a single character (except those whose names begin with a period).

Square brackets allow you to specify a set of characters to use in the matching process. The set of characters is defined by listing individual characters or by specifying two characters separated by a hyphen. In the former case, the script program looks for all file names which use any one of the enclosed characters in the appropriate place. In the latter, the two characters specify a class of characters containing the two characters themselves and any characters which lexically fall between them in the ASCII character set. For example, if your working directory contains nine files named *chapter1*, *chapter2*, *chapter3*, and so forth, the following command lists the first three chapters, the fifth chapter, and the last three chapters:

```
list chapter[1-357-9]
```

If the shell program cannot find a match for any of the arguments containing matching characters, it aborts the command. If it finds a match for at least one argument containing matching characters, it ignores any other arguments containing matching characters for which it cannot find a match.

If a filename actually contains one of the matching characters or either a space or a comma, you must enclose the name in single or double quotation marks. In such a case the shell program passes the arguments to the command without performing any character matching.

QUOTING

The following characters have a special meaning to the shell and cause termination of a command word unless quoted.

";" "&" "(" ")" newline space tab

A character may be quoted by preceding it with a \. \newline is ignored. All characters enclosed between a pair of single quote marks ', except a single quote, are quoted. Inside double quotes " " parameter and command substitution occurs and \ quotes the characters \, ', ", and $.

EXECUTION

Each time a command is executed, the above substitutions are carried out.

You can run commands in the background by inserting a "&" as either the first or last nonblank character on a command line. *shell* prints the name and process ID for each background task when it begins, and again when it terminates.

You can group commands for a subshell with parentheses, put the subshell in the background by following the closing parentheses with "&", and redirect I/O for the subshell.

You can time execution of a command by using "%" as the first or last nonblank character on a command line. *shell* prints the real, user, and system times for the command's execution when the command ends.

To quickly access the program, *script,* use "!" as the first non-blank character on a line. To pass the remaining characters to *script* uninterpreted, use the +c option.

REDIRECTING INPUT AND OUTPUT AND ERROR

To redirect standard output, use ">" and ">>". ">" directs standard output of a preceding command into the filename following it, writing over an old file. ">>" appends the standard output of a preceding command into the filename following it.

To redirect standard input into a command, follow the command with "<" in front of the command that will generate the input for the first command.

To redirect standard error, use "^", and "^^" as you would standard output redirection. You can combine redirection of standard input, output, and error to a file by using a combination of symbols. For example you can redirect both standard error and output to the file *temp* with ^>*temp*. You can also connect both standard output and error to a pipe with "^|".

Table 2-4
I/O Redirection

Symbol	Meaning
<	Take standard input from file following symbol.
>	Send standard output to file following symbol.
^ or %	Send standard error to file following symbol.
^^	Append standard output to file following symbol.
|	Connect programs so output of one becomes input of next.

SUMMARY OF *shell* COMMANDS

Table 2-5 lists the commands (followed by a brief description) that are part of *shell.* You cannot redirect I/O for these commands.

Table 2-5
shell COMMANDS

Command [arguments]	Description
addpath [dir_name_list]	Add the named directories to the search path of the shell.
alias [name][string]	With no arguments, prints the names of all defined aliases. With one argument, prints the associated alias. With two arguments, the second argument is defined to be an alias for the first.
chd [arg]	Change current directory (default to user's home directory)
dirs	Lists the current working directory and the directory stack.
dperm [u-rwx][o-rwx]	Sets default permissions for file creation.
exit	Terminate a subshell.
history	Displays saved command history.
jobs	Lists currently executing background jobs for present user.
login [arg]	Terminate this interactive session and start the login process.
logout	Terminate this interactive session.
popd	Changes the working directory to the one whose name is on the top of the directory stack.
pushd [dir]	Pushes the name of the working directory on the directory stack and changes to the specified directory. With no argument, this command exchanges the top of the directory stack and the current working directory.
rmpath [dir_name_list]	Remove the named directories from the search path of the shell.
set [file]	Without an argument, *set* displays the current state of the shell and the values of the defined environment variables. If you specify a file, it executes the commands in it as if you had typed them. Use this option to set environment variables and the user file creation mask. *set* terminates an input line and cannot be used as an alias.
unalias [name] [+a]	Deletes the named alias from the set of aliases. Use the option *+a* to remove all aliases.
unset [name] [+a]	Removes named environment variables declared by *set*. Use the option *+a* to remove all environment variables.
wait	Waits for all background processes to terminate and reports their termination status. If the *wait* command is interrupted, then a list of currently active processes is displayed.

SYNTAX

```
shell [+lx][+h=<file_name>] [<file_name>] [+c <string>]
      [+i<file_name>]
```

DESCRIPTION OF THE shell"

If you call *shell* with no arguments, it spawns a subshell with which you then interact until you issue either the *exit* or *logout* commands. This shell executes commands in the file *of your home directory. When you exit the subshell, control returns to the parent shell.*

OPTIONS

l	The *l* option tells *shell* to run as a login shell. This option causes shell to execute commands from the files *.login* and *.shellbegin* (in your home directory) when it begins execution, and from the file *.logout* when it terminates. The *exit* command terminates a subshell, use *logout* to end a session with the login shell.
h=<file_name>	This option causes *shell* to initialize its state from that saved in <filename>. When *shell* terminates it saves its history, environment variables, and aliases into this file. Without this option, *shell* reads and writes its state into the file *.shellhistory* in your home directory. To prevent state recovery and saving, use *none* as the <filename> (+h=none).
<file_name>	If *shell* is followed by a filename without the *c* or *i* options, it assumes that the file is a command script. *shell* passes control and the argument to the program, *script.*
c <string>	The *c* option causes *shell* to assume the next string of characters is a shell command, to execute that command and then terminate.
i <file_name>	The *i* option causes *shell* to process the commands contained in <filename> and then terminate, rather than passing the commands to *script.*
x	On the next command, do not fork unless necessary. This option is used only when calling a script program from another program.

DIAGNOSTICS

Shell gives error messages similar to other messages detailed in this manual whenever directories and files cannot be opened, whenever it detects a syntax error, and when it reaches its memory limits.

LIMITS

Shell has the following limits:

- 256 environment variables
- 30 saved commands (history)
- 16 entries on the directory stack
- 128 characters per command line
- Command expansion cannot exceed 512 arguments and 5120 characters

SEE ALSO

addpath
alias
chd
dirs
dperm
exit
history
jobs
login
logout
popd
pushd
rmpath
script
set
unalias
unset
wait

status

Display the status of running tasks.

SYNTAX

```
status [+alsx] [+w[=<num>]]
```

DESCRIPTION

The *status* command reports, to standard output, the status of tasks running on the system. By default, this report does not include shell or login programs and is restricted to tasks belonging to the user who executes the command. The command is not always completely accurate due to the dynamic nature of the operating system. By default, the *status* command reports on the following parameters:

Task-id	The number assigned to the task by the operating system.
Mode	Indicates whether the task is in memory ("c") or has been swapped to the disk ("s").
tty	The number of the terminal from which the task originated. An *xx* in the field indicates that no terminal is associated with the task.
Prio	If the entry in this field is a number, it indicates the priority of the task. A higher number indicates a higher priority. Non-numeric priorities are described in Table 2-6, Possible Task Priorities.

Table 2-6
POSSIBLE TASK PRIORITIES

Priority	Meaning
buf	Waiting for a system buffer.
disk	Waiting for some disk activity.
file	Waiting for some file activity.
halt	Halted by another task.
in	Waiting for input from the terminal.
out	Waiting for output to the terminal to end.
pipe	Waiting for pipe data (usually input).
upd	Updating an fdn.
slp	Sleeping (not executing).
swap	Being swapped to or from the disk.
sys<	Highest possible priority.
wait	Waiting for another task to end.

Time	If the command is *System*, this parameter is the amount of unused CPU time since the system was booted. Otherwise, it is the total CPU time that a particular task has used.
Command	The command which originated the task. By default, the *status* command shows the first thirty-five characters of the command line; the rest are truncated. The command *System* is the operating system. The command */etc/init* executes the login program. If the *status* command cannot determine what was on the command line, this field contains the entry "???".

OPTIONS

a	List all tasks on the system, not just those belonging to the user.
l	Produce a more detailed description of the status of each task.
s	Produce a statistical summary of the use of the operating system.
w[=<num>]	Wait <num> seconds after reporting the status; then produce another report. The command repeats 100 times. The default is thirty seconds.
x	List every task (a normal listing does not include shell programs, the *System* command, or the command */etc/init*).

If the user specifies the "l" option, the following additional items are included in the report:

Status	The status of the task. Possible values include run (task is running), sleep (task is waiting for something to happen), and term (the task has terminated).
User	The user name of the person who owns the task. If two or more user names share the same user ID, *status* uses the name that appears first in the password file.
Parent	The task ID of the parent task. If the parent task in no longer active, the ID shown in this field is 1.
Size	The amount of memory that the task is using.
Res	A rough measure of the amount of time a task has been in memory or swapped out to the disk. Each unit represents four seconds. The largest number that is ever displayed is 255. This number is set to 0 whenever a task is swapped into or out of memory.

If the user specifies the "s" option, the following statistics are included in the report. They represent activity on the system since the time the system was booted.

```
Total block I/O transfer attempts.
```

The number of times the system has tried to access a disk block in the cache.

```
Total disk I/O operations.
```

The number of times the system has had to access the disk. This statistic does not include swap operations.

```
Total blocks freed.
```

The number of blocks that have been released from a file to the free list. If the same block has been released more than once, each release is counted.

```
total system calls
```

The number of times the system has executed a system call.

```
total PAGE IN operations
```

The number of times the system has read a page from the swap device.

```
total PAGE OUT operations
```

The number of times the system has written a page to the swap device.

```
total pages stolen
```

The number of times that the system had to take memory from one user to give to another.

EXAMPLES

```
status +s
```

Displays the default information about the status of all tasks except shell programs that belong to the user. A summary of the use of the operating system is included in the output.

```
status +alxw=15
```

Displays detailed information about the status of all tasks on the system. It waits fifteen seconds, then issues another report. The command repeats 100 times unless the user interrupts it by typing a CTRL-C.

stop

Stop the system and prepare to shut off the power or reset.

SYNTAX

```
stop
```

DESCRIPTION

The command *stop* terminates any background processes, closes open files, flushes buffers to the disk, and does the general housekeeping necessary to perform an orderly system shut-down.

You should always run *stop* before turning off the power to the Tek 4400 series system, or pressing the Reset Button.

EXAMPLES

```
stop
```

This is the only form of this command.

MESSAGES

When *stop* is finished, it prints the message:

At this point, the system has been completely shut down and it is safe to turn off the power or to reset the system.

strip

Remove the symbol table from an executable binary file.

SYNTAX

```
strip <file_name_list>
```

DESCRIPTION

The *strip* command removes the symbol table from an executable binary file. This reduces the size of the file.

ARGUMENTS

<file_name_list> A list of files to process.

EXAMPLES

```
strip testprog
```

This example removes the symbol table from the executable binary file *testprog*.

ERROR MESSAGES

```
Error creating <file_name>: <reason>
```

The operating system returned an error when *strip* tried to create the specified file. This message is followed by an interpretation of the error returned by the operating system.

```
Error opening <file_name>: <reason>
```

The operating system returned an error when *strip* tried to open the specified file. This message is followed by an interpretation of the error returned by the operating system.

```
Error unlinking <file_name>: <reason>
```

The operating system returned an error when *strip* tried to unlink the specified file. This message is followed by an interpretation of the error returned by the operating system.

```
File <file_name> cannot be located.
```

The specified file does not exist.

```
File <file_name> is a device or a directory.
```

The specified file is not a regular file.

tail

Print a specifiable number (default is 250) of characters from the end of a text file.

SYNTAX

```
tail <file_name> <n>
```

DESCRIPTION

This utility prints the last *n* characters in a text file. If *n* characters from the end of the file happens to fall in the middle of a line, the line will be preceded by ... to show that only a part of the line has been printed. Whole lines are printed as they appear in the file.

Special characters such as carriage returns and tabs are counted as part of the "n" characters.

ARGUMENTS

<file_name>	The file from which characters are to be printed.
<n>	The number of characters from the end to start printing. The default is 250 characters. If "n" exceeds the number of characters in the file, the whole file is printed.

EXAMPLES

```
tail .shellbegin
```

Prints the last 250 characters of *characters*.

```
tail testfile 30
```

Prints the last 30 characters from the file *testfile*.

SEE ALSO

```
list
```

touch

Set the time of the last modification of a file to the current date and time.

SYNTAX

```
touch <file_name_list>
```

DESCRIPTION

The *touch* command sets the time of last modification for the specified file to the current date and time. The user must have read and write permission in a file in order to *touch* it. This command is often used in conjunction with the *update* command. It is also useful for correcting the last modification time of a file which was created or modified when the system time was incorrect.

ARGUMENTS

<file_name_list> A list of the names of the files to modify.

EXAMPLES

```
touch letter memo
```

Changes the modification time of the *letter* and *memo* files to the current date and time.

ERROR MESSAGES

```
Error seeking to beginning of file <file_name>: <reason>
```

The operating system returned an error when *touch* tried to seek to the beginning of <file_name>. This message is followed by an interpretation of the error returned by the operating system.

```
Error touching <file_name>: <reason>
```

The operating system returned an error when *touch* tried to change the last modification time of <file_name>. This message is followed by an interpretation of the error returned by the operating system.

```
File <file_name> does not exist!
```

The *touch* command could not find <file_name> in the file system.

SEE ALSO

create
date
update

unalias

The named aliases are deleted from the set of aliases. This is a shell command.

SYNTAX

```
unalias [<alias_name>] [+a]
```

DESCRIPTION

The *unalias* command is from the shell program and deletes named aliases from the set of aliases.

ARGUMENTS

<alias_name>	The name of the alias to delete.

OPTIONS

a	Remove all of the defined aliases.

EXAMPLE

```
unalias cp
```

This example deletes the alias *cp*.

SEE ALSO

alias
shell

unset

Delete the named variables from the set of environment variables. This is a shell command.

SYNTAX

```
unset [<env_name>] [+a]
```

DESCRIPTION

The named variables are deleted from the set of environment variables.

ARGUMENTS

<env_name>	The name of the environment variable to delete.

OPTIONS

a	Remove all environment variables. This will almost certainly make the current shell unusable.

EXAMPLE

```
unset TERM
```

This example deletes the environment variable *TERM*.

SEE ALSO

set
shell

update

Process a set of files, performing the specified operation on each file if it is newer than the file it is compared to.

SYNTAX

```
update [<makefile_name>] [+q]
```

OPTIONS

q	Do not send informative messages to standard output.

DESCRIPTION

The *update* command reads the specified *makefile*, which must conform to a special format, and conditionally performs the command or commands in that file. By default, the *update* sends informative messages to standard output telling the user what it is doing. The command is most often used to recompile programs whose sources have been updated.

ARGUMENTS

<makefile_name>	The name of the file to read instructions from. This file must be in a special format (see FORMAT OF THE "MAKEFILE"). The default is the file, *makefile*, in the working directory. If a makefile is specified, any following arguments are passed in as $1, $2, etc.

FORMAT OF THE *MAKEFILE*

The *makefile* is composed of modules, each of which is terminated with a percent sign, "%", in column 1. A module itself is composed of up to two parts. The first part specifies the process that *update* is to perform. The format for this first part is as follows:

```
<item-one>::<item_two>;<command_sequence>
```

Where <item_one> and <item_two> are the names of files; "::" is the *is newer than* operator; and the semicolon, ";", separates the file names from the command sequence.

The command sequence is composed of one or more operating system commands. The *update* command replaces any sequence of more than one space character with a single space. Multiple commands are separated by additional semicolons. If the commands do not fit on one line, the user must begin and end the sequence with an exclamation point, "!", which serves to delimit the entire command sequence. If the first portion of the module uses more than one line, the second exclamation point marks the boundary between the first and second portions of the module. The command sequence is executed if <item_1> is newer than <item_2>.

The user may substitute an ampersand, "&", for any character or sequence of characters in <item_one>, <item_two>, or the command sequence. In such a case the *update* command substitutes for all ampersands the strings specified in the second portion of the module. If the second portion of the file is absent, no command sequence is performed. This portion consists of one or more lines, each of which contains a single string to substitute for the ampersands. The *update* command replaces each occurrence of an ampersand with the string on the first line of the second portion of the module and performs the command sequence if <item_one> is newer than <item_2>. It then replaces all ampersands with the string from the second line, continuing in this fashion until it reaches the end of the second portion of the module (marked by a percent sign in column 1).

If the file represented by <item_two> does not exist, *update* considers that <item_one> is newer than <item_two>. If the file represented by <item_one> does not exist, or if neither the file represented by <item_one> nor <item_two> exists, <item_one> is not considered to be newer than <item_two>.

For instance, consider the following *makefile*:

```
&:&.b;asm & +sly
file_1
file_2
    .
    .
    .
file_n
%
```

An *update* command which references this file makes the following translation:

If *file_1* is newer than *file_1.b*, execute the command *asm file_1 +sly*.

If *file_2* is newer than *file_2.b*, execute the command *asm file_2 +sly*.

It continues in this fashion until *file_n* is processed. The percent sign in column 1 marks the end of the module, and because it is the only module in the file, the update command terminates.

More than one set of commands can be processed with a single *makefile* if the user includes more than one module in the file.

NOTES

- The *chd* command has no effect in a *makefile*.
- The *update* command always tries to substitute the strings strings specified in the second portion of a module for all ampersands which appear in the first portion. Thus, the command sequence itself cannot contain an ampersand. Consequently, tasks specified in a *makefile* cannot be executed in the background (although the *update* command itself may be sent to the background).

ERROR MESSAGES

```
*** Can't access Makefile <file_name> aborted!
```

The operating system returned an error when *update* tried to open <file_name> for reading. Most probably, the file specification is incorrect, the file does not exist, or the user does not have read permission for the file.

```
*** Error: Command too long.
<command_sequence>
```

After substitution for the ampersands has taken place, the command sequence is too long (the limit is 512 characters).

```
*** Error: Pattern too long.
<command_sequence>
```

The pattern for the command sequence (before substitution for ampersands takes place) is too long (the limit is 512 characters).

```
syntax error Makefile aborted
```

The *update* command was unable to interpret the *makefile*.

```
Syntax: update [<makefile_name>] [+q]
```

The *update* command requires exactly one argument. This message indicates that the argument count is wrong.

```
Unknown option: <char>
```

The option specified by <char> is not a valid option to the *update* command.

SEE ALSO

touch

wait

Wait for a background task to complete before accepting any more input.

SYNTAX

```
wait [<task_ID>]
wait any
```

DESCRIPTION

The *wait* command, which is part of the shell program, tells the shell program not to accept any more commands until the specified background task is complete. The termination status of the task is reported when the task is complete. If the user does not specify a task ID, the shell program waits for all active background tasks that are children of the shell program that issued the *wait* command to finish before accepting any new commands. The user may interrupt the *wait* command with a control-C.

ARGUMENTS

<task_ID>	The ID of the task to wait for. The shell program reports the ID when it sends a task to the background. The ID may also be obtained by executing either the *jobs* or the *status* command.
any	If the user specifies the argument *any*, the shell program waits for any one background task that is one of its children to finish before accepting a new command.

EXAMPLES

```
wait 495
```

Tells the shell program to accept no further commands until task 495 is complete.

```
wait
```

Tells the shell program to accept no further commands from the user until all background tasks belonging to that shell program are complete.

```
wait any
```

Tells the shell program to accept no further commands from the user until one background task belonging to that shell is complete.

ERROR MESSAGES

```
No tasks running in the background.
```

The shell program has no tasks running in the background.

```
Specified task not running in the background.
```

The task specified either is not a child of the current shell program or does not exist.

SEE ALSO

jobs
script
shell
status

Section 3
SYSTEM UTILITIES

The system utilities are generally reserved for the user logged in as *system*. They tend to be either powerful utilities, with great potential for misuse, or utilities that should be reserved to a limited number of users where many accounts are set up.

User *system* generally has the directory */etc* defined in the search path, and needs only enter the name of the utility to invoke it. The full path name is given here, however, to emphasize the special purpose of these utilities.

SYSTEM UTILITY DESCRIPTIONS

Descriptions of the system utilitys are contained in the following pages of this section. System Utilities are sumarized in Table 3-1.

Table 3-1
System Utilities

Name	Function
adduser	add a new user to the system
badblocks	remove bad disk blocks from the free list on the specified device
blockcheck	check the integrity of the allocation of all blocks used in files and of the free list of the specified device
deluser	remove a user from the system
devcheck	check a device for I/O errors
diskrepair	check and, optionally, repair inconsistancies in the logical structure of the disk
fdncheck	check the integrity of the structure of the file descriptor nodes (fdns) on the specified disk
makdev	create a special type of file, representing a device
mount	insert a block device at a node of the directory tree structure
owner	change the owner of a file
unmount	unmount a previously mounted device from the file system

adduser

Add a new user to the system.

SYNTAX

```
/etc/adduser <user_name>
```

DESCRIPTION

The *adduser* command is used to add a new user to the system. The specified user name must be unique to the system. It must be between one and eight letters long. All letters must be lowercase. Only the *system* user may invoke this command.

The *adduser* command performs the following tasks:

1. Adds the new name to the end of the password file, */etc/log/password.*
2. Assigns a user ID to the user.
3. Creates a home directory owned by the new user with *rwxr-x* permissions. The name of this directory is */<user_name>*.
4. Copies the file default *.shellbegin* file into the user's home directory and creates empty *.login* and *.shellhistory* files.

The *system* user or the new user should use the *password* command to ensure protection of the new user's files.

ARGUMENTS

<user_name>	A unique name assigned to the new user for use in response to the login prompt.

EXAMPLES

```
/etc/adduser chris
```

This example adds the user name *chris* to the bottom of the file */etc/log/password*, assigns a user ID, and creates the directory */chris*, which is owned by *chris*. The permissions in this directory for the owner is read, write and execute, while for others it is read and execute (*rwxr-x*).

ERROR MESSAGES

```
Error adding <user_name> to password file: <reason>
```

The operating system returned an error when *adduser* tried to add <user_name> to the password file. This message is followed by an interpretation of the error returned by the operating system.

```
Error assigning owner to /<user_name>: <reason>
```

The operating system returned an error when *adduser* tried to make the specified user the owner of the file /*<user_name>*. This message is followed by an interpretation of the error returned by the operating system.

```
Error creating /<user_name>: <reason>
```

The operating system returned an error when *adduser* tried to create the file /*<user_name>*. This message is followed by an interpretation of the error returned by the operating system.

```
Error creating . file: <reason>
```

The operating system returned an error when *adduser* tried to create the file .. This message is followed by an interpretation of the error returned by the operating system.

```
Error creating .. file: <reason>
```

The operating system returned an error when *adduser* tried to create the file ... This message is followed by an interpretation of the error returned by the operating system.

```
Name must be 1 to 8 lowercase letters.
```

The specified user name must be between one and eight letters long. All letters must be lowercase.

```
Syntax: /etc/adduser <user_name>
```

The *adduser* command expects exactly one argument. This message indicates that the argument count is wrong.

```
The name <user_name> is already in use.
```

The specified user name must be unique to the system.

```
You must be system manager to run adduser.
```

Only the *system* user may execute the *adduser* command.

SEE ALSO

deluser
password
perms

badblocks

Removes bad disk blocks from the free list on the specified device.

SYNTAX

```
/etc/badblocks <dev_name> <block_number> [options]
```

DESCRIPTION

Removes bad disk blocks from the free list on the specified device. The bad block information is recorded in the file */.badblocks*. Once the bad-block information is recorded, the *diskrepair* utility is run to check the file system integrity. Bad blocks are identified by the *devcheck* utility, whichs reports the bad-blocks by HEX block number, *badblocks* expects the bad-block number to be in decimal radix, be warned! Hard-disks utilize the controller option to mask out bad-blocks so the */.badblocks* file is initially empty. Should blocks become defective they are masked out by software via the *badblocks* utility. Total system reformat and rebuild will utilize the controller option to mask out bad-blocks.

ARGUMENTS

<dev_name>	The name of the device to check, must be a block device.
<block_number>	The number of the bad block in decimal radix! If a diskette contains one or more bad blocks it should be discarded.

OPTIONS

This utility has the same options as *diskrepair*.

SEE ALSO

diskrepair

blockcheck

Check the integrity of the allocation of all blocks used in files and of the free list on the specified device.

SYNTAX

```
/etc/blockcheck <dev_name>
```

DESCRIPTION

blockcheck checks the integrity of the block allocation used in the files and free list on the specified device. It locates problems such as duplicate blocks, missing blocks, and invalid block addresses.

This command is primarily intended for use by the *diskrepair* utility, which calls it. It may also be used on its own as a diagnostic utility; however, *blockcheck* can only check the disk; it cannot repair it. If *blockheck's* output suggests that the disk is damaged, use *diskrepair* on the disk.

You should only use *blockcheck* if no other tasks are active on the system; otherwise, the results are unpredictable.

ARGUMENTS

<dev_name>	The name of the device to check. It must be a block device.

EXAMPLES

```
/etc/blockcheck /dev/floppy
```

This example checks the integrity of the the allocation of blocks on the floppy disk.

SEE ALSO

devcheck
diskrepair
fdncheck

deluser

Remove a user from the system.

SYNTAX

```
/etc/deluser <user_name> [+x]
```

DESCRIPTION

The *deluser* command removes the specified user from the system. It removes the corresponding entry from the file */etc/log/password* and by default destroys files and subdirectories in the user's home directory that are owned by that user. It also deletes the home directory itself if it is empty after all the deletions are complete. Only the *system* user may execute this command.

ARGUMENTS

<user_name>	The name of the user to delete from the system.

OPTIONS

x	Delete the user, but do not delete the user's files from the system.

EXAMPLES

```
/etc/deluser chris
```

This example deletes the line containing the entry for the user name *chris* from the file */etc/log/password*. It also deletes all files and subdirectories in the directory */chris*, as well as that directory itself.

CAUTION

This command should be used with great care as it may recursively descend the user's directory tree, deleting all files within.

ERROR MESSAGES

```
Cannot delete a user with an ID of 0 or 1.
```

The *deluser* command cannot delete user ID 0 (system) or 1.

```
Cannot execute remove.
<user_name> not removed from system.
```

The *remove* command, which is called by *deluser* is not in */bin* or */etc*. The command aborts without editing the password file.

```
Name must be 1 to 8 lowercase letters.
```

The specified user name must be between one and eight letters long. All letters must be lowercase.

```
Syntax: /etc/deluser <user_name>
```

The *deluser* command expects exactly one argument. This message indicates that the argument count is wrong.

```
<user_name> is not in the password file.
```

The file */etc/log/password* does not contain an entry for the specified user name.

```
You must be system manager to run deluser.
```

Only the *system* user may execute the *deluser* command.

SEE ALSO

adduser
remove

devcheck

Check a device for I/O errors.

SYNTAX

```
/etc/devcheck <dev_name> [+fsv]
```

DESCRIPTION

The *devcheck* command checks the specified device for I/O errors. As it checks the device, it prints informative messages, which tell the user what part of the device is being checked. It always checks the boot sector and the system information record (SIR). By default, it also checks the fdn space, the swap space, and the volume space.

Every time it finds a bad block, it prints a message giving the address of the block in hexadecimal. When it is finished, *devcheck* prints a message reporting the total number of bad blocks on the disk.

If a floppy disk contains one or more bad blocks, it should probably be discarded. If a hard disk contains one or more bad blocks, it should be reformatted with the addresses of all bad blocks placed in the file *.badblocks*. It is wise to run this command immediately after formatting a disk.

ARGUMENTS

<dev_name>	The name of the device to check. It must be a block device.

OPTIONS

f	Check only the fdn space.
s	Check only the swap space.
v	Check only the volume space.

EXAMPLES

```
/etc/devcheck /dev/floppy
```

Checks the entire disk in the floppy drive for I/O errors.

```
/etc/devcheck /dev/floppy +v
```

Checks the boot sector, the SIR, and the volume space of the disk in the floppy drive for I/O errors.

MESSAGES

```
Bad blocks file too large - continuing without list.
```

Devcheck cannot read a *.badblocks* file that has more than 138 bad blocks in it. Currently, this theoretical limitation on the number of bad blocks is unlikely to present a practical limitation. The number of bad blocks on a disk should not even approach 138.

```
Can't open character device <dev_name>.
```

The *devcheck* command cannot open the character device which corresponds to the block device specified on the command line. Most probably, either the device does not exist or the user does not have the permissions necessary to open it. In such a case the command continues, but it may report the blocks in the file

```
Can't read .badblocks file - continuing without list.
```

The *devcheck* command encountered an I/O error when it tried to read the file *.badblocks.*

```
File .badblocks not found - continuing with check.
```

The device specified does not contain a file named *.badblocks*, or due to damage in the logical structure of the disk, *devcheck* cannot locate the file.

ERROR MESSAGES

```
Can't open <dev_name>.
```

The *devcheck* command cannot open the device specified on the command line. Most probably, either the device does not exist or the user does not have the permissions necessary to open it.

```
File <file_name> is not a block device.
```

The *devcheck* command can only check a block device.

```
Unknown option <char> - option ignored.
```

The option specified by <char> is not a valid option to the *devcheck* command.

SEE ALSO

blockcheck
diskrepair
fdncheck

diskrepair

Check and, optionally, repair inconsistencies in the logical structure of a disk.

SYNTAX

```
/etc/diskrepair <dev_name_list> [+bfmnpqruv]
```

DESCRIPTION

The *diskrepair* utility checks the structure of the disk or disks specified in <dev_name_list>. The structure of the disk refers to the layout of and the connections among files, directories, free space, swap space, and other information that makes up the file system. Any inconsistencies in the structure are reported and, optionally, repaired. *Diskrepair* does not check or repair media errors (I/O errors).

DEFINITIONS

- A *file descriptor node* (or fdn) is an area on the disk which contains all the information the system needs about a file. There should always be at least one fdn per file on the disk.
- A 4400 series directory entry is simply a file name and a pointer to the proper fdn. There may be multiple directory entries pointing to the same fdn (multiple names for the same file).
- Each pointer to an fdn is called a *link* to that file. If there is a file with no links, it is considered to be *unreferenced. Out-of-range* refers to a pointer to a disk block or to an fdn which is beyond the valid number of blocks or fdns for the disk being tested.

RELATED UTILITIES

While it is operating, *diskrepair* calls two other utilities— *blockcheck* and *fdncheck*, which are both located in the directory */etc*.

- *Blockcheck* is concerned with the allocation of blocks on the disk. It locates problems such as duplicate blocks, missing blocks, and invalid block addresses.
- *Fdncheck* is concerned with the directories on the disk. It locates problems such as unreferenced files, file names with invalid associated files, and so forth.

MAJOR MODES OF OPERATION

There are two major modes of operation: simple and verbose.

- The simple mode is selected by default; the verbose mode is selected by the "v" option.
- In the verbose mode *diskrepair* reports all detected errors. In the simple mode it reports only those errors which require the deletion of files or of directory entries.
- If executed in simple mode, *diskrepair* issues a message upon completion which informs the user whether or not the disk is in need of repair. By default, all detected errors are automatically repaired (if possible).

OPTIONS

Two options ("n" and "p") exist to alter the handling of errors.

- The "n" option instructs *diskrepair* not to repair any errors. The "p" option instructs *diskrepair* to prompt the user for permission to repair the errors it reports.
- In verbose mode the "p" option causes *diskrepair* to prompt the user regarding all errors. In the simple mode, the user is prompted only for those errors which require the deletion of files or of directory entries; all other errors are automatically repaired without prompting.

NOTE

Most repairs result in a loss of data. The user can generally infer which data have been lost from the messages displayed.

- When using the command in simple mode (without the 'v' option), the user need not understand what types of checks are made by *diskrepair*. The only decisions required are whether or not to delete the reported files. In verbose mode, much more information is given to the user.

While this discussion is not intended to give full details of this utility, the following list shows most of the inconsistencies in disk structure for which *diskrepair* checks. First, however, a few definitions are in order.

INCONSISTENCIES

Here is a partial list of inconsistencies that *diskrepair* checks for:

- Blocks duplicated in files or free list
- Out-of-range blocks or fdns
- Missing blocks
- Bad free list
- Unreferenced files
- Inactive fdns
- Unknown fdn type
- Incorrect link counts
- Incorrect free block or free fdn count
- Invalid sizes in System Information Record

UNREFERENCED FILES

These are handled in one of two ways:

1. An attempt is made to give the file a name by putting it into the directory *lost+found* in the root directory of the disk being tested. The name given to the file is of the form *file<fdn>*, where <fdn> represents the fdn number of the file.

   In order for this procedure to work, the directory *lost+found* must already exist on the disk being checked, and it must have room for the entry. The program *crdisk* creates this directory, but if for any reason it has been deleted, the user should recreate it before running *diskrepair*. The user must must also create empty slots for entries by creating a number of files and then deleting them.

2. If it is not possible to put the unreferenced file into the *lost+found* directory (because there is either no directory *lost+found* or no room in it), *diskrepair* deletes the file (or prompts for permission to delete it if "p" was specified).

FDN ERROR DATA

If an error is associated with an fdn, a display of pertinent data from that fdn is printed. The display includes the fdn number of the file, its size in bytes, its owner, the time of its last modification, and one of the following types:

b = block device
c = character device
d = directory
f = file
i = inactive
u = unknown

The *diskrepair* utility should generally be run only on an otherwise inactive system. It should never be run on an active disk. If the "n" option is not specified (the disk may be written to), *diskrepair* attempts to unmount the disk being tested. If the disk being tested is the system disk, and if a repair is made which requires writing to the System Information Record (block number 1), *diskrepair* stops the system upon completion and issues an appropriate message instructing the user to reboot the operating system. This procedure is necessary to prevent conflicts between the written data and similar data kept in memory.

OPTIONS

b — Instructs *diskrepair* to run only the *blockcheck* portion of the utility. This procedure is often considerably faster, but still provides a fairly complete assessment of the validity of the disk structure.

f — Instructs *diskrepair* to run only the *fdncheck* portion of the utility. This option is useful if a problem is suspected in the directory structure, but the result is by no means a thorough check of the structure of the disk.

m — The operating system maintains a list of blocks available for use called the free list. A missing block is any block in the volume space which is not a part of any file and is not in the free list. The existence of such blocks is a harmless error in the structure of the disk.

Diskrepair generally places missing blocks in the free list. The "m" option, however, instructs *diskrepair* not to rebuild the free list solely on account of missing blocks. This option reduces the time required for *diskrepair* to run if missing blocks are the only problem in the free list.

n — Tells *diskrepair* to report all errors but to make no attempt to fix them. Therefore, *diskrepair* opens the device for reading only. This option is useful for checking the structure of a disk without risking the loss of data during repairs.

p

If the user specifies the "p" option, *diskrepair* reports each error, followed by a prompt requesting permission for the proposed repair. All prompts require an answer of either "y" (*yes*) or "n" (*no*).

NOTE

Many repairs result in the loss of data. (You can generally infer what has been lost from the messages diskrepair *displays.) Judicious use of the "n" and "p" options not only allows you to assess the damage to the disk and decide which information you are willing to sacrifice during the repair process; it also gives you the opportunity to try to salvage the data before repairing the disk.*

q — Inhibits certain warnings and messages from *diskrepair*. Several conditions exist which, while not technically errors in disk structure, may cause problems. These conditions usually result in a warning message; the "q" option inhibits them.

r — By default, if *diskrepair* finds that the free list is in error, it rebuilds it. The "r" option instructs *diskrepair* to rebuild the free list whether or not it contains errors. This option is useful if the free list is known to be bad or if the user wants to reduce fragmentation within the list.

u — Generates a report on the block usage of the specified device. This report is printed at the end of the *diskrepair* operation, and contains statistics on the following: (1) the number of each type of file in the file system and the total number of files in the system; (2) the number of unused blocks and the number of used blocks, including a breakdown of how the used blocks are allocated; (3) the number of free fdns and the number of fdns in use.

v — *Diskrepair* operates in one of two modes: simple or verbose. Simple mode is selected by default; verbose mode is selected by the "v" option. In simple mode, *diskrepair* reports only those errors which require the deletion of either files or directory entries. In verbose mode, all errors are reported. In addition, informative messages are printed describing what phase *diskrepair* is performing.

In verbose mode the "p" option causes *diskrepair* to prompt for permission regarding all errors. In simple mode the user is prompted only for those errors which require the deletion of either files or directory entries; all other errors are automatically repaired without prompting.

EXAMPLES

```
/etc/diskrepair /dev/disk
```

Checks the logical structure of the system disk. By default, *diskrepair* tries to fix every error it encounters. These repairs may result in the loss of data from the disk.

```
/etc/diskrepair /dev/disk +n
```

Checks the logical structure of the system disk, and reports those errors which require the deletion of either files or directory entries, but performs no repairs.

```
/etc/diskrepair /dev/floppy +pv
```

Checks the logical structure of the disk in the floppy drive, reports all errors it finds and prompts for permission before making any repairs.

```
/etc/diskrepair /dev/floppy +ru
```

Checks the logical structure of the disk in the floppy drive. *Diskrepair* rebuilds the free list no matter what and prints a summary of block usage when finished.

```
/etc/diskrepair /dev/disk1 +mq
```

Checks the logical structure of the auxiliary disk, */dev/disk1*. It does not rebuild the free list solely on account of missing blocks; neither does it print the warnings and messages which result from problems not technically errors in the structure of the disk, but which may cause problems.

NOTES

Diskrepair cannot solve all the problems your disk may have. For example, it cannot fix physical media problems. As for problems with the logical structure of the disk, *diskrepair* can only repair an error if the damaged information is redundant — that is, if there is some way of determining what the information should be.

Diskrepair cannot, for instance, fix a badly damaged SIR; nor can it repair a disk if the root directory is severely damaged. It is therefore imperative that up-to-date backups of all important files be maintained.

ERROR MESSAGES

```
Blockcheck terminated abnormally.
```

Blockcheck received a program interrupt from the operating system. The user cannot determine the source of such an error; however, it is not indicative of a problem with either *diskrepair* or the device. *Diskrepair* should be rerun, for the problem may not recur.

```
Can't call /etc/blockcheck.
```

Diskrepair cannot read or execute the file */etc/blockcheck.*

```
Can't call /etc/fdncheck.
```

Diskrepair cannot read or execute the file */etc/fdncheck.*

```
Can't read System Information Record.
```

The SIR is so badly damaged physically that *diskrepair* cannot read it. The user may be able to salvage some information from the disk, but must eventually reformat it.

```
Can't stat root.
```

Diskrepair cannot read the fdn which describes the root directory. The user may be able to salvage some information from the disk, but must eventually reformat it.

```
Can't stat std. output.
```

Diskrepair cannot read the fdn of whatever file is opened as standard output. The user should rerun *diskrepair* with */dev/console* as standard output.

```
Conflicting options.
```

The options specified on the command line conflict with each other.

```
Device is busy.
```

Any alterations that *diskrepair* makes must be made when the disk is not in use. Therefore, *diskrepair* determines whether or not the specified disk is mounted, and, unless the user specifies the "n" option, it tries to unmount a mounted disk before proceeding. This error message means that either some user's working directory is on the specified disk or some task is accessing a file on that disk.

```
Disk needs repair!
```

The structure of the disk is not logically sound. The user should rerun *diskrepair* to correct the problems.

```
Error reading block <block_num>.

Error reading fdn <fdn_number> in block <block_num>.

Error writing block <block_num>.

Error writing fdn <fdn_num> in block <block_num>.
```

Diskrepair encountered a physical error on the disk. If either the "p" or "n" option is in effect, *diskrepair* prompts for permission to continue. If the user chooses to continue when the "n" option is not in effect, the results are entirely unpredictable. They depend on precisely which block is damaged. Continuing with *diskrepair* may cause further damage to the disk, but in some cases, it may be the desired course of action.

NOTE

The first time diskrepair *reports an I/O error, answer* no *to the offer to continue and immediately rerun* diskrepair. *It is possible, though unlikely, that the I/O error is a soft one and will not recur.*

```
Error updating SIR.  Disk is bad!
```

Diskrepair encountered an I/O error when it tried to make the necessary changes in the SIR. The user should try again to execute *diskrepair*. If the error persists, the user cannot salvage any of the data on the disk.

```
/etc/blockcheck is invalid.
```

The version of the *blockcheck* command is not the correct one.

```
/etc/fdncheck is invalid.
```

The version of the *fdncheck* command is not the correct one.

```
Fdncheck terminated abnormally.
```

Fdncheck received a program interrupt from the operating system. The user cannot determine the source of such an error; however, it is not indicative of a problem with either *diskrepair* or the device. *Diskrepair* should be rerun, for the problem may not recur.

```
Intentional system stop.  Reboot system.
```

If the SIR of the root device must be updated, *diskrepair* kills all tasks running on the system and locks up the system so that no new tasks can begin. It then modifies the SIR. This procedure is necessary to prevent conflicts between the written data and similar data kept in memory. After updating the SIR, *diskrepair* stops the system and prints this error message. The user must reboot the system before proceeding.

```
No device specified.
```

The user did not specify a device on the command line.

```
No such device.
```

The user specified a nonexistent device on the command line.

```
Not a block device.
```

Diskrepair can only operate on block devices.

```
Output directed to device under test.
```

When testing the structure of a disk, it is impractical to try to redirect the output (the results of the test) to a file on the disk being tested. The user should reexecute *diskrepair* without redirecting the output or redirecting it to a different, mounted device.

```
Permission denied.
```

A user who executes *diskrepair* without the "n" option must have both read and write permission on the specified device. A user who executes *diskrepair* with the "n" option needs only read permission.

```
Problems encountered.  Diskrepair should be rerun.
```

Diskrepair may encounter more problems than it can fix during one run. For example, it can only handle a certain number of duplicate or out-of-range blocks. If *diskrepair* cannot fix all the errors it encounters, or if it encounters an I/O error but continues operation, it prints this error message when it finishes.

```
Unknown option: <char>
```

The option specified by <char> is not a valid option to the *diskrepair* command.

```
Unmount error: <error_num>
```

Diskrepair encountered some problem other than a busy device when it tried to unmount the device. The accompanying error number is the number of the 4404 error that caused the failure. The user should consult the operating system manual for an explanation of the error.

SEE ALSO

blockcheck
fdncheck

fdncheck

Check the integrity of the structure of the file descriptor nodes (fdns) on the specified disk.

SYNTAX

```
/etc/fdncheck <dev_name>
```

DESCRIPTION

The *fdncheck* command checks the integrity of the structure of the file descriptor nodes (fdns) on the specified disk. An fdn contains all the information that the operating system needs to know about a file.

This information includes, but is not limited to, the type of file, the owner of the file, the size of the file, and the addresses of all the blocks that are a part of the file. The *fdncheck* command locates problems such as unreferenced files, directory entries with invalid associated files, and so forth.

This command is primarily intended for use by the *diskrepair* utility, which calls it. It may also be used on its own. However, *fdncheck* can only check the structure of the disk; it cannot repair it. If the output from the command suggests that the structure of the fdns is damaged, the user should execute *diskrepair* on the disk.

The *fdncheck* command should be executed only when no other tasks are active on the system. Otherwise, the results are unpredictable.

ARGUMENTS

<dev_name>	The name of the device to check. It must be a block device.

EXAMPLES

```
/etc/fdncheck /dev/floppy
```

Checks the structure of the fdns on the disk in the floppy drive.

makdev

Create a special type of file, representing a device.

SYNTAX

```
/etc/makdev <file_name> <dev_type> <maj_dev_num> <min_dev_num>
```

DESCRIPTION

The *makdev* command creates a special type of file which represents a device. This type of file allows the user to access the device drivers for the corresponding physical device. Only the *system* user may invoke this command.

The major device numbers are listed in Table 3-2.

Table 3-2
Major Device Numbers

Number	Device
0	Block or Console
1	Memory
2	Null
3	Floppy char
4	Disk char
5	Sound
6	Printer
7	Communication port
8	Tape char

ARGUMENTS

<file_name>	The name of the file to create. For a block device, the last component of the file name must consist of a string of letters followed by a string of digits. For a character device, the last component of the file name must consist of the same string of letters, followed by the letter "c", followed by the same string of digits.
<dev_type>	A letter designating whether the device is a block device, (b), or a character device, (c).
<maj_dev_num>	A number which tells the operating system which set of device drivers to use for the specified device.
<min_dev_num>	A number which tells the operating system which physical device to associate with <file_name>.

EXAMPLES

```
/etc/makdev /dev/floppy b 0 0
```

Creates a special file named */dev/floppy*, which represents a block device. Currently, all block devices have the same major device number, 0. The first four (beginning with 0) minor device numbers for this major device number designate floppy disk drives 0 through 3. Thus, this command tells the operating system to use the device driver for block devices and to associate the file with the floppy drive.

```
/etc/makdev /dev/floppyc c 3 0
```

Creates a special file named */dev/floppyc*, which represents the character device associated with the block device */dev/floppy*. The major device number for a character device associated with a floppy disk drive is 3. The first four (beginning with 0) minor device numbers for this major device number designate floppy disk drives 0 through 3. Thus, this command tells the operating system to use the device driver for a character device associated with a floppy disk drive and to associate the file with the floppy drive.

NOTES

- Every disk device requires both a block device and a corresponding character device in order to function properly.

ERROR MESSAGES

```
<char> is not a valid type of device.
```

The argument <dev_type> must be either "b", for a block device, or "c", for a character device.

```
Error creating <file_name>: <reason>
```

The operating system returned an error when *makdev* tried to create the special file <file_name>. This message is followed by an interpretation of the error returned by the operating system.

```
Invalid major device number: <num>
```

The number specified as the major device number is invalid.

```
Invalid minor device number: <num>
```

The number specified as the minor device number is invalid.

```
Syntax: /etc/makdev <file_name> <dev_type> <maj_dev_num> <min_dev
```

The *makdev* command expects exactly four arguments. The command line does not conform to the syntax.

```
You must be system manager to run makdev.
```

Only the *system* user may execute the *makdev* command.

mount

Insert a block device at a node of the directory tree structure.

SYNTAX

```
/etc/mount [<dev_name> <dir_name> [r]]
```

DESCRIPTION

The *mount* command temporarily inserts a block device at a node of the directory tree structure. As long as the device is mounted, any references to <dir_name> actually access the root directory of the device mounted there. Any files in the directory at which the device is mounted are inaccessible while the device is mounted.

The *mount* command with no arguments prints the staus of any currently mounted devices.

ARGUMENTS

<dev_name>	The name of the device to mount. It must be a block device.
<dir_name>	The name of the directory on which to mount the specified device.

OPTIONS

r	Mount the device for reading only. This option must not be preceded by a plus sign. It is useful when trying to salvage data from a damaged disk because it prevents inadvertent writing to the disk, which could make matters worse.

EXAMPLES

```
/etc/mount /dev/floppy /usr2
```

Mounts the disk in the floppy drive on the directory */usr2*. References to */usr2* now access the root directory of that disk.

```
/etc/mount /dev/disk1 /disk1 r
```

Mounts an accessory hard disk drive, disk1, as */disk1*. Because the "r" option appears on the command line, no user may write to the disk.

NOTE

When a user's working directory is the root directory of a mounted device, the command chd .. *does not change the working directory.*

ERROR MESSAGES

```
<dev_name> is not a block device.
```

The device specified either does not exist or is not a block device. Only block devices may be mounted.

```
Error mounting <dev_name> on <dir_name>: <reason>
```

The operating system returned an error when *mount* tried to insert the specified device in the directory tree. This message is followed by an interpretation of the error returned by the operating system.

```
Only read option allowed for mode.
```

The only acceptable option is the "r" option, which must not be preceded by a plus sign.

```
Syntax: /etc/mount <dev_name> <dir_name> [r]
```

The *mount* command expects exactly two arguments and, optionally, the single option "r". This command indicates that the command line does not conform to the syntax.

SEE ALSO

unmount

owner

Change the owner of a file.

SYNTAX

```
owner <user_name> <file_name_list>
```

DESCRIPTION

The *owner* command changes the owner of the specified file. Only the system manager may execute this command.

ARGUMENTS

<user_name>	The user name or user ID of the new owner of the file.
<file_name_list>	A list of the names of the files for which to change the owner. The file characteristics are preserved, including permissions and the date/time information.

EXAMPLES

```
owner system /john/*
```

Changes the owner of all the files in the directory */john* to *system.*

```
owner 110 /john/*
```

Changes the owner of all the files in the directory */john* to the user whose ID is 110.

ERROR MESSAGES

```
Error changing owner for <file_name>: <reason>
```

The operating system returned an error when *owner* tried change the owner of the specified file. This message is followed by an interpretation of the error returned by the operating system.

```
<name> is not a valid user name.
```

The specified name is not in the password file and, therefore, is not a valid user name.

```
<num> is not a valid user identification number.
```

The specified number is not in the password file and, therefore, is not a valid user ID.

```
Syntax: owner <new_owner> <file_name_list>
```

The *owner* command expects at least two arguments. This message indicates that the argument count is wrong.

```
You must be system manager to run owner.
```

Only the system manager may execute the *owner* command.

unmount

Unmount a previously mounted device from the file system.

SYNTAX

```
/etc/unmount <dev_name>
```

DESCRIPTION

The *unmount* command unmounts the specified device from the file system. Once the device is unmounted, the files in the directory on which it was mounted become accessible. Only the system manager may execute this command.

ARGUMENTS

<dev_name> The name of the device to unmount.

EXAMPLES

```
/etc/unmount /dev/floppy
```

Unmounts the floppy drive from the system, making the directory that it was mounted to accesible.

ERROR MESSAGES

```
Error unmounting <dev_name>: <reason>
```

The operating system returned an error when *unmount* tried to unmount the specified device. This message is followed by an interpretation of the error returned by the operating system.

```
Syntax: /etc/unmount <dev_name>
```

The *unmount* command expects exactly one argument. This message indicates that the argument count is wrong.

SEE ALSO

mount

Section 4
edit

INTRODUCTION

This section describes *edit*, the standard 4400 text editor, including how to call the editor, the interface between the editor and the 4400 operating system, a description of each of the editor commands (with examples), and an annotated list of the messages that the editor may issue.

edit is both content-oriented and line-oriented. Lines in the file being edited may be referenced either by specifying a line number or by specifying some part of the content of the line. *edit* is not a screen-oriented editor.

SYNTAX

```
edit [<file_name_1> [<file_name_2>]] [+bny]
```

CALLING THE EDITOR

Example:

```
edit
```

When the editor is called with no arguments, it issues a message that a new file is being created, and then prompts for the information that is to be put into the file. When the editing session is terminated (by the *stop* command, for example), the editor will prompt for the name of the file to which to write the information. The user responds to this prompt by typing in the file name, including a path name if necessary.

If an end-of-file signal is typed in response to the prompt for a file name, all information is discarded and the editing session is terminated. (See the discussion *Operating System Interface* later in this section for more information on the end-of-file signal.)

Calling the Editor with a File Name

Example:

```
edit test
```

If only one file name is given as an argument, the editor assumes that this is the file or the name of the file that is being edited.

If the file does not exist, a new file having the specified name is created. A message stating that fact is issued, and the editor then prompts for the information to be stored in the file. When the editing session is terminated, the information is written to the file.

If the file already exists, the information in it is read into an edit buffer and a prompt for an editor command is issued. When the editing session is terminated, the file will contain the revised information. The information as it was before the editor was called is preserved in a backup file (unless the *b* option was specified, as described later on). The name of the backup file is normally the name of the original file with the characters *.bak* appended to the end of it. If the original name is too long to accommodate the additional four characters, the name is truncated and the *.bak* appended to the shortened name.

Calling the Editor with Two File Names

Example:

```
edit test newtest
```

When the editor is called with two file names, the first file name is assumed to be the name of the file containing the information to be edited, and the second name is that of the file that is to receive the revised information. Both file names may contain path names if necessary to adequately describe their locations. If a path name is specified for the first file name, it is not propagated to the second file name.

In the example, the file *test* is assumed to contain the information which is to be edited, and the file *newtest* is going to contain the edited information. If the first file does not exist, the editor writes a message indicating that the edit file does not exist, and then terminates the edit session. If the second file already exists, a prompt is issued asking for permission to delete the existing file. (This prompt may be avoided with the *y* option, described below.) If an end-of-file signal is typed in response to this prompt, it is assumed that the file is not to be deleted, and the editing session is immediately terminated with no changes having been made.

Options

Options are specified to the editor by specifying an argument whose first character is a plus sign (+). The plus sign is immediately followed by one or more lowercase letters indicating the option or options selected. The options may be before, after, or intermixed with file name arguments.

b Do not create a backup file, by appending *.bak* to the source file.

n Do not initially read the file being edited. This option is meaningful only if an existing file is being edited. Normally, the editor reads the file into memory so that the information may be manipulated with editor directives. By specifying *n* as an option, the information is not initially read into memory. The user may then use editor directives to enter new information, either from the terminal or by reading other files, which will appear in front of the information in the file being edited. The *new* command must be used to start the reading of the edit file.

This option is most useful if a large amount of information is to be entered in front of the data being read from the file being edited. To insert only a small amount of information at the front of a file, the *insert* command may be used.

y Delete any existing copy of the new file or the backup file. *y* causes the editor to delete any existing copy of the backup file (if only one file name is specified) or the new file (if two file names are specified), without asking permission from the user.

If the editor cannot recognize an argument as a valid option, it issues an error message and continues to look for valid arguments.

Examples of calls including options:

```
edit test +b
edit test newtest +y
edit +nb test
```

OPERATING SYSTEM INTERFACE

The text editor follows the operating system conventions with regard to special characters and file names. For a discussion of file names, see Section 1 of this manual. The special characters and their effect on the editor are treated below.

Normally, the editor allows any character to be in a file, including control characters. There are some characters, however, which have special meaning to the operating system and thus cannot be typed in from the keyboard. The special characters with which the editor is concerned are:

- backspace character
- escape character
- line delete character
- horizontal tab character (control-i)
- control-d: keyboard signal for end-of-file
- control-c: keyboard interrupt
- control-\: *quit* signal

Backspace Character

The backspace character (Back Space on the keyboard) is used when entering commands and data to erase the last character typed.

Escape Character

The ASCII *escape* character (Esc on the 4400 keyboard) is used to temporarily stop and resuming the printing of information at the terminal. A more detailed description of the function of the *escape* character is described in the documentation of the 4400 Operating System. Here, it suffices to say that it is not possible to enter the *escape* character into a file using the editor.

Line Delete Character

The line delete character is used when entering commands and data to delete the line currently being typed.

Horizontal Tab Character

This character (Tab from the 4400 keyboard) refers to the ASCII horizontal tab character (HT), a hexadecimal 09. This is not the same as the tab character that can be defined within the editor. The editor itself is not concerned with the HT character, but the operating system may perform special handling when this character is typed or displayed. The editor treats the HT character as a single character, regardless of how the 4400 displays it.

Control-D: Keyboard Signal for End-of-File

The editor treats a *control-d* as an *end-of-file*. The action taken by the editor depends on what the editor was expecting as input. A *control-d* typed in the middle of a command has the same effect as a line delete character. If the control-d is typed as the first character in response to a request for a command (that is, in response to the # prompt), it is treated as a *stop* command. A *control-d* typed while inserting lines has the same effect as typing the line delete character followed by the line number character and a carriage return. That is, it cancels the current input line and the editor requests an editor command.

The effect of typing control-d in response to specific prompts depends on the prompt that was issued. Each such case is treated in the *Editor Command* discussions.

Control-C: Keyboard Interrupt

The editor traps the *control-c* keyboard interrupt and uses it as a signal to stop executing an *append*, *cchange*, *change*, *find*, or *print* command. It has no effect on other commands. If the editor is executing multiple commands typed on a single line, typing a *control-c* will cause the editor to stop processing those commands and request a command from the keyboard.

Control-\: Quit Signal

The *quit* signal causes the editor to terminate immediately, without making any attempt to save the edited information. If an existing file was being edited when the *quit* signal was typed, the original file is left intact without any of the changes that had been made during the edit session.

THE EDITOR'S USE OF DISK FILES

The standard 4400 text editor is a disk-oriented editor: the information being edited is read from and written to disk files. Other than the user's terminal, the only way to provide information to the editor is through disk files. When the editor is called to edit an existing file, the information in that file is read into a large buffer in memory called the *edit buffer*. It is in this buffer that all of the changes to the information take place. When the user is satisfied with the changes made, the updated information is written to a disk file in response to specific commands. If a file is larger than will fit in the edit buffer, the file must be processed in segments.

With few exceptions, the editing commands operate only on data that is in the edit buffer. Commands are provided which permit the user to flush the edit buffer of updated information and read in the next segment of data for editing. How the editor manipulates disk files depends on whether it is creating a new file or editing an existing file. In some cases, a temporary file is created to hold the updated information. If used, this temporary file is named *edit* followed by a period, 5 digits, and a single letter; for example, *edit.00324a*. Unless the editor is terminated by a *quit* signal or a fatal system error, the temporary file is destroyed at the end of the edit session.

Creating a New File

When the editor is called with a single file name and that file does not already exist, the editor will create the file at the start of the edit session and write directly into it as the edit session progresses.

When the editor is called with no file names specified, a temporary file in the user's current directory is created and the information is written to it as the edit session progresses.

At the end of the edit session, this temporary file is given the name specified in response to the *File name?* prompt.

Editing an Existing File

When the editor is called with a single file name, and that file already exists, a temporary file is created and the information is written to it as the edit session progresses. The temporary file is created in the same directory in which the file being edited resides. At the end of the edit session, the original file is renamed to the backup file name, and the temporary file is given the name of the original file. If no backup file is requested (by specifying a *b* option), the original file is destroyed and the temporary file is given the name of the original file.

When the editor is called with two file names specified, the second file is created and the updated information is written directly into it. The original file is not changed.

Command Input From a File

It is possible to use I/O redirection to have the editor read its commands from a file instead of from the keyboard. The editor will process the commands as though they were entered from the terminal's keyboard. If the end of the command file is reached before a *stop* or *abort* command is read, the action is the same as though a *control-d* were typed from the keyboard. (See the discussion of *control-d* earlier in this section.)

Fatal Errors

The text editor attempts to make an intelligent decision when confronted with an error response to an operating system call. However, if an error is received which is unexpected and indicates that the editor cannot continue to function, it will issue a message and terminate immediately. The various messages, both fatal and nonfatal, are listed under the heading *Editor Messages* later in this section.

EDITOR COMMANDS

Using Strings

Several editor commands use character strings as arguments. These arguments are either matched against strings in the text, or replace a string in the text. A string argument begins after a delimiter character and continues as a sequence of any characters until the delimiter is again encountered. The delimiters are not considered part of the string to be used in the matching or replacement operations.

Although the delimiters in the following descriptions are frequently represented as slashes, /, nearly any non-blank, non-alphanumeric character may be used as the delimiter such as: * / ( ) $, . [ ] : ' etc. Note that the following characters may not be used to enclose strings unless they are preceded by either a plus (+) or minus (-) sign: ^ (denotes first line of file), ! (denotes last line of file), - (denotes target is above current line), and the character denoted by *lino* (normally a pound sign), which is used to indicate line numbers. The equals sign = may not be used as a string delimiter.

The delimiter character is redefined in each new request by its appearance before a string. If two strings exist in one command (as in the *change* command), the same delimiter character must be used for each string.

All editor commands use the <line> information preceding the command to position the pointer prior to any command action. The <line> parameter may of course be null, meaning leave the pointer at its current position. All of the following are valid <line> designators:

Any number	The specific line number
+n	The nth subsequent line
-n	The nth previous line
/<string>/	The next line in the file containing the indicated string of characters
-/<string>/	A previous line containing the indicated string
^	The first line of the file
!	The last line of the file
null	The current line

Line numbers less than 1.00 must be specified with a leading zero. For example, even though the editor may display a line number as *.10*, it should be specified as *0.10* when used in commands. The maximum line number is 65535.99. Inserting after this maximum line number will cause the line numbers to *wrap around* back to zero.

Many editor commands require <target> information. This tells the editor to operate on the *current* line and all other lines in the file up to the line referenced by the <target>. In cases where a <target> is required, leaving it null will make the <target> default to one, and only the current line will be affected. All of the following are valid <target> designators:

an integer n	n lines should be affected by the edit operation
#n	The line number of the last line to be affected. The # is actually the *lino* character and may be changed by the user with the *set* command.
/<string>/	The next line in the file containing the specified character string.
-/<string>/	The previous line containing the indicated string
^	All lines up to the top of the file
!	All lines to the bottom or last line of the file
+or- n	Indicates that n lines should be affected and in which direction from the current line
(null)	Defaults to 1 and only the current line is affected

As we have seen, <target> is used to specify a range of lines to which the command will apply. The command will be applied to each line, starting with the line specified by <line> and continuing until the target is reached.

If a string <target> is specified, the command will apply to successive lines of text until a line containing the string is reached. Processing proceeds downward in the edit buffer unless the target is preceded by a - (minus sign), indicating that processing is to proceed upward (toward the first line) in the edit buffer. Targets may also be preceded by a plus sign (indicating downward movement). If a line number target is specified, processing begins at <line> and proceeds toward the target line number. Some examples of <target>s are:

```
2
+10
-3
/STRING/
+/STRING TARGET/
-/BACKWARD DISPLACEMENT TO A STRING/
+*ANY DELIMITER WILL WORK FOR STRING*
++EVEN PLUS SIGNS CAN WORK+
#23.00
```

Specifying a Column Number

Any */<string>/* descriptor may be postfixed with a column number immediately after the second delimiter to indicate that the preceding string must begin in the column specified. If the column specified is not in the range of the zone in effect, the request will be ignored. (See the *zone* command.) Some examples are:

```
/IDENT/11
/PROGRAM/77
*LABEL*2
$COMMENT$30
```

Using the Don't-Care Character

A *Don't-Care Character* may be set to allow indiscriminate matches of parts of a string. When this character is placed in a string, any character in the file will automatically match. The Don't-Care Character will have its special meaning only in a string being used to search the file. In other words, the Don't-Care Character will not act as such in a replacement string such as the second string of a *change* command. The Don't-Care Character may be effectively disabled by setting it to a null. Assuming we have previously set the Don't-Care Character to a *?*, here are some examples:

/A???/	Matches any 4-letter string beginning with A
@03/??/78@	Matches all days in the 3rd month of 1978
/???/9	Matches any 3-letter string starting in column 9

The Command Repeat Character

The *command repeat character,* control-r, repeats the last command in the input buffer. Some examples of commands which may be useful to repeat are:

PRINT 15	To print a screen of lines at a time
NEXT	Allows you to single step through the file with one key
^CO!!	To quickly fill the workspace
FIND/SOME STRING/	If the first string found is not the one desired

Using the EOL Character

The editor supports an *eol* or *End Of Line* character to allow multiple commands in a single line. There are some commands that cannot be followed by another command on the same line. This fact is documented in the descriptions of those commands. The *eol* character may be changed by using the editor's *set* command. An example of *eol* use (with *eol* set to *$*) is:

```
^D2$P10$T
```

This sequence will delete the first 2 lines of the file, then print the next 10 lines, and finally return the pointer to the top of the file.

Using Tabs

You may specify a tab character and up to 20 tab stops. The tab character may then be inserted into a line, where it will be replaced by the appropriate number of fill characters when the end of the line is received. The fill character defaults to a space, but may be changed to another character with the editor's *set* command. If tab stops or the tab character have not been previously set, but some character has been used throughout the file as a tab, it can still be expanded by setting it to be the tab character, setting up your tab stops and then using the *expand* command on the file.

Note that if the tab character has been set, subsequent uses of the *insert* or *replace* commands will cause automatic tab expansion. However if a tab character is added to the file by the use of a *change*, *append*, or *overlay* command, that character will remain intact in the file until the *expand* command is invoked on the line containing that tab character.

After tabs are expanded, the tab character no longer exists in the data. All occurrences will have been replaced by the appropriate number of fill characters. Setting the tab character to be the same as the fill character effectively disables the tab feature. Note the the tab character described above is distinct from the ASCII horizontal tab character (HT or control-i). The effect of the HT character is described in the *Operating System Interface* discussion earlier in this section. It is possible to set the editor tab character to the HT character. If this is done, the operating system may take special action when the HT character is typed, but the character will be replaced by fill characters when it is put into the edit buffer.

Length of Text Lines

Lines entered from the keyboard are limited to 255 characters. The lines in the text file may be of any length. Lines longer than 255 characters may be created with the *merge* and *append* commands.

Commands

There are five groups of editor commands: environment commands, system commands, *current line* movers, edit commands, and disk commands. A complete description of all commands in each group is given below. In the following descriptions, quantities enclosed in square brackets ([...]) are optional and may be omitted. A backslash (\ ) is used to separate options. Many commands have abbreviations. Both the full name of the command and its abbreviation are given. A command and its abbreviation may be used interchangeably. All commands below are in lower case; however, in use, a command may be in either upper case or lower case.

ENVIRONMENT COMMANDS

dk1

Syntax

```
dk1 <command string>
```

Description

dk1 is used to define one of two *command constants*, which can be executed at any time by the *k1* command. The <command string> is a single command or several commands separated by the *eol* character (see *set* command). All of the command line, including the carriage return is assumed to be the argument to the *dk1* command. The *dk1* command is most useful for remembering and re-executing a frequently used sequence of commands.

Example

dk1 f -/.nl/1$i/.sp	Define a command sequence of *f -/.nl/1* followed by *i/.sp*. This assumes that *eol* is *$*. This sequence may be executed by typing *k1*.

dk2

Syntax

```
dk2 <command string>
```

Description

dk2 is used to define one of two *command constants*, which can be executed at any time by the *k2* command. The <command string> is a single command or several commands separated by the *eol* character (see *set* command). All of the command line, including the carriage return is assumed to be the argument to the *dk2* command. The *dk2* command is most useful for remembering and re-executing a frequently used sequence of commands.

Example

dk2 c /sample// 1 2	Define the command constant: *c /sample// 1 2*. This command may be executed by typing *k2*.

esave

Syntax

```
esave [<path_name>]
```

Description

The *esave* command saves the current editor *environment* on an *editor configuration* disk file named *.editconfigure* in the user's directory. The editor environment consists of the *header* column count; the *numbers* and *verify* flags; current tab stops; the *tab*, *dcc*, *fill*, *eol*, and *lino* characters; the commands saved as command constants *k1* and *k2*; and the search zones in effect. When the editor is called, the environment is automatically set from the configuration file in the user's directory, if one exists. The editor environment may also be reset from the configuration file at any time during the edit session by the *eset* command, described below.

The environment information may be saved in a directory other than the user's current directory by specifying a path name as an argument to the *esave* command. This path must include only directory names and must be terminated by the pathname separator /.

Example

esave	Save the current editor environment on the file
esave /dde/	*Save the current editor environment in file /dde/.editconfigure.*

eset

Syntax

```
eset [<path_name>]
```

Description

The *eset* command is used to reset the editor environment from an editor *configuration* file created by the *esave* command (see above). The configuration file is named *.editconfigure* and is normally expected to be found in the user's current directory. A path name may be specified as an argument to the *eset* command to force the searching of a different directory. This path must include only directory names and must be terminated by the pathname separator /.

Example

eset	Reset the editor environment from the file
eset /dde/	*Reset the editor environment from file /dde/.editconfigure.*

header

Syntax

```
header [<count>]
h [<count>]
```

Description

A header line of <count> columns will be displayed. The heading consists of a line showing the column numbers by tens, followed by a line of the form *123456789012...* to indicate the column number. Columns for which tab stops are set will contain a hyphen instead of the normal digit. If a column count is given, it becomes the default so that if just *h* is subsequently typed, that number of columns will be printed.

Example

header 72	Display column number headings for 72 columns
h 30	Display column numbers for 30 columns

k1

Syntax

```
k1
```

Description

Execute the command constant that was defined by *dk1*. If no command constant was defined, the current line is printed. This command may not be followed by another command on the same line.

Example

k1	Execute the command constant.

k2

Syntax

```
k2
```

Description

Execute the command constant that was defined by *dk2*. If no command constant was defined, the current line is printed. This command may not be followed by another command on the same line.

Example

k2	Execute the command constant.

lk1

Syntax

```
lk1
```

Description

Display the command constant that was defined by *dk1*. If no command constant was defined, a blank line is printed.

Example

lk1	Display the command constant.

lk2

Syntax

```
lk2
```

Description

Display the command constant that was defined by *dk2*. If no command constant was defined, a blank line is printed.

Example

lk2	Display the command constant.

numbers

Syntax

```
numbers [off/on]
nu [off/on]
```

Description

The line number flag is turned off or on. If the flag is off, then line numbers will never be printed. If neither *off* nor *on* is specified, then the flag will be toggled from its current state.

Example

numbers off	Turn line number printing off
nu on	Turn it back on
nu	Toggle from on to off or from off to on

renumber

Syntax

```
renumber
ren
```

Description

The *renumber* command will renumber all of the lines in the current edit buffer. Lines in the renumbered buffer will start with the line number of the first line in the buffer and will have an increment of one. The current line does not change, although its number will probably have been changed.

Example

renumber	Renumber the lines in the current edit buffer
ren	Renumber the lines in the current edit buffer

set

Syntax

```
set <name> = '<char>'
```

Description

set is used to define certain special characters or symbols. The <name>s which may be set are:

tab	the tab character
fill	the tab fill character
dcc	the "don't care" character for string searches
eol	the end of line character which may be used to separate several commands on a single line
lino	the line number flag character which is used to indicate that a target is a specific line number

The default values are: dcc, tab, and eol are null, fill is the space character, lino is #

The default values may be initialized from a configuration file in the user's directory. See the *esave* command.

Example

set tab='/'	Set the tab character to a slash
set tab=''	Disable tabbing by setting the tab character to a null
set fill=' '	Set tab fill character to a blank
set eol='$'	Set the EOL character to $
set lino='@'	Set the line number flag to @

tab

Syntax

```
tab [<columns>]
```

Description

Used to set the tab stops. All previous tab stops are cleared. If no columns are specified, then the only action is to clear all tab settings. Any tab characters occurring beyond the last tab stop are left in the text. The maximum number of tab stops allowed is 20. Tab stops MUST be entered in ascending order.

Example

tab 11,18,30	Set tab stops at columns 11, 18, and 30
tab	Clear all tab stops

verify

Syntax

```
verify [on/off]
v [on/off]
```

Description

The verify flag is turned on or off. The verify flag is used by the commands *change* and *find* (and several others) to display their results. If neither *on* nor *off* is specified, then the flag will be toggled from its current state.

Example

verify off	Turn verification off
v on	Turn it back on

zone

Syntax

```
zone [c1,c2]
z [c1,c2]
```

Description

zone is used to restrict all sub-string searches (find, change, <target>s, etc.) to columns *c1* through *c2* inclusive. Any substrings beginning outside those columns will not be detected. If *c1* and *c2* are not specified, then the zones will be reset to their default values (columns 1 and 255). A string which starts within the specified search zone and extends out of it will still match a target.

Example

zone 11,29	Restrict searches to columns 11 through 29
zone	Search columns 1 through 255

SYSTEM COMMANDS

abort

Syntax

```
abort
```

Description

This command terminates the edit session without saving any of the changes made during that session. The original file, if one exists, is left intact. When typed, this command will prompt "Are you sure?". If a *y* is then typed, the edit session will be terminated. Typing an *n* or end-of-file signal will cause the editor to look for another command. Typing any other character will cause the prompt to be issued again.

Example

abort	Abort the editing session.

edit

Syntax

```
edit <editor arguments>
e <editor arguments>
```

Description

The *edit* command causes the current editing session to be terminated (as though a *stop* or *log* command had been entered), and another editing session started. The <editor arguments> are any valid file names and editor options as described earlier in this section under the heading *Calling the Editor*. This command may not be followed by another command on the same command line. All changes to the editing environment made by *Environment Commands* remain in effect.

Example

edit test +b	Terminate the current editing session and start editing file *test* with editor option *b*.

log

Syntax

```
log
```

Description

This command ends the editing session. The updated information is written to the new file, and, if necessary, any unprocessed data from any existing file is copied to the new file. A backup file is created if circumstances warrant it. (see the *Operating System Interface* discussion earlier in this section for more information on the editor's handling of disk files at the end of an editing session.)

Example

log

stop

Syntax

```
stop
s
```

Description

Same as *log*.

Example

stop

s

u

Syntax

```
u <operating_system_command>
```

Description

The *u* command permits the execution of an operating system command. The specified command is passed to the *shell* program for execution. The editor waits for the operating system command to finish before prompting for another editor command. This command may not be followed by another editor command on the same line. See the "x" system command.

Example

u list test	List the file *test*
u copy test test1	Copy the file *test* to *test1*

wait

Syntax

```
wait
```

Description

The *wait* command is used to wait for the completion of a background task generated by the *x* command (described below). This command cannot be used to wait for completion of a background task that was not generated by the editor. The editor will not request a command until the background task is completed or a keyboard interrupt (control-c) is typed. When the background task terminates, a message is displayed specifying the task number and whether it completed normally or abnormally. In the event of abnormal termination, the response code or interrupt code that caused the termination is given.

Example

wait	Wait for the background task to complete

X

Syntax

```
x <operating_system_command>
```

Description

The *x* command is used to start a background task running. The <operating_system_command> which was specified as the argument is passed to the *shell* program for execution. The task generated must run to completion before the editor will allow the generating of another such background task. The *wait* command must be used to receive the termination status of a task before the *x* command may be used again. This command may not be followed by another command on the same line. See the "u" system command.

Example

x copy test test1	Copy *test* to *test1* as a background task. A *wait* command must be used to determine the termination status of the task before another background task can be generated.

CURRENT LINE MOVERS

bottom

Syntax

```
bottom b
```

Description

Moves to the last line in the file and makes it the current line.

Example

bottom	Make the last line of the file the current line
b	Make the last line of the file the current line

find

Syntax

```
find <target> [<occurrence>]
f <target> [<occurrence>]
```

Description

Moves the current line pointer to the line specified by <target> and makes it the current line. If the verify flag is on (see *verify*), the line will be printed. If <occurrence> is specified (an unsigned integer or an asterisk), the command will be repeated <occurrence> times. If <occurrence> is an integer, it must not start in the first column following the second delimiter of a string <target>, as it would then appear to be a column specifier for that string. If no column is to be specified, insert a space after the second delimiter and before the <occurrence>, as in the second example given below. An asterisk means all occurrences of the <target> will be found until the bottom or top of the edit buffer is reached. If the target is not found, the current line pointer will not be moved.

Example

find /string/	Find the next line containing the string *string*
f/three lines/ 3	Find the next three lines containing the *three lines*
f/all 'til bottom/*	Find all following occurrences of the indicated string
f-/program/7 *	Find all previous lines which have the word *program* starting in column seven

next

Syntax

```
next [<target> [<occurrence>]]
n [<target> [<occurrence>]]
```

Description

The line specified by the target is made the current line. If the verify flag is on (see *verify*), the line will be printed. If <occurrence> is specified, it must be an unsigned integer. It indicates which occurrence of a line containing the target is to be made the current line. If the target is not reached, the current line pointer will be positioned at the bottom of the edit buffer (or top of the edit buffer for a negative <target>). If no target is specified, the next line will be made the current line.

Example

next 5	Make the fifth following line the current line
n	Make the next line the current line
n-10	Make the 10th previous line current
n/string target/	Make the next line containing *string target* to be the current line
n/3rd occurrence/3	Make the third line containing the indicated string the current line

position

Syntax

```
position <target>
pos <target>
```

Description

Searches forward through the file for an occurrence of <target> and makes the line in which it occurs the current line. If the target is not found in the current edit buffer, the edit buffer is flushed and the next edit buffer is read from the file being edited. This process continues until the target is located or the end of the file is detected. If the target cannot be located, the current position is the first line in the last edit buffer.

The <target> may not be a *backwards target* (preceded by a minus sign) and may not be an integer indicating relative displacement. Only a string or a line number (preceded by the *lino* character) are valid targets. Search zones are honored during the search for the target. A column number is allowed after the target, but an occurrence specification is not permitted.

Example

position /string/5	Position to the line containing the string *string* in column 5.
pos #1000	Position to line number 1000

top

Syntax

```
top
t
```

Description

The first line of the file becomes the current line.

Example

top	Make the first line of the file the current line

EDITING COMMANDS

append

Syntax

```
append /<string>/ [<target>]
a /<string>/ [<target>]
```

Description

Appends the specified <string> after the last character of the current line (and to successive lines until the target is reached).

If the string is postfixed with a column number, then the string is added beginning at the specified column (rather than at the end of the line). Any characters previously in the line following the specified column are overwritten.

Example

append /./	Append a period to the end of the current line
a *HELLO*	Append the word *HELLO* to the end of the current line and to the end of the next line.
a/sequence/73 *END*7	Append the word *sequence* starting in column 73 of the current line and successive lines until a line containing the characters *END* beginning in column seven is found.

break

Syntax

```
break
```

Description

The *break* command allows the splitting of a line into two lines. The current line is printed, then a line of input is accepted from the terminal (the break line). When the line is printed, all ASCII HT characters will be displayed as spaces so that the terminal cursor will not be artificially advanced. The break line will be positioned directly beneath the line printed out.

In response to the *Break---* prompt, type any characters to move the cursor until it is beneath the character that is to be the first character of the second line. Then type a carriage return.

After the line is split, the second half of the broken line becomes the current line. If you type an end-of-file signal in response to the *Break---* prompt, the current line will not be changed. The current line will also not be changed if the carriage return typed in the break line is beyond the end of the current line.

Example

```
break
      25.00 This is the current line.
   Break--xxxxxxxxxxxx
```

The line will be broken at the start of the word *current.*

Example 4-1. .

change

Syntax

```
change /<string1>/<string2>/ [<target> [<occurrence>]]
c /<string1>/<string2>/ [<target> [<occurrence>]]
```

Description

Replaces <string1> with <string2>. If <string2> is omitted, <string1> is deleted. If no <target> is specified, only the current line is affected. The slashes represent any non-blank delimiter character.

<occurrence> specifies which occurrence of <string1> is to be replaced in each line. It is either an unsigned integer or an asterisk (*) signifying that all occurrences of the substring <string1> are to be replaced with <string2>. By default, only the first occurrence will be changed. Note that if <occurrence> is specified, and if changes are to occur to the current line only, then the target should be *1*.

Example

change /this/that/	Replace the first occurrence of *this* in the current line with *that*
c/A/B/ 1*	Change all occurrences of *A* in the current line to *B*
c /first/last/10	Change the first occurrence of *first last* in the current line and also in the nine following lines
c /new/old/ /a target/	Change the first occurrence of *new* to *old* in each line down through the line containing the string *a target*
c ,a,, -10 *	Remove all *a*s in the current line and in the nine preceding lines
c*Hello*	Delete the character string *Hello* from the current line

cchange

Syntax

```
cchange /<string1>/<string2>/ [<target> [<occurrence>]]
cc /<string1>/<string2>/ [<target> [<occurrence>]]
```

Description

cchange stands for Controlled Change. This command is exactly like the normal *change* command except that you can interactively specify whether each line containing <string1> should actually be changed or left as is. This allows you to step through the edit buffer and selectively change certain strings. When a line containing <string1> is found, it is displayed at the terminal and you receive a prompt, *Change?* Type a *y* to change the line. If you type an *s* or end-of-file signal, the command will terminate. Other characters will cause a search for the next line containing <string1>.

Example

cchange/ALPHA/OMEGA/!*	Perform a Controlled Change on all occurrences of *ALPHA* through the rest of the file
cc;a;z;-20 3	Perform a Controlled Change on the third occurrence of *a* in the current and previous 19 lines

copy

Syntax

```
copy [<destination-target> [<range-target>]]
co [<destination-target> [<range-target>]]
```

Description

Copies the current line through <range-target> and places the copied text after the <destination-target>. The default <destination-target> is 1, thereby placing a copy of the current line after the next line. The default <range-target> is 1, thereby copying only one line. After the command is executed, the current line pointer will be set to the new position of the last line copied. Some lines may be renumbered after a copy with no renumbering message issued.

Example

co #18	Put a copy of the current line after line 18
copy #3 4	Copy four lines beginning with the current line and place them after line 3
co /check/ +/range/	After the next line which has the string *check*, place a copy of each line starting with the current line through the line containing *range*

delete

Syntax

```
delete [<target>]
d [<target>]
```

Description

Deletes the current line (and successive lines until the target is reached). After the command is executed, the current line will be the line following the last line deleted.

Example

delete 5	Delete five lines (the current line and the next four lines)
d	Delete the current line
d /STRING/	Delete lines from the current line through the next line that contains the string *STRING*

expand

Syntax

```
expand [<target>]
exp [<target>]
```

Description

The current tab character is expanded within all lines, beginning with the current line, continuing down to and including the line specified by <target>. Since tabs are normally expanded as lines are inserted into the file, this command is primarily of use when one has forgotten to define a tab character or has inserted a tab character with an *append*, *overlay*, or *change* command.

Example

expand 100	Expand 100 lines starting with the current line
exp	Expand the current line

insert

Syntax

```
insert
i
```

Description

The editor will enter the input mode, prompting with line numbers (unless line numbers have been disabled, with the *numbers* command) and insert the lines below the current line. The editor remains in *insert* until you begin a line with the *lino* character or the end-of-file signal in column one. The editor treats any characters following the *lino* character as an editor command. (If you type the *line delete character*, the editor does not re-issue the prompt.

If possible, the editor will number the inserted lines with an increment small enough to insert at least 10 lines between the current line and the next line. The editor will renumber lines following the inserted text if the inserted text line numbers overlap numbers already in the file. (The current line pointer is left at the last line inserted.)

You may insert lines at the top of the edit buffer by specifying a line number of zero.

This command may not be followed by another command on the same line.

Example

insert	Accept line input after the current line
0i	Insert at the top of the edit buffer.

insert

Syntax

```
insert <text>
i <text>
```

Description

Inserts <text> as a separate line below the current line of the file. Use a space as a separator following the command name. The line inserted becomes the current line. The editor may renumber text lines following the inserted text if the inserted line number overlaps line numbers already in the file.

This command may not be followed by another command on the same line.

Example

```
I This below the current line of the file
              insert everything after the first blank
```

Example 4-2. .

merge

Syntax

```
merge
```

Description

Merges the current line and the line immediately following it into a single line. The merged line becomes the current line.

Example

merge	Merge the current line and the next line into a single line.

move

Syntax

```
move [<destination-target> [<range-target>]]
mo [<destination-target> [<range-target>]]
```

Description

Moves the current line through <range-target> so that they follow the line specified by <destination-target>. The defaults for <destination-target> and <range-target> are both 1, so *move* without arguments interchanges the current line and the next line. After the command is executed, the current line pointer will be set to the new position of the last line moved. Some lines may be renumbered with no renumbering message issued.

Example

move 3	Move the current line down three lines
mo #1 /TARGET STRING/	Move the current line and all lines down thru the line containing *TARGET STRING* after line 1
mo -/Program/ 5	Move five lines (including the current line) up within the file so that they follow a line containing the character string *Program*
mo #10 -5	Move the current line and the four previous lines below line number 10

overlay

Syntax

```
overlay[<delimiter>]
o[<delimiter>]
```

Description

This command prints the current line, then accepts a line of input (the overlay line). When the line is printed, all ASCII HT characters will be displayed as spaces so that the terminal cursor will not be artificially advanced. The overlay line will be positioned directly beneath the line printed out. Each character of the overlay that is different from the <delimiter> character (which defaults to a blank) will replace the corresponding character in the current line. The overlaid line will be printed if verify is *on*. If the end-of-file signal is typed in response to the prompt for the overlay line, the current line will not be changed.

Example

```
overlay
     25.00=THIP IS THE CORRENT LUNE.
   Overlay    S          U
     25.00=THIS IS THE CURRENT LINE.
```

Example 4-3. .

overlay

Syntax

```
overlay<d><text>
o<d><text>
```

Description

This command is similar to the previous form of the *overlay* command with these differences: (1) The current line is not printed. (2) The remainder of the command line (after the delimiter character) is taken as the overlay text.

Example

```
overlay--- AT-------------------- NUMBER.
   25.00=THAT IS THE CURRENT LINE NUMBER.
```

Example 4-4. .

print

Syntax

```
print [<target>]
p [<target>]
```

Description

Prints all lines from the current line through the line specified by <target>. By default, only the current line will be printed.

Example

p	Print the current line
print 5	Print 5 lines starting with the current line
p -10	Print the current line and the nine previous lines
print *string*	Print all lines down thru the next line containing *string*
p -/string/	Print all lines up through the next previous line containing *string*

replace

Syntax

```
replace [<target>]
r [<target>]
```

Description

This command deletes from the current line through <target>, then places the editor in input mode, putting the new lines into the area vacated. It is not necessary to enter the same number of lines as were deleted. The line numbers of the lines inserted will probably not be the same as those deleted. The current line pointer will be positioned at the last line inserted. By default, only the current line will be deleted. This command may not be followed by another command on the same line.

Example

r	Replace the current line
replace 10	Replace 10 lines starting with the current line
r /TARGET STRING/	Replace all lines from the current line through the line containing *TARGET STRING*

text

Syntax

```
=<text>
```

Description

Replaces the current line with the text that follows the equal sign. The current line pointer is not moved.

Example

```
=THIS IS REPLACEMENT TEXT.
```

Example 4-5. .

null

Syntax

```
(null)
```

Description

The null command (i.e., just a carriage return) prints the current line.

DISK COMMANDS

flush

Syntax

```
flush
```

Description

The information above the current line in the edit buffer is written to the file containing the updated data and then deleted from the edit buffer. Use this command to make room in the edit buffer for large insertions.

Example

flush	Flush information above the current line to updated file.
200flush	Flush information above line 200 to the updated file.

new

Syntax

```
new
```

Description

The information above the current line in the edit buffer is written to the file containing the updated data and then deleted from the edit buffer. The available space in the edit buffer is then filled with data read from the file being edited. This command is used primarily to proceed to the next segment of the file when modifications to the current edit buffer have been completed. If a new file is being created, the *new* command is the same as the *flush* command.

Example

new	Write the information above the current line to the updated file and read more data from the file being edited.
new	Write the current edit buffer (except for the first line) to the updated file and read the next segment from the file being edited into the edit buffer.

read

Syntax

```
read [<file name>]
```

Description

Places the contents of the specified file after the current line. The last line of the information read becomes the current line. If you omit the file name, the editor prompts you for it. If you type an end-of-file signal in response to the prompt, no data is read. The file name may contain path information if any is necessary to locate the file.

The entire contents of the file must fit into the remaining unused space in the edit buffer. If the file being read will not fit into the edit buffer, the message *Not enough room* is issued and no data is read.

Example

read /dde/data	Reads the information in the file */dde/data* and place it after the current line.
100read moredata	Read the information in the file *moredata* and place it after line 100.

write

Syntax

```
write [<target>]
```

Description

The editor prompts you for a file name, then writes the information from the current line through <target> to a file. If an end-of-file signal is typed in response to the prompt, no information is written. If the file being written already exists, it is destroyed and a new file created. If no <target> is specified, only the current line is written.

Example

write /window/	Write the information from the current line through the line containing the string *window*.
100write #200	Write lines 100 through 200, inclusive, to a scratch file.

EDITOR MESSAGES

`A task is already running`

The *x* command was used when there was already a task generated by a previous *x* command still running. The *wait* command must be used to wait for the previous task to complete before initiating another background task.

`Attempting to merge onto last line of text`

The *merge* command joins the specified line with the following line, and if the specified line is the last line of the file, there is no line following the specified line to join with it.

`Bottom of file reached`

An informative message issued when the last line of the file is deleted.

`Cannot create configuration file`

A configuration file could not be created in the directory specified in the *esave* command (current directory if no directory was mentioned). Usually this means that the directory specified could not be found or you don't have write permissions on that directory. Make sure the directory was specified with a trailing "/" character.

`Cannot create new file`

The editor was called with two file names as arguments, but the second file could not be created. This message is preceded by a message indicating which error was detected. This is a fatal error and will cause an immediate exit from the editor. This message occurs only at the beginning of an editing session.

`Cannot create new backup file`

The editor detected an error attempting to create a backup file. This message is preceded by a message indicating which error was detected. The new backup file is not created and the editing session continues.

`Cannot create task`

An error was detected when trying to generate a task with the *u* or *x* command. This message is preceded by a message indicating which error was detected. The command is aborted and the editor requests a new command.

```
Cannot create temporary file
```

The editor detected an error when trying to create the temporary file that holds the updated information. This message is preceded by a message indicating which error was detected. This message occurs only at the beginning of an editing session.

```
Cannot delete old backup file
```

At the end of an editing session, the editor attempts to create a backup file containing the information as it was prior to the editing session. However, a file already exists with the backup file name, and that file could not be deleted. This message is preceded by a message indicating which error was detected. The new backup file is not created and the editing session continues.

```
Cannot open configuration file
```

The configuration file in the directory specified in an *eset* command could not be opened. This usually means that there was no configuration file in the specified directory, or that the specified directory could not be found, or that you do not have read permission for the configuration file. Remember that the directory name must be specified with a trailing '/' character.

```
Cannot open edit file
```

The file that is being edited exists, but could not be opened. This message is preceded by a message indicating which error was detected. This is a fatal error and will cause an immediate exit from the editor. This message occurs only at the beginning of an editing session.

```
Cannot open new file
```

The editor was called with two file names as arguments, but could not open the second file to determine if it already exists. This message is preceded by a message indicating which error was detected. This is a fatal error and will cause an immediate exit from the editor. This message occurs only at the beginning of an editing session.

```
Cannot read configuration file
```

The operating system reported a media error while the editor was trying to read from the editor configuration file.

```
Cannot read edit file
```

The operating system reported a media error while the editor was reading from the file whose data is being edited.

`Cannot rename files`

The editor detected an error trying to rename the files at the end of an editing session. This message is preceded by a message indicating which error was detected. This is a fatal error and will cause an immediate exit from the editor. The user should then search for the temporary file used by the editor. This file will contain the updated information and should be copied to another file for safe keeping.

`Cannot write configuration file`

The operating system reported a media error while the editor was writing configuration data to the configuration file in the specified directory (current directory if the specification was omitted).

`Delete existing backup file?`

At the end of an editing session, the editor attempts to create a backup file containing the information as it was prior to the editing session. However, a file with the same name as the backup file would have already exists. This message is a request for permission to delete the existing file, replacing it with the new backup file. The prompt must be answered with a *y*, for *yes*, or an *n*, for *no*. If *y* or the end-of-file signal is typed, the file is deleted and the new backup file is created. If *n* is typed, the file will not be deleted and no new backup file created. If none of these are typed, the prompt is re-issued.

`Delete existing copy of new file?`

The editor was called with two file names as arguments. The second file already exists and must be deleted before the editing session can continue. This message is a request for permission to delete the file. The prompt must be answered with a *y*, for *yes*, or an *n*, for *no*. If *y* is typed, the file is deleted and the editing session continues. If *n* or the end-of-file signal is typed, the file will not be deleted and the editing session is terminated. If none of these are typed, the prompt is re-issued.

`Edit file does not exist`

The editor was called with two filenames, but the first file, which contains the data to be edited, could not be found. The editor will terminate immediately.

`Empty text buffer`

The text buffer is empty (contains no text) and the requested command could not be completed.

`Error attempting to open file`

The file specified in a *write* command could not be opened for writing. This usually means that the specified file could not be created because the path to the file was inaccessible, or the permissions on the directory in which the file was to reside exclude the you from creating a file there, or the file exists but the you do not have write permission for the file.

`Error copying edit file`

At the end of an editing session, any unread data on the file that is being edited is copied to the new file being written. An error was detected during this copy process. This message is preceded by a message indicating which error was detected. This is a fatal error and will cause an immediate exit from the editor.

`Error creating scratch file`

The file specified in a *write* command could not be created. This message is preceded by a message indicating which error was detected. The *write* command is aborted and the editor requests a new command.

`Error opening scratch file`

The file specified in a *read* command could not be opened. This message is preceded by a message indicating which error was detected. The *read* command is aborted and the editor requests a new command.

`Error reading data file`

The editor detected an error when trying to read from the file being edited or from a scratch file with the *read* command. This message is preceded by a message indicating which error was detected. The current command is aborted and the editor requests a new command; no data read from the file is kept. If the file being read was the file being edited, you should use the *abort* command to abandon the editing session since the file being read is no longer positioned correctly.

`Error waiting for task to complete`

An error was detected when waiting for a task generated by the *u* or *x* command to complete. This message is preceded by a message indicating which error was detected. The command is aborted and the editor requests a new command.

```
Error writing new file
```

The editor detected an error when trying to write the contents of the edit buffer to the file that holds the updated information. This message is preceded by a message indicating which error was detected. This is a fatal error and will cause an immediate exit from the editor. All changes to information still in the edit buffer are lost.

```
File is a directory
```

An attempt was made to edit a directory, not a text file. This is a fatal error and causes an immediate exit from the editor. This message occurs only at the beginning of an editing session.

```
File name?
```

This is the prompt used when the editor requests a file name. Commands that may request a file name are *read* and *write*. The editor will also request a file name in response to the *stop* and *log* commands if no file names were specified when the editor was called.

```
Input error
```

An error status was returned by the operating system in response to a request for input from the standard input device. This is normally the terminal keyboard and should not generate any such error. If the standard input has been redirected to a disk file, an error may be generated when reading the disk for input characters. In either case, this is a fatal error and causes an immediate exit from the editor. All changes to information still in the edit buffer are lost.

```
Line too long
```

The maximum size for a line being input to the editor is 255 characters. Lines in the file being edited may be of any length, but those entered from the standard input device are limited to 255 characters.

```
Name too long
```

The file name entered in response to a *File name:* prompt is too long. The maximum size of a file name, including the path specification, is 55 characters.

```
New file being created
```

This is an informative message indicating that there is no existing file of information to be edited and that a new file is being created.

```
New file is the same as the old file
```

The editor was called with two file names as arguments, but both names point to the same file. Either the file names are the same, or the two files have been linked with the *link* system call.

```
No child task exists
```

The *wait* command was used when no background task had been generated by the editor.

```
No lines deleted
```

An informative message indicating that the *delete* command was used but the target could not be located, and you answered *no* to the prompt asking if the delete was to proceed.

```
No such line
```

A line number or target could not be found.

```
Not enough room
```

The file being read with the *read* command could not fit in the available space in the edit buffer. None of the information read from the file is kept. You can use the *flush* command to try to make room for the file. If that fails, the file being read should be split into smaller files that may be read individually.

```
Not found
```

A target could not be found.

```
Output error
```

An error status was returned by the operating system in response to a request to send output to the standard output device. This is normally the terminal display and should not generate any such error. If the standard output has been redirected to a disk file, an error may be generated when writing the data to the disk file. In either case, this is a fatal error and causes an immediate exit from the editor. All changes to information still in the edit buffer are lost.

```
Positioning backwards is not allowed
```

The *position* command was called with a target that has a leading minus sign, indicating a backward search.

```
Relative positioning is not allowed
```

The *position* command was called with a target that is an unsigned integer, indicating a relative displacement forward in the file.

```
Some lines renumbered
```

An *insert, replace,* or *break* command caused some lines in the file to be renumbered. Note that the *move* and *copy* commands will cause renumbering without this message being issued.

```
Source overlaps destination
```

With the *copy* or *move* commands the target line was within the range of data being copied or moved.

```
Syntax error
```

A syntax error was detected in a command. Check the *Editor Commands* part of this section for correct editor command syntax.

```
Target not reached
```

Are you sure? The *delete* command was used but the target could not be located. If you want the delete to proceed to the end of the edit buffer, answer this prompt with a *y*. Answering with an *n* or the end-of-file signal will cause the delete to be aborted.

```
Task ttt: Abnormal Termination
```

Interrupt code: i The background task *ttt* generated by the *x* command was interrupted before it could complete. The interrupt code returned by the task is indicated by *i*. This message is returned only in response to the *wait* command.

```
Task ttt: Abnormal Termination
```

Termination response: xxx The background task *ttt* generated by the *x* command has completed abnormally. The termination response returned by the task is indicated by *xxx*. This message is returned only in response to the *wait* command.

```
Task ttt initiated
```

Task number *ttt* has been started by the use of the *x* command.

```
Task ttt: Normal termination
```

The background task *ttt* generated by the *x* command has completed normally. This message is returned only in response to the *wait* command.

```
Too many file names specified
```

More than two file names were specified as arguments to the editor. This is an informative message only; the extra file names and any options specified after them are ignored.

```
Unable to open file
```

The file specified in a *read* command could not be found or could not be opened for reading because of its permissions.

```
Unexpected error, edit session aborted
```

An error response that the editor is incapable of handling was received from a system call . The editing session is terminated immediately.

```
Unknown option specified
```

An unrecognizable option was specified when the editor was called. This is an informative message only; the unrecognizable option is ignored.

```
Write ends with an error
```

The operating system reported a media error while the editor was writing data to the file specified in a *write* command.

```
zones OK?
```

A target could not be found and the search zones were not set to their default values. This is an informative message asking you to check the zones because they may have been the reason that the target could not be found. This message does not require a response from you.

```
?
```

The editor is not able to interpret the given command. Either the command could not be recognized or the format of the command was undecipherable.

Section 5
TERMINAL EMULATION

OVERVIEW

When working on the 4400 series you type on a keyboard and see messages displayed on a screen, just as with any terminal. When using *remote*, the terminal emulator program, you can think of the entire 4400 series as a terminal that is connected through an RS-232C line to a remote host computer. When you are using the 4400 series as a stand-alone computer, you can think of the keyboard and display as a local terminal connected to the 4400 processor.

The 4400 appears to both the host and to it's internal software as an ANSI X3.64 compatible terminal with a few extensions that make it more compatible with other common ANSI X3.64 terminals.

The terminal emulator itself is a local terminal emulator which talks to the 4400 operating system's console driver. In conjunction with a local communication utility called *remote*, the local terminal emulator, console driver, and the driver for the communications port combine to create a remote terminal emulator connected to the RS-232 hardware and device driver. This makes the entire unit appear to an external host as a terminal.

This section contains a brief description of the appearance of the ANSI terminal emulator, a discussion of the interface between the emulator and the operating system, information on its default modes, and a description of how non-ASCII keys are handled. The section is concluded by a list and short description of all the implemented ANSI commands.

Description

The terminal emulator supports a display of 32 lines of 80 characters per line, using 8 by 15 pixel characters.

Compliance With ANSI and ISO Standards

The ANSI terminal emulator complies with the following ANSI (American National Standards Institute) and ISO (International Standards Organization) standards:

ANSI X3.4-1977,
American National Standard Code for Information Interchange. (This defines the ASCII character set.)

ANSI X3.41-1974,
American National Standard Code Extension Techniques for Use With the 7-Bit Coded Character Set of American National Standard Code for Information Interchange. (This defines ways to extend the ASCII character set, including the exact way the SO and SI characters work to invoke G0 and G1 character sets.)

ISO 2022,
Code Extension Techniques for use with the ISO 7-bit Coded Character Set. (This is the international standard which corresponds to ANSI X3.41.)

ANSI X3.64-1979,
Additional Controls for Use With American Standard Code for Information Interchange. (This defines a variety of standard commands used for displaying text, editing the display of text, and for other functions.)

Compatibility with the DEC VT-100

The ANSI terminal emulator is NOT intended to emulate the VT-100. Some VT-100 DEC-private features which are of use to host editors have been included, but other DEC-private features have been omitted. Therefore, not all programs which run correctly with a VT-100 will run correctly with a 4400 series product.

Compatibility with Tektronix Terminals

The ANSI terminal emulator is also NOT intended to emulate any of the Tektronix 4100 Series terminals. Many of the 4100 Series ANSI mode commands have been included, but some have been intentionally omitted.

Interface to the Operating System

The interface to the 4400 series operating system is with the *ttyget/ttyset* system calls. These system calls are used to examine or modify the programmable modes of the emulator. This includes such things as autowrap on/off, screen normal/reverse, keypad application/numeric, cursor key application/numeric, LF/CR-LF, and tab locations.

The programmable modes of the emulator, mentioned above, all have default states which are specified in the discussion on ANSI commands. These defaults can be overridden by sending ANSI escape sequences to the terminal, or by using a *ttyset* system call (as in the *termset* utility).

The standard output of the non-ASCII keys on the keyboard (the function keys, the break-key, the keypad keys, and the joydisk) is an ANSI escape sequence (see the discussion on non-ASCII keys).

SUPPORTED ANSI COMMANDS

The following ANSI commands are supported on the 4400 terminal emulator:

NOTE

The ANSI <CSI> (control sequence identifier) is the two character sequence <Esc [>. In this discussion, it is represented as ESC [.

<ACK> Acknowledge Character (#6)

Syntax Form: (char #6)

Description: This control function (CTRL-F) is a no-op.

If this control character is received during an ANSI command sequence this control action is a no-op and the ANSI command sequence processing continues.

<BEL> Bell Character

Syntax Form: (char #7)

Description: Sounds the terminal's bell. (CTRL-G)

If this control character is received during an ANSI command sequence this control action occurs and the ANSI command sequence processing continues.

<BS> Backspace Character

Syntax Form: (char #8)

Description: The control function BS, (CTRL-H) moves the active position backward by one character position. If the cursor is already at column 1, then BS has no effect.

If this control character is received during an ANSI command sequence this control action occurs and the ANSI command sequence processing continues.

<CAN> Character (#24)

Syntax Form: (Char #24)

Description: If this control character (CTRL-X) is received during an ANSI command sequence this control function will print a snoopy <CAN> character and resets the command parser to an initialized state.

<CBT> Cursor Backward Tab

Syntax Form: ESC [<Pn> Z

Descriptive Form: ESC [<desired number of preceding tab stops> Z

Description: Moves the cursor backwards to a preceding tab stop on the current line.

A parameter value of one moves the cursor to the preceding tab stop. A parameter value greater than one (n) moves the cursor to the *n*th preceding tab stop on the current line. If there are less than n preceding tab stops, the cursor moves to column 1 of the current line.

If the parameter is zero or omitted, it defaults to 1.

<CHT> Cursor Horizontal Tab

Syntax Form: ESC [<Pn> I

Descriptive Form: ESC [<desired number of succeeding tab stops> I

Description: Moves the cursor forward to a succeeding tab stop on the current line.

A parameter value of one moves the cursor to the next tab stop. A value greater than one (n) moves the cursor to the *n*th next tab stop on the current line. If there are less than n following tab stops, the cursor moves to the rightmost column of the current line.

If the parameter is zero or omitted, it defaults to 1.

<CPR> Cursor Position Report

Syntax Form: ESC [<Pn> ; <Pn> R

Descriptive Form: ESC [<row> ; <column> R

Description: The <CPR> message is sent from the terminal to the host in response to a <DSR: 6> *device status report* command.

If the origin mode is relative, the coordinates reported are *row, column* coordinates in the scrolling region. *Row 1, column 1* means the upper left corner of the region.

If the origin mode is absolute, the coordinates reported are *row, column* coordinates of the screen. *Row 1, column 1* means the upper left corner of the screen.

If the <CPR> is echoed back to the terminal, the terminal treats the echo as a no-op.

<CR> Carriage Return Character

Syntax Form: (char #13)

Description: Moves the cursor to the first column in the current line. If *carriage return/line feed* (CR/LF) mode is set, then a line feed action is also performed.

If this control character (CTRL-M) is received during an ANSI command sequence this control action occurs and the ANSI command sequence processing continues.

<CRM> Control Representation Mode

Syntax Form: ESC [3 h or l

Descriptive Form: ESC [3 set or reset

Description: <CRM> is a parameter of the <RM> and <SR> commands.

This command is commonly referred to as a "snoopy" mode.

Reset: Normal operation. <RM: CRM> resets this mode.

NOTE

The implementation of this command in the 4400 requires that <RM: CRM> not be embedded with other <RM> commands.

Set: *Snoopy* mode. CRM is set <SM: CRM>, commands are not interpreted, but rather the characters that make up the command are displayed.

Defaults: Reset

<CUB> Cursor Backward

Syntax Form: ESC [<Pn> D

Descriptive Form: ESC [<number of columns> D

Description: Moves the cursor backward by the specified number of columns. The cursor stops at column 1.

If the numeric parameter is 0 or is omitted, it defaults to 1.

<CUD> Cursor Down

Syntax Form: ESC [<Pn> B

Descriptive Form: ESC [<number of rows> B

Description: Moves the cursor downward by the specified number of rows.

Margins Set Inside Screen Boundaries (i.e., Top Margin >1 or Bottom Margin <32)

If origin mode is absolute, the cursor moves with respect to the screen. If the cursor is on the last row of the screen or on the Bottom Margin, Cursor Down has no effect.

If origin mode is relative, the cursor moves with respect to the area bounded by Top and Bottom Margins. If the cursor is on the Bottom Margin, Cursor Down has no effect.

Margins Set To Screen Boundaries (i.e., Top Margin =1 and Bottom Margin =32)

The cursor moves with respect to the screen. If the cursor is on the last row of the screen, Cursor Down has no effect.

If the <Pn> numeric parameter is zero or is omitted, it defaults to one.

<CUF> Cursor Forward

Syntax Form: ESC [<Pn> C

Descriptive Form: ESC [<number of columns> C

Description: Moves the cursor the specified number of columns to the right. The cursor stops at the rightmost column.

If the <Pn> numeric parameter is omitted, or is zero, it defaults to one.

<CUP> Cursor Position

Syntax Form: ESC [<Pn> <; <Pn> > H

Descriptive Form: ESC [<row number> <; <column number> > H

Description: Moves the cursor to a specified row and column. The cursor may stop at Top Margin, Bottom Margin and the top and bottom of the screen, depending on origin mode.

If a row or column coordinate is zero or is omitted, it defaults to one.

<CUU> Cursor Up

Syntax Form: ESC [<Pn> A

Descriptive Form: ESC [<number of rows> A

Description: This command is completely analogous to <CUD>, except that the cursor moves upward instead of downward.

<DA> Device Attributes

Syntax Form: ESC [<Pn> c

Description: A device sends this command with a parameter of 0 to the terminal asking it to identify the type of VT100 terminal it is. The 4400 sends the command ESC [? 1 ; 0 c back to the device which says it is a VT100 with no options.

NOTE

The 4400 does support the following features of the VT-100 Advanced Video Options *(see <SGR>):*

- *Bold*
- *Underline*
- *Reverse video*

If the device echoes this command back to the terminal, it is treated as a no-op.

If the parameter is omitted, it defaults to 0.

<DC1> Character (#17)

Syntax Form: (Char #17)

Description: If this control character (CTRL-Q) is received during an ANSI command sequence this control action is a no-op and the ANSI command sequence processing continues. However if flagging is set in the communications system to DC1/DC3 flagging; a flagging action will occur within the communications system.

<DC2> Character (#18)

Syntax Form: (Char #18)

Description: This control function (CTRL-R) is a no-op.

If this control character is received during an ANSI command sequence this control action is a no-op and the ANSI command sequence processing continues.

<DC3> Character (#19)

Syntax Form: (Char #19)

Description: This control function (CTRL-S) is a no-op.

If this control character is received during an ANSI command sequence this control action is a no-op and the ANSI command sequence processing continues. However if flagging is set in the communications system to DC1/DC3 flagging; a flagging action will occur within the communications system.

<DC4> Character (#20)

Syntax Form: (Char #20)

Description: This control function (CTRL-T) is a no-op.

If this control character is received during an ANSI command sequence this control action is a no-op and the ANSI command sequence processing continues.

<DCH> Delete Character

Syntax Form: ESC [<Pn> P

Descriptive Form: ESC [<number of characters> P

Description: Deletes the character at the cursor and possibly following characters depending on the parameter value. Any characters to the right of the deleted characters are moved left by the same number of character positions; thus the gap is filled.

Only characters on the current line are affected by this command.

If the parameter is zero, or is omitted, it defaults to one.

<DEL> Character (#127)

Syntax Form: (Char #127)

Description: This control function is a no-op.

If this control character is received during an ANSI command sequence this control action is a no-op and the ANSI command sequence processing continues.

<DL> Delete Line

Syntax Form: ESC [<Pn> M

Descriptive Form: ESC [<number of lines> M

Description: Deletes the current line and possibly succeeding lines, depending on the parameter.

All following lines are shifted in a block toward the line containing the cursor. The lines following the shifted portion are erased. The cursor does not change position.

If split-screen scrolling is in effect, this command only affects lines in the region that the cursor is currently in. (E.g., if the cursor is in the top fixed region, only the lines in the top fixed region are affected.)

If the parameter is zero, or is omitted, it defaults to one.

<DLE> Character (#16)

Syntax Form: (Char #16)

Description: This control function is a no-op.

If this control character is received during an ANSI command sequence this control action is a no-op and the ANSI command sequence processing continues.

<DMI> Disable Manual Input

Syntax Form: Esc ‘ (Char #27 and Char #96)

Description: Locks the keyboard. This command is equivalent to ANSI <SM: KAM>. (See also <EMI>.)

<DSR> Device Status Report

Syntax Form: ESC [Ps n

Description: This is a command from the host or a report from the terminal. Table 5-1 shows the meaning of various parameters.

Table 5-1
Parameter Meanings

Parameter	Parameter Meaning
0	Report from 4400. Ready, no malfunctions detected.
3	Report from 4400. Malfunction - retry.
5	Command from host. Please report status (using a DSR control sequence).
6	Command from host. Please report cursor position (using a cursor position report). See <CPR> command.

When the 4400 receives a DSR with a parameter value of 5, it always sends back a DSR with a parameter value of 0 or 3. When the 4400 receives a DSR with a parameter of 6, it always sends back a CPR report. When the 4400 receives a DSR with a parameter value of 0 or 3 (which could be the echo of a report it has sent to the host), it executes the <DSR: 0> or <DSR: 3> command as a no-op.

<ECH> Erase Character

Syntax Form: ESC [<Pn> X

Descriptive Form: ESC [<number of characters> X

Description: Erases the character at the cursor, and possibly succeeding characters, according to the parameter. The cursor location remains unchanged.

The effect of the <ECH> command is not confined to the current line. For example, if the cursor is in column 41, and an <ECH: 45> command is issued, the character at the active position is erased along with the next 39 characters on the current line and the first 5 characters of the next line.

<ED> Erase in Display

Syntax Form: ESC [<Ps> J

Descriptive Form: ESC [< 0 or 1 or 2 > J

0 = from cursor to end of screen, inclusive
1 = from start of screen to cursor, inclusive
2 = entire screen.

Description: Regardless of whether margins are set, the command erases with respect to the screen. Therefore, text in the scrolling region and fixed regions can be erased with the same command.

The cursor does not change position.

If the parameter is omitted, it defaults to 0.

<EL> Erase in Line

Syntax Form: ESC [<Ps> K

Descriptive Form: ESC [<0 or 1 or 2> K

0 = from cursor to end of line, inclusive
1 = from start of line to cursor, inclusive
2 = entire line

Description: Erases part or all of the current line, according to the parameter. The cursor does not change position.

If parameter is omitted, it defaults to 0.

<EM> Character (#25)

Syntax Form: (Char #25)

Description: This control function is a no-op.

If this control character is received during an ANSI command sequence this control action is a no-op and the ANSI command sequence processing continues.

<EMI> Enable Manual Input

Syntax Form: Esc b

Description: Unlocks the keyboard. This command is equivalent to ANSI <RM: KAM>

<ENQ> Character (#5)

Syntax Form: (Char #5)

Description: This control function is a no-op.

If this control character is received during an ANSI command sequence this control action is a no-op and the ANSI command sequence processing continues.

<EOT> Character (#4)

Syntax Form: (Char #4)

Description: This control function is a no-op.

If this control character is received during an ANSI command sequence, this control action is a no-op and the ANSI command sequence processing continues.

<ESC> Character (#27)

Syntax Form: (Char #27)

Description: This control function is the introduction character of an escape sequence or control sequence for the ANSI command parser.

If this control character is received during an ANSI command sequence, the ANSI command sequence parser processing is reinitialized.

<ETB> Character (#23)

Syntax Form: (Char #23)

Description: This control function is a no-op.

If this control character is received during an ANSI command sequence this control action is a no-op and the ANSI command sequence processing continues.

<ETX> Character (#3)

Syntax Form: (Char #3)

Description: This control function is a no-op.

If this control character is received during an ANSI command sequence this control action is a no-op and the ANSI command sequence processing continues.

<FF> Form Feed Character

Syntax Form: (char #12)

Description: Erase the screen.

<FS> Character (#28)

Syntax Form: (Char #28)

Description: This control function is a no-op.

If this control character is received during an ANSI command sequence this control action is a no-op and the ANSI command sequence processing continues.

<GS> Character (#29)

Syntax Form: (Char #29)

Description: This control function is a no-op.

If this control character is received during an ANSI command sequence this control action is a no-op and the ANSI command sequence processing continues.

<HT> Horizontal Tab Character

Syntax Form: (char #9)

Description: Advances the cursor forward on the current line to the next horizontal tab stop. If there are no horizontal tab stops to the right of the active position, the cursor moves to the rightmost column.

If this control character is received during an ANSI command sequence this control action occurs and the ANSI command sequence processing continues.

<HTS> Horizontal Tab Set

Syntax Form: ESC H

Description: Sets a tab stop at the current cursor location.

Defaults: Tab stops at columns 9, 17, 25, 33, 41, 49, 57, 65, and 73. Read from setup file on installation.

<HVP> Horizontal and Vertical Position

Syntax Form: ESC [<Pn> <; <Pn> > f

Descriptive Form: ESC [<row> <; <column> > f

Description: This command is identical to the <CUP>, Cursor Position command.

<ICH> Insert Character

Syntax Form: ESC [<Pn> @

Descriptive Form: ESC [<number of characters> @

Description: Inserts the specified number, (n), of erased character cells at the cursor position. The character currently at the cursor position and all other characters to the right of the cursor are shifted "n" columns to the right. Characters shifted off the end of the line are lost. The cursor position remains unchanged.

If the parameter is zero, or is omitted, it defaults to one.

<IL> Insert Line

Syntax Form: ESC [<Pn> L

Descriptive Form: ESC [<number of lines> L

Description: Inserts the specified number, (n), of blank lines in place of the active line.

The active line and all succeeding lines are shifted downwards. The last "n" lines of the scroll are lost. The cursor position does not change.

If split-screen scrolling is in effect, this command only affects lines in the region that the cursor is currently in. (E.g., if the cursor is in the scrollable (non-fixed) region, only the lines in the scrollable region are affected.)

If the parameter is zero or is omitted, it defaults to one.

<IND> Index

Syntax Form: ESC D

Description: Moves the active position down one line without affecting the character position on the line.

If the cursor is at the bottom margin, but is not at the bottom of the scroll, a scroll up function is performed. If the cursor is at the bottom margin and is also at the bottom of the scroll, a blank line is added to the bottom of the scroll and a scroll up is performed.

The cursor can index into the scrolling region from the top fixed region, but cannot index into bottom fixed region. An index on the last line of the bottom fixed region has no effect.

<IRM> Insertion/Replacement Mode

Syntax Form: ESC [4 h or l

Descriptive Form: ESC [4 set or reset

Description: <IRM> is a parameter for the <RM> and <SM> commands.

Reset: Normal operation. When a character is entered, it replaces any character already at the active position.

Set: Insert mode. As each character is entered, the text at the cursor position and to its right is moved one character cell to the right and the cursor advances to the next character cell. Any text which is shifted off the end of the line is lost.

Defaults: Reset

<KAM> Keyboard Action Mode

Syntax Form: ESC [2 h or l

Descriptive Form: ESC [2 set or reset

Description: A parameter for the <RM> and <SM> commands.

Reset: Resetting KAM enables the keyboard and is equivalent to issuing <EMI>.

Set: Setting KAM disables the keyboard and is equivalent to issuing <DMI>.

Defaults: Reset

<LF> Line Feed Character

Syntax Form: (char #10)

Description: If <LNM> mode is reset, then <LF> has exactly the same effect as the <IND> command; it advances the cursor to the same position on the following line of text. See the <IND> command description for details.

If <LNM> mode is set, then <LF> has the same effect as <CR> <IND>; it advances the active position to the first character position on the following line.

If this control character is received during an ANSI command sequence this control action occurs and the ANSI command sequence processing continues.

<LNM> Line-Feed/New-Line Mode

Syntax Form: ESC [20 h or l

Descriptive Form: ESC [20 set or reset

Description: A parameter for the <RM> and <SM> commands.

Reset: (LF) is equivalent to <IND>; goes down one line without changing character position within the line.

Set: (LF) is equivalent to <NEL> (which is equivalent to (CR)<IND>). Advances the cursor to the first character position of the next line of text.

Defaults: Reset

<NAK> Character (#21)

Syntax Form: (Char #21)

Description: This control function is a no-op.

If this control character is received during an ANSI command sequence this control action is a no-op and the ANSI command sequence processing continues.

<NEL> Next Line

Syntax Form: ESC E

Description: Moves the cursor to the start of the next line. Has the same effect as (CR)<IND> (or as (LF) when LNM is set).

<NUL> Character (#0)

Syntax Form: (Char #0)

Description: This control function is a no-op. If this control character is received during an ANSI command sequence this control action is a no-op and the ANSI command sequence processing continues.

<PU1> Private Use 1

Syntax Form: ESC Q

Description: This two-character sequence is used to introduce a private ANSI control sequence. It introduces all sequences which specify or request from 4400 reports on the state of the mouse buttons and the graphic cursor position.

<Report-Syntax-Mode>

Syntax Form: ESC # ! 0

Description: This command sends a 4100 series terminal terminal-settings-report> to the host on the status of the syntax mode. The form will always be the following:

```
% ! <SP> <SP> 1 <CR>
```

NOTE

The <SP> is an ASCII space character. The <CR> ASCII Carriage Return Character is the default 4100 series EOM character.

<RI> Reverse Index

Syntax Form: ESC M

Description: Completely analogous to the IND (Index) command except that it moves the cursor one line upward.

<RIS> Reset to Initial State

Syntax Form: ESC c

Description: Resets specified terminal attributes to their initial default states.

This command affects terminal attributes in the following way:

- Erases screen and moves cursor to home position.
- Resets Insert/Replace mode to Replace.
- Clears edit margins.
- Turns off the character graphic rendition.
- Selects the default G0 and G1 character sets.
- Shifts in the G0 character set.
- Resets Auto-Repeat (TEKARM) mode := true.
- Resets Auto-Wrap (TEKAWM) mode := true.
- Resets Screen mode (TEMSCNM) to normal.
- Sets Origin mode to relative.

<RM> Reset Mode

Syntax Form: ESC [<Ps> l

Description: Causes one or more modes to be reset, as specified by each selective parameter in the <Ps> parameter list. Each mode to be reset is specified by a separate parameter in the list. A mode is reset until set again by a <SM>, Set Mode, control sequence.

If the first character in the parameter list is *?*, then all subsequent parameters, that consist of numeric digits only, are interpreted as if they began with a *?* character before those numeric digits. If the first parameter consists ONLY of *?*, then its only use is to provide an implicit *?* at the start of each subsequent numeric-digits-only parameter in the parameter list.

For example:

The control sequence: **ESC [? 5 ; 8 l**
is interpreted as if it were: **ESC [? 5 ; ? 8 l**

The control sequence: **ESC [? ; 5 ; 8 l**
is interpreted as if it were: **ESC [? 5 ; ? 8 l**

Table 5-2, Valid Reset Mode Parameters, summarizes the meaning of the valid parameters.

Table 5-2
Valid Reset Mode Parameters

Parameter	Mode
2	KAM Keyboard-Action-Mode.
3	CRM Control-Representation-Mode.
4	IRM Insertion-Replacement mode.
1 2	SRM Send/Receive mode.
2 0	LNM Line-Feed/New-Line mode.
? 1	TEKCKM TEK private Cursor Key mode.
? 5	TEKSCNM TEK private Screen mode (normal)
? 6	TEKOM TEK private Origin Mode (viewport)
? 7	TEKAWM TEK private Auto-Wrap mode.
? 8	TEKARM TEK private Auto-Repeat mode.

Any parameters other than those specified here are recognized and ignored.

<RS> Character (#30)

Syntax Form: (Char #30)

Description: This control function is a no-op.

If this control character is received during an ANSI command sequence this control action is a no-op and the ANSI command sequence processing continues.

<SCS> Select Character Set

Syntax: <SCS> = <designate-G0-set> or <designate-G1-set>. <its designate-G0-set> = (ESC) (() <set-selector>. <designate-G1-set> = (ESC) ()) <set-selector>. <set-selector> = (A)or(B)or(0)or(1)or(2)or(3).

Description: Designates a particular character set as the G0 set or the G1 set.

Table 5-3, Character Set Selection, summarizes the escape sequences necessary to designate particular character sets.

Table 5-3
Character Set Selection

Escape Sequence to Designate a G0 Set	Escape Sequence to Designate a G1 Set	Character Set Being Designated As G0 Or G1
ESC (A	ESC) A	(no-op)
ESC (B	ESC) B	U.S. (ASCII)
ESC (0	ESC) 0	Rulings
ESC (1	ESC) 1	(no-op)
ESC (2	ESC) 2	(no-op)
ESC (3	ESC) 3	Supplementary

Defaults: On installation, the terminal emulator automatically designates the U.S. (ASCII) character set as its G0 and G1 character set.

<Select-Code>

Syntax Form: ESC % ! <code-selector>

Description: This control function is a no-op.

<SGR> Select Graphic Rendition

Syntax Form: ESC [<Ps-list> m

Parameters: The Ps-list consists of zero or more *Ps* selective parameters, separated by semicolons. Each parameter in the list specifies a graphic rendition for subsequent characters.

0	Default rendition. On the 4400, *default rendition* is: No underscore, normal boldness, standard (not reversed) image. That is, the effect of any preceding <SGR: 1>, <SGR: 4> or <SGR: 7> command is canceled.
1	Bold or increased intensity: The 4400 represents this by simulating a bold font (it paints each character twice, shifted one pixel horizontally).
4	Underscore.
7	Negative (reverse) image: white characters on black background.
21	Not bold. Cancels the effect of <SGR: 1>.
24	Not underlined. Cancels the effect of <SGR: 4>.
27	Positive image. Cancels the effect of <SGR: 7>.

Description: Invokes the graphic rendition specified by the parameters in the Ps-list parameter string. All following characters in the data stream are displayed according to the parameter(s) until the next occurrence of an <SGR> command in the data stream.

On Tektronix terminals, each occurrence of the <SGR> control function causes only those graphic rendition aspects to be changed that are specified by that <SGR>. All other graphic rendition aspects remain unchanged. (In other words, the GRAPHIC RENDITION COMBINATION MODE of ISO 6429 is always set to CUMULATIVE in Tektronix terminals.)

Defaults: An omitted parameter in the <Ps-list> defaults to zero. The state is that of <SGR: 0>.

<SI> Shift In Character

Syntax Form: (char #15)

Description: Invokes the current G0 character set.

If this control character is received during an ANSI command sequence, the G0 character is invoked and the ANSI command sequence processing continues.

Defaults: The G0 set is invoked.

<SM> Set Mode

Syntax Form: ESC [<Ps> h

Description: Causes one or more modes to be set, as specified by each selective parameter in the <Ps> parameter list. Each mode to be set is specified by a separate parameter. A mode is set until reset by a <RM> (Reset Mode) control sequence.

If the first character in the parameter list is *?*, then all subsequent parameters, that consist of numeric digits only, are interpreted as if they began with a *?* character before those numeric digits. If the first parameter consists ONLY of *?*, then its only use is to provide an implicit *?* at the start of each subsequent numeric-digits-only parameter in the parameter list.

For example:

The control sequence: **ESC [? 5 ; 8 h**
is interpreted as if it were: **ESC [? 5 ; ? 8 h**

The control sequence: **ESC [? ; 5 ; 8 h**
is interpreted as if it were: **ESC [? 5 ; ? 8**

Table 5-4, Set Mode Parameters, summarizes the meanings of the valid parameters to the *set mode* command.

Table 5-4
Set Mode Parameters

Par	Mode
2	KAM Keyboard-Action-Mode
3	CRM Control-Representation-Mode.
4	IRM Insertion-Replacement Mode.
1 2	SRM Send/Receive Mode.
2 0	LNM Line-Feed/New-Line Mode.
? 1	TEKCKM TEK private Cursor Key Mode.
? 5	TEKSCNM TEK private Screen Mode (Normal)
? 6	TEKOM TEK private Origin Mode (viewport)
? 7	TEKAWM TEK private auto-wrap mode.
? 8	TEKARM TEK private auto-repeat mode.

If no parameter is supplied, a parameter of zero is assumed. Any parameters other than those specified here (including zero) are recognized and ignored.

<SO> Shift Out Character

Syntax Form: SO = (char #14)

Description: Invokes the G1 character set. If this control character is received during an ANSI command sequence, the G1 character set is invoked and the ANSI command sequence processing continues.

Defaults: The G0 set is invoked (default is SO state).

<SOH> Character (#1)

Syntax Form: (Char #1)

Description: This control function (CTRL-A) is a no-op.

If this control character is received during an ANSI command sequence this control action is a no-op and the ANSI command sequence processing continues.

<SP> Space Character

Syntax Form: (char #32)

Description: SP functions as an ordinary graphic character. Spaces replace any characters already in the locations where the spaces are typed.

<SRM> Send/Receive Mode

Syntax Form: ESC [1 2 h or l

Descriptive Form: ESC [1 2 set or reset

Description: <SRM>, Send/Receive Mode, is not a command in its own right. Rather, it is a parameter for the <SM>, Set Mode, and <RM>, Reset Mode, commands.

Resetting SRM mode turns the terminal's local echo on. (In the standards documents, this is called *monitor send/receive mode.*)

Setting SRM mode turns the local echo off. (In the standards documents, this is *simultaneous send/receive mode.*)

Defaults: Reset

<STX> Character (#2)

Syntax Form: (Char #2)

Description: This control function is a no-op.

If this control character is received during an ANSI command sequence this control action is a no-op and the ANSI command sequence processing continues.

<SUB> Character (#26)

Syntax Form: (Char #26)

Description: If this control character is received during an ANSI command sequence this control function will print a SUB) character and reset the command parser to an initialized state.

<SYN> Character (#22)

Syntax Form: (Char #22)

Description: This control function is a no-op.

If this control character is received during an ANSI command sequence this control action is a no-op and the ANSI command sequence processing continues.

<TBC> Tabulation Clear

Syntax Form: ESC [<Ps> g

Descriptive Form: ESC [<0 or 2 or 3> g

Description: Clears one or more tab stops, according to the specified parameters.

Valid Parameters:

0 Clear the horizontal tab stop at the active position.

2 Clear all tab stops in the active line. (In the 4400 *tab stop mode* is always reset, so <TBC: 2> has the same effect as <TBC: 3>.)

3 Clear all horizontal tab stops.

If no parameter is supplied, a parameter of zero is assumed. If the supplied parameter is not 0, 2 or 3, then command is ignored.

<TEKARM> Auto-Repeat Mode

Syntax Form: ESC [? 8 h or l

Descriptive Form: ESC [? 8 set or reset

Description: A TEK private parameter for the <SM> and <RM> commands. If set, all keyboard keys repeat when held depressed. If reset, none of the keys repeat when held depressed.

Defaults: Reset

<TEKAWM> Auto-Wrap Mode

Syntax Form: ESC [? 7 h or l

Descriptive Form: ESC [? 7 set or reset

Description: A TEK-private parameter for the <SM> and <RM> commands. When set, the wrap-around feature is enabled. When reset, it is disabled.

This mode determines what happens to the cursor after a character is displayed in the rightmost column. Since a character is always displayed at the current cursor location, this mode determines whether text is overprinted in the rightmost column or whether it wraps around to the next lines.

If Auto-Wrap mode is set, an index function is performed and the cursor moves to column 1 of the next line. If Auto-Wrap mode is reset, the cursor remains in the rightmost column.

Defaults: Reset

<TEKBKCM> Block Cursor Mode (Select Cursor)

Syntax Form: ESC [> 31 h or l

Descriptive Form: ESC [> 31 set or reset

Description: A TEK private parameter for the <SM> and <RM> commands. If set, the alpha cursor is represented by a "full character set" block cursor. The character at the cursor position appears as a background color character on a reversed character cell. If reset, the cursor is represented by a bold underline.

Defaults: Reset (Underscore)

<TEKBNCM> Blinking Cursor Mode

Syntax Form: ESC [> 32 h or l

Descriptive Form: ESC [> 32 set or reset

Description: A TEK private parameter for the <SM> and <RM> commands. If set, the alpha cursor is displayed blinking. The character at the cursor position alternates, at the blink rate, between its normal rendition and its cursor position rendition. If reset, the cursor does not blink.

Defaults: Set (Blinking)

<TEKCKM> Cursor Key Mode

Syntax Form: ESC [? 1 h or l

Descriptive Form: ESC [? 1 set or reset

Description: A TEK private parameter for the <SM> and <RM> commands. Provides compatibility with programs designed for the DEC VT-100 terminal. This mode is only effective when TEKKPAM is set.

The joydisk keys assume the alternate meanings shown in Table 5-5.

Table 5-5
Alternate Joydisk Meanings

Joydisk Key	TEKCKM Reset	TEKCKM Set
Up	ESC [A	ESC O A
Down	ESC [B	ESC O B
Right	ESC [C	ESC O C
Left	ESC [D	ESC O D

Defaults: Reset

<TEKGCREP> Graphic Cursor Position Report

Syntax Form: ESC P S <Pn1 ; Pn2> ESC \ e

Descriptive Form: DCS S [optional position report] ST

Description: This is a report string sent to the host in response to a <TEKRGCR> graphic cursor position request. The form which the optional position report takes depends on the report types specified by a <TEKSGCRT> report type selection, or if report types have not been specified, by the default types defined there.

If cell coordinate reports have been specified, then Pn1 and Pn2 contain row and column values, respectively. If pixel coordinate reports have been specified, then Pn1 and Pn2 contain x and y screen coordinate values, respectively.

If graphics cursor position reports have been disabled by specifying that none be returned, then no parameters are returned for Pn1 and Pn2.

<TEKID> Identify Terminal

Syntax Form: ESC Z

This command, when sent from the host requests the terminal to identify itself with a Device Attributes sequence. It has the same effect as a <Device Attributes> command with no parameter or parameter of 0.

<TEKKPAM> Keypad Application Mode

Syntax Form: ESC =

Description: See <TEKKPNM>, Keypad Numeric Mode

<TEKKPNM> Keypad Numeric Mode

Syntax Form: ESC >

Description: The <TEKKPAM> and <TEKKPNM> commands set and reset the terminal's *Keypad Application Mode*, respectively. These commands are provided for compatibility with applications programs designed for the DEC VT100 terminal.

Reset State (Keypad Numeric Mode)
In the *reset* state (Keypad Numeric Mode), the keypad keys and function keys F9 to F12 assume the values shown in the *reset state* part of the following table. For the keypad keys, these are the values labeled on the keys, except that the ENTER key sends a <CR> character.

Set State (Keypad Application Mode)
In the *set* state (Keypad Application Mode), the keypad keys and function keys F9 to F12 assume the alternate meanings shown in the *set state* part of Table 5-6.

Table 5-6
Keypad Application Mode Key Meanings

Key	Meaning in *Reset* State Keypad Numeric Mode	Meaning in *Set* State Keypad Application Mode
0	0	ESC O p
1	1	ESC O q
2	2	ESC O r
3	3	ESC O s
4	4	ESC O t
5	5	ESC O u
6	6	ESC O v
7	7	ESC O w
8	8	ESC O x
9	9	ESC O y
-	-	ESC O m
,	,	ESC O l
.	.	ESC O n
ENTER	CR	ESC O M
F9	ESC O P	ESC O P
F10	ESC O Q	ESC O Q
F11	ESC O R	ESC O R
F12	ESC O S	ESC O S

Defaults: Reset (Keypad Numeric Mode)

<TEKMBREP> Mouse Button and Graphic Cursor Position Reporting

Syntax Form:

DCS (Esc P)
Meta-State-Code
Mouse-Button-Number
Stroke-Info (up-down)
Optional-Position-Report
ST (Esc \)

Description: There are three buttons on the mouse and there are different codes output for each button on it's down-stroke and up-stroke.

Table 5-7 summarizes the Mouse Button Reports.

Table 5-7
Mouse Button Reports

Button	Left	Middle	Right
DOWN	DCS A 1 D x ST	DCS A 2 D x ST	DCS A 3 D x ST
UP	DCS A 1 U x ST	DCS A 2 U x ST	DCS A 3 U x ST
Shifted-DOWN	DCS B 1 D x ST	DCS B 2 D x ST	DCS B 3 D x ST
Shifted-UP	DCS B 1 U x ST	DCS B 2 U x ST	DCS B 3 U x ST
Control-DOWN	DCS C 1 D x ST	DCS C 2 D x ST	DCS C 3 D x ST
Control-UP	DCS C 1 U x ST	DCS C 2 U x ST	DCS C 3 U x ST
Cntrl-Shifted-DOWN	DCS D 1 D x ST	DCS D 2 D x ST	DCS D 3 D x ST
Cntrl-Shifted-UP	DCS D 1 U x ST	DCS D 2 U x ST	DCS D 3 U x ST

The *x* information is the optional report of the current graphic cursor position, i.e. *Pn1 ; Pn2* of the Graphic Cursor Report. See <TEKGCREP>.

ANSI Terminal Emulator Mouse Button and Position Reporting

Each of the three buttons on the mouse reports a different code on its downstroke and its upstroke. The mouse reports are ANSI standard DCS (Device Control String — Esc-P) reports. The reports take the form:

DCS (Esc-P)	Lead-in to all mouse button reports
Meta-State-Code	A = unshifted, B = shifted, C = control, and D = control-shift
Mouse-Button-Number	1 = left, 2 = middle, 3 = right
Stroke-Info (up-down)	D = down, U = up
Optional-Position-Report	Pn1, Pn2 of the current position of the graphic cursor.
ST (ESC-\)	Terminator for mouse button and position reports.

For example, the report (32;80 is the position report of Row 32, Column 80 in Char.Cell coordinates)) of the unshifted, middle button, in the down state would be:

```
DCS A 2 D 32;80 ST
```

<TEKOM> Origin Mode

Syntax Form: ESC [? 6 l or h

Parameters: l - Reset (Absolute Mode)
h - Set (Relative Mode)

Description: Margins Set To Screen Boundaries (that is, Top Margin = 1, and Bottom Margin = 32)

Specifies Row 1, Column 1 of the screen as the origin. Moves the cursor to the origin.

Margins Set Inside Screen Boundaries (i.e., Top Margin >1 or Bottom Margin < 32)

If origin mode absolute is requested, specifies Row 1, Column 1 of the screen as the origin. If origin mode relative is requested, specifies the row corresponding to the Top Margin, Column 1 as the origin. In both cases, it moves the cursor to the origin.

Defaults: Reset

<TEKRC> Restore Cursor

Syntax Form: ESC 8

Description: Restores the previously saved cursor position, graphic rendition, character set and origin mode.

If no preceding <Save Cursor> command has been executed, then the power-up graphic rendition, character set, and origin mode are restored and the cursor is homed.

<TEKREQTPARM> Request Terminal Parameters

Syntax Form: ESC [<Pn> x

Description: Request from the host for the terminal to send a <Report Terminal Parameters> sequence. This command is treated as a no-op in the 4400.

<TEKRGCR> Request Graphic Cursor Position Report

Syntax Form: ESC Q K

Description: This command requests the terminal to send a report to the host as to the position of the graphics cursor. This report is a <TEKGCREP> report. The form of the report is as follows:

DCS (ESC P) S Pn1 ; Pn2 ST (ESC-\)

Pn1 contains: The Row value if cell coordinates have been selected or the X value if pixel coordinates are selected. no parameter will be returned if so specified in the Select Graphic Cursor Report Type command.

Pn2 contains: The Column value if cell coordinates have been selected or the Y value if pixel coordinates are selected. no parameter will be returned if so specified in the Select Graphic Cursor Report Type command.

<TEKSC> Save Cursor

Syntax Form: ESC 7

Description: Saves the cursor position, graphic rendition, character set and origin mode.

<TEKSCNM> Screen Mode

Syntax Form: ESC [? 5 l or h

Parameters: l - Reset (Normal Mode — white on black)
h - Set (Reverse Mode — black on white)

Description: This is a parameter for the <Set Mode> and <Reset Mode> commands.

The reset state causes the screen to be black with white characters. The set state causes the screen to be white with black characters.

There is no effect if the terminal is already in the requested mode.

Defaults: Reset

<TEKSGCRT> Select Graphic Cursor Report Type

Syntax Form: ESC Q <Pn1> <;<Pn2>> J

Descriptive Form: ESC Q <Report When> <;<Report Type>> J

Parameter	Parameter Meaning
Pn1 = 0	None. Do not report mouse button action.
Pn1 = 1	Down. Report to host when mouse button is depressed.
Pn1 = 2	Up. Report to host when mouse button is released.
Pn1 = 3	Both. Report to host when a mouse button is either depressed or released.
Pn2 = 0	None. Do not report graphic cursor position.
Pn2 = 1	Char. Report graphics cursor position in character cell coordinate terms (Row, Column).
Pn2 = 2	Pixel. Report graphics cursor position in pixel (screen) coordinate terms (X,Y).
Defaults:	
	Pn1 = 0: No mouse button report. Pn2 = 1: Report graphic cursor position in character cell coordinates

<TEKSTBM> Set Top and Bottom Margins

Syntax Form: ESC [<Pn> <; <Pn>> r

Descriptive Form: ESC [<top margin> <; <bottom margin>> r

Description: A TEK private command to set top and bottom margins for a split viewport scrolling region.

The parameter value for the top margin specifies which row of the screen becomes the top line of the scrolling region. Similarly, the value for the bottom margin specifies the row of the buffer for the bottom line of the scrolling region.

The rows preceding the top margin and the rows following the bottom margin become fixed regions. No scrolling actions occur in the fixed regions.

If the first parameter is zero or is omitted, it defaults to one. If the second parameter is zero or is omitted, it defaults to 32.

Defaults: Margins set to 1 and 32

<US> Character (#31)

Syntax Form: (Char #31)

Description: This control function is a no-op.

If this control character is received during an ANSI command sequence this control action is a no-op and the ANSI command sequence processing continues.

<VT> Vertical Tab Character

Syntax Form: (char #11)

Description: VT has the same effect as (LF), linefeed.

If this control character is received during an ANSI command sequence this control action occurs and the ANSI command sequence processing continues.

KEYBOARD DETAILS

Shift, Ctrl, and Caps Lock Keys

The two SHIFT keys have identical functions. They and the CTRL key are used to access alternate meanings for other keys.

Pressing CAPS LOCK turns on the led in the key and puts the keyboard in *caps lock mode*. Pressing the key again turns the led off and removes the terminal from caps lock mode. While in caps lock mode, each of the alphabetic keys has its uppercase meaning, regardless of whether a SHIFT key is being held down. Caps lock mode affects only the alphabetic keys.

Default ANSI Mode Meanings of Keys

Alphanumeric Keys

Table 5-8 shows the ANSI mode meanings for the main part of the keyboard — the *alphanumeric keys*.

In this table, control characters are represented by the standard two-or three-letter abbreviations, given in ANSI X3.4 and ISO 646. Special symbols are represented by the four-character codes assigned to those symbols in ISO 6937.

Table 5-8
ANSI Meanings of Alphanumeric Keys

	Key Name	Shifted	Unshifted	Control	CTRL-Shifted
Keyboard Row 1	{ [	{	[	ESC	ESC
	! 1	!	1	1	!
	@ 2	@	2	2	@
	# 3	#	3	3	#
	$ 4	$	4	4	$
	% 5	%	5	5	%
	^ 6	^	6	6	^
	& 7	&	7	7	&
	* 8	*	8	8	*
	(9	(	9	9	(
	) 0	)	0	0	)
	· –	_	–	–	US
	+ =	+	=	=	+
	}]	}	]	]	}
	RUB	DEL	DEL	DEL	DEL

Table 5-8 (cont.)
ANSI Meanings of Alphanumeric Keys

	Key Name	Shifted	Unshifted	Control	CTRL-Shifted
Keyboard Row 2	ESC	ESC	ESC	ESC	ESC
	~ |	~	|	|	~
	Q	Q	q	DC1	DC1
	W	W	w	ETB	ETB
	E	E	e	ENQ	ENQ
	R	R	r	DC2	DC2
	T	T	t	DC4	DC4
	Y	Y	y	EM	EM
	U	U	u	NAK	NAK
	I	I	i	HT	HT
	O	O	o	SI	SI
	P	P	p	DLE	DLE
	` \	`	\	FS	FS
	BS	BS	BS	BS	BS
	LF	LF	LF	LF	LF

Table 5-8 (cont.)
ANSI Meanings of Alphanumeric Keys

	Key Name	Shifted	Unshifted	Control	CTRL-Shifted
Keyboard Row 3	TAB	HT	HT	HT	HT
	A	A	a	SOH	SOH
	S	S	s	DC3	DC3
	D	D	d	EOT	EOT
	F	F	f	ACK	ACK
	G	G	g	BEL	BEL
	H	H	h	BS	BS
	J	J	j	LF	LF
	K	K	k	VT	VT
	L	L	l	FF	FF
	: ;	:	;	;	:
	" '	"	'	'	"
	RTN	CR	CR	CR	CR

Table 5-8 (cont.)
ANSI Meanings of Alphanumeric Keys

	Key Name	Shifted	Unshifted	Control	CTRL-Shifted
Keyboard Row 4	Z	Z	z	SUB	SUB
	X	X	x	CAN	CAN
	C	C	c	ETX	ETX
	V	V	v	SYN	SYN
	B	B	b	STX	STX
	N	N	n	SO	SO
	M	M	m	CR	CR
	< ,	<	,	,	<
	> .	>	.	.	>
	? /	?	/	/	?

Row 5 Keys — Spacebar is *space* in all states.

Numeric Pad Keys

The numeric pad is located to the right of the main set of alphanumeric keys. The codes sent by these keys are determined by the state of the Keypad Numeric/Applications mode setting (TEKKPNM/TEKKPAM). In Numeric mode, the meaning of the keys is that marked on the keytops; in Applications mode, the numeric pad keys are defined to be a control sequence. Table 5-9 shows the Applications mode (TEKKPAM) ANSI meanings of these keys.

Table 5-9
Applications Mode (TEKKPAM) Meanings of Keypad Keys

Key	Key Pad State			
	Shifted	Unshifted	Control	Ctrl-Shifted
0	ESC O p	ESC O p	ESC O p	ESC O p
1	ESC O q	ESC O q	ESC O q	ESC O q
2	ESC O r	ESC O r	ESC O r	ESC O r
3	ESC O s	ESC O s	ESC O s	ESC O s
4	ESC O t	ESC O t	ESC O t	ESC O t
5	ESC O u	ESC O u	ESC O u	ESC O u
6	ESC O v	ESC O v	ESC O v	ESC O v
7	ESC O w	ESC O w	ESC O w	ESC O w
8	ESC O x	ESC O x	ESC O x	ESC O x
9	ESC O y	ESC O y	ESC O y	ESC O y
-	ESC O m	ESC O m	ESC O m	ESC O m
,	ESC O l	ESC O l	ESC O l	ESC O l
.	ESC O n	ESC O n	ESC O n	ESC O n
ENT	ESC O M	ESC O M	ESC O M	ESC O M

Joydisk Keys

The joydisk is located to the upper left of the main set of alphanumeric keys. The function of the joydisk in ANSI mode is to act in the place of cursor keys. The codes sent by the joydisk are affected by the Cursor Key mode in union with the Keypad Applications mode. The default codes are sent unless both TEKKPAM and TEKCKM are set. Table 5-10 shows the ANSI mode meanings of its keys.

Table 5-10
ANSI Joydisk Key Meanings

Position	Default Mode				TEKPAM and TEKCKM Mode			
	Shifted	Unshifted	CTRL	CTRL-Shifted	Shifted	Unshifted	CTRL	CTRL-Shifted
Up	ESC [A	ESC [A	ESC [A	ESC [A	ESC O !	ESC O A	ESC O 1	ESC O 1
Down	ESC [B	ESC [B	ESC [B	ESC [B	ESC O "	ESC O B	ESC O 2	ESC O 2
Right	ESC [C	ESC [C	ESC [C	ESC [C	ESC O #	ESC O C	ESC O 3	ESC O 3
Left	ESC [D	ESC [D	ESC [D	ESC [D	ESC O $	ESC O D	ESC O 4	ESC O 4

Function Keys

The function keys F1-F12 are grouped in three groups of four keys and are located in a row above both the alphanumeric keys and the numeric key pad. Table 5-11 shows the ANSI mode meanings of these keys.

Table 5-11
ANSI Meanings of FUnction Keys

Function Key	State			
	Shifted	Unshifted	Ctrl	Ctrl-Shifted
F1	ESC O %	ESC O E	ESC O 5	ESC O 5
F2	ESC O &	ESC O F	ESC O 6	ESC O 6
F3	ESC O ´	ESC O G	ESC O 7	ESC O 7
F4	ESC O (	ESC O H	ESC O 8	ESC O 8
F5	ESC O)	ESC O I	ESC O 9	ESC O 9
F6	ESC O *	ESC O J	ESC O :	ESC O :
F7	ESC O +	ESC O K	ESC O ;	ESC O ;
F8	ESC O,L	ESC O L	ESC O <	ESC O <
F9	ESC O P	ESC O P	ESC O P	ESC O P
F10	ESC O Q	ESC O Q	ESC O Q	ESC O Q
F11	ESC O R	ESC O R	ESC O R	ESC O R
F12	ESC O S	ESC O S	ESC O S	ESC O S

Special Function Keys

There are only two special function keys on the 4400 keyboard. One is the *up-arrow/left-arrow* key in the upper left corner of the main key area, while the other is the BREAK key in the lower right corner of the main key area. While most terminal emulators do not send a character sequence when the BREAK key is pressed, this emulator does — under the assumption that the communication program will recognize the sequence and perform the appropriate break signal. Table 5-12 shows the default ANSI mode meaning of these keys.

Table 5-12
ANSI Meanings of Special Function Keys

Key Names	Shifted	Unshifted	Ctrl	Ctrl-Shifted
↑	ESC O U	ESC O T	ESC O U	ESC O T
Break	ESC O @	ESC O @	ESC O @	ESC O @

Section 6

ACCESSING SYSTEM RESOURCES

INTRODUCTION

You can access the 4400 series hardware directly, but in general, this tends to be cumbersome and error-prone. The operating system has embedded within it *device drivers*, or software routines that offer a uniform interface to the operating system. Most programs should interface with the 4400 series hardware through these device drivers.

This section discusses the device drivers and system calls to the 4400 series hardware.

DEVICE DRIVERS

Device drivers are divided into two types: block-oriented and character-oriented. Block-oriented devices include the disks and other peripherals on the SCSI bus. Character-oriented devices include the console, the communications port, the sound generator, the printer port, the optional network interface, and special devices for "raw" access to the block-oriented devices. Each of these devices is identified by a file in the */dev* directory.

The system calls *ttyget* and *ttyset* can be used with the console, communications port, sound, and printer devices. Descriptions of the parameters to these calls are found in Section 4, *System Calls*, of the 4400 Series Assembly Language Reference.

SCSI Peripherals

A SCSI bus gives access to the block-oriented devices. These devices include such things as winchester disks, floppy disks, and (optional) streaming tape drives.

The */dev* SCSI peripheral devices are:

- disk. The winchester disk with the system files.
- disk1..diskn. Optional winchester disks.
- floppy. Floppy disk drive.

The standard 4400 series contains a single floppy disk (*/dev/floppy*) and a winchester disk mass storage device (*/dev/disk*). Option 20 contains an additional 40 Megabyte winchester disk and a streaming tape drive.

Device */dev/disk* is the standard system device and is the default device from which to boot the system. You must use the interactive boot procedure to boot from another device.

Console Device

The device */dev/console* supports the 4400 display and keyboard. It is connected to a terminal emulator that acts like an industry-standard terminal (described in Section 5 of this manual).

To read and write terminal settings and other parameters of this device, use the *ttyget* and *ttyset* system calls, or the *conset* utility.

Communications Port

The device */dev/comm* supports the RS-232C host communications port. You can control the baud rate, number of stop bits, parity, and XON/XOFF or DTR flow control. You can also cause new input or completion of output to generate a signal, as well as read the number of characters pending in the input and output queues.

The *ttyget* and *ttyset* system calls, and the *commset* utility allow you to read and write the communications port parameters.

Sound Generator

The device */dev/sound* is the 4400 sound generator. By sending a formatted byte stream to this device (a TI 76496 sound generator chip), you can cause the 4400 to produce sounds.

This device is a write-only device. An attempt to read it will return an error. It is also an exclusive-use device and may be opened by only one task at a time.

The *ttyset* and *ttyget* system calls can change operation settings and examine device status.

Controlling the Sound Device

/dev/sound expects a stream of bytes in the following form:

```
\n,c,c,c . . . c,t

or

\0,tempo
```

with the following values:

`n`	A single byte specifying the number of commands to follow.
`c`	A single byte binary command to the sound chip. (See the following discussion on sound chip operation and commands.)
`t`	A byte value specifying the length of time to hold this set of commands. T is in units of tempo set by the second format.
`tempo`	A 16-bit (word) value of time with a unit value of 16.667 ms.

/dev/sound Operation and Commands

The sound chip contains three frequency generators, each coupled to a programmable attenuator. It also contains a white-noise generator (a shift register with an exclusive-OR feedback network) that contains a frequency control and programmable attenuator.

Frequency Control

Changing the value in a frequency generator requires two command bytes. Byte 1 contains the address information (which frequency generator to alter) and the low order 4 bits of the value to store. Byte 2 contains the high order 6 bits to set the frequency. Thus, the two bytes contain a 3-bit address and a 10-bit binary number to set the frequency to be generated.

The frequency is equal to the clocking rate of the chip (which is 3.15 MHz) divided by thirty-two times the binary number that is stored in the frequency generator.

Table 6-1 shows the bit assignments in Byte 1 and Table 6-2 shows the bit assignments in byte 2.

Table 6-1
Frequency Selection (BYTE 1)

Bit	Type	Description
0	1	This bit is always 1
1	R2	Register address bit 2
2	R1	Register address bit 1
3	R0	Register address bit 0
4	F3	Frequency data bit 3
5	F2	Frequency data bit 2
6	F1	Frequency data bit 1
7	F0	Frequency data bit 0

Table 6-2
Frequencey Selection (BYTE 2)

Bit	Type	Description
0	0	This bit is always 0
1	x	Unused
2	F9	Frequency data bit 9
3	F8	Frequency data bit 8
4	F7	Frequency data bit 7
5	F6	Frequency data bit 6
6	F5	Frequency data bit 5
7	F4	Frequency data bit 4

Controlling Attenuation

You can control the attenuation on any frequency generator with a single command byte. This byte contains a 3-bit field to select the attenuator and a 4-bit field to specify the amount of attenuation. Four bits give you 16 possible attenuations. Table 6-3 shows the attenuation settings and Table 6-4 shows the bit assignments for the attenuation control byte.

Table 6-3
Attenuation Control

A3	A2	A1	A0	Attenuation Weight (dB)
0	0	0	0	ON (Full Volume)
0	0	0	1	2
0	0	1	0	4
0	1	0	0	8
1	0	0	0	16
1	1	1	1	Off

Table 6-4
Attenuation Byte Bit Assignments

Bit	Type	Description
0	1	Always 1
1	R2	Register address bit 2
2	R1	Register address bit 1
3	R0	Register address bit 0
4	A3	Attenuation control bit 3
5	A2	Attenuation control bit 2
6	A1	Attenuation control bit 1
7	A0	Attenuation control bit 0

Controlling the Noise Generator

The noise generator consists of a noise source and an attenuator. You can control the type of feedback in the exclusive-OR network, the shift rate, and the attenuator itself.

Table 6-5 shows how you control the feedback, Table 6-6 shows the shift-rate control, and Table 6-7 shows the bit assignments for the noise generator command byte.

Table 6-5
Noise Feedback Control

FB	Configuration
0	Periodic noise
1	White noise

Table 6-6
Noise Frequency Control

NF1	NF0	Shift Rate
0	0	49218
0	1	24609
1	0	12304
1	1	Tone generator 3 output

Table 6-7
Noise-Control-Byte Bit Assignments

Bit	Type	Description
0	1	Always 1
1	R2	Register address bit 2
2	R1	Register address bit 1
3	R0	Register address bit 0
4	x	Unused
5	FB	Feedback control bit
6	NF1	Shift rate control bit 1
7	NF2	Shift rate control bit 0

Control Registers

The sound chip has eight internal registers that determine whether the byte(s) sent control the frequency or attenuation of the three tone generators or the control or attenuation of the noise generator. The destinations for all addressed bytes are given in Table 6-8.

Table 6-8
Control Register Addresses

R2	R1	R0	Address register
0	0	0	Tone 1 frequency
0	0	1	Tone 1 attenuation
0	1	0	Tone 2 frequency
0	1	1	Tone 2 attenuation
1	0	0	Tone 3 frequency
1	0	1	Tone 3 attenuation
1	1	0	Noise control
1	1	1	Noise attenuation

Sound Examples

The following examples show how you can control the sound device by sending bytes to it. The "C" program in example 1 outputs an array of bytes to the standard output device. Redirect the compiled program output with the command:

```
sound-example > /dev/sound
```

The data in the array controls the sound device output. To determine what the program data means to the sound output device, each byte is described in the text following the example. Using Tables 6-1 through 6-8, you can create your own sound effects or music from the 4400 series products.

```
/*************************************************************
*                                                            *
*           Sound generation routine sample.                 *
*                                                            *
*************************************************************/

#include <stdio.h>

  int buf[] = {0,0,59,/* Set the tempo          */
          2,175,13,0,/* Set the frequency      */
          1,176,1,   /* Set the volume         */
          1,188,2,   /* Reduce volume 2 beats  */
          1,191,1,   /* Turn voice off         */
          1,228,0,   /* Turn on white noise    */
          1,249,2,   /* Reduce volume 2 beats  */
                     /* Replace 249 with 244   */
                     /* for greater volume     */
          1,255,1};  /* Turn voice off         */
main()

{
  int i;
  for (i = 0;i<25 ;i++)/* Set up a FOR loop to  */
      putchar(buf[i]);/* Output the data in     */
                      /* the array              */

}
```

Example 6-1. Sound Example.

Set the tempo to be 1 beat per second (1000 millisec/beat):

```
Byte 1  =  0   "Tempo word follows."

Byte 2  =  0   "High order byte = (1000 div 16.667) div 256."

Byte 3  = 59   "Low order byte = (1000 div 16.667) mod 256."
```

Set the frequency for voice 2 to be 440 Hz:

```
Byte 4  =  2   "2 command bytes follow."

Byte 5  =175    1   0 1 0   1 1 1 1
                ___  _____  _______
    "Always 1" _|     |       |
                      |       |
          "Voice 2  __|       |
          frequency"          |
                              |
                 "Low order 4 bits calculated as
                  (3,150,000 div (32 * 440)) div 16."

Byte 6  = 13    0    0   0 0 1 1 0 1
                ___  ___  ___________
     "Always 0" _|    |       |
                      |       |
            "Unused" _|       |
                              |
                       "High order 6 bits:
                       (3,150,000 div(32 * 440)) div 16."

Byte 7  =  0   "Hold this set of commands 0 beats."
```

Play voice 2 at full volume for 1 beat:

```
Byte 8  =  1   "1 command follows."

 Byte 9  =176    1   0 1 1   0 0 0 0
                 ___  _____   _______
     "Always 1" _|     |       |
                       |       |
           "Voice 2   __|       |
            attenuation"        |
                                |
                    "Leave it all the way on."

Byte 10 =  1   "Play for 1 beat."
```

Turn the volume of voice 2 down by 12 dB and play for 2 beats:

```
Byte 11 =  1   "1 command byte follows."

Byte 12 =188    1   0 1 1   1 1 0 0
                ___  _____   _______
     "Always 1" _|     |       |
                       |       |
           "Voice 2   __|       |
            attenuation"        |
                                |
                  "Attenuate by 12 dB."

Byte 13 =  2   "Hold for 2 beats."
```

Turn voice 2 off so it won't play forever:

```
Byte 14 =  1   "1 command byte follows."

Byte 15 =191    1 0 1 1   1 1 1 1
                          _______
                             |
                   "Turn voice 2 off."

Byte 16 =  1   "Hold for 1 beat."
```

Play white noise (hissing sound):

```
Byte 17 =  1   "1 command byte follows"

Byte 18 =228    "1   1 1 0   0   1   0 0"
                 ___  _____  ___ ___  ___
     "Always 1" _|     |     |   |    |_ "Shift rate = 0 (least
                       |     |   |        coarse hissing sound)."
              "Noise  __|     |   |
               control"       |   |__ "White noise"
                              |
                          "Unused"

Byte 19 =  0   "Hold 0 beats."
```

Turn down the volume 18 dB and hold for 2 beats:

```
Byte 20 =  1   "1 command follows"

Byte 21 =249   1   1 1 1   1 0 0 1
               ___  _____   _______
   "Always 1" _|     |       |
                     |       |
            "Noise  __|       |
             attenuation"     |
                              |
                    "Attenuate by 18 dB."

Byte 22 =  2   "Hold 2 beats."
```

Turn noise off:

```
Byte 23 =  1   "1 command follows"

Byte 21 =255     1   1 1 1   1 1 1 1
                 ___  _____   _______
     "Always 1" _|     |       |
                       |       |
              "Noise  __|       |
               attenuation"     |
                                |
                      "Attenuate by 30 dB."

Byte 25 =  1   "Hold 1 beat."
```

Printer Port

The device */dev/printer* provides interface to the 4400 series parallel interface printer port. This port provides a Centronics-compatible parallel port that can drive most inexpensive dot-matrix (and some letter quality) printers.

/dev/printer accepts character streams and recognizes the ANSI X3.64 *Select Graphic Rendition* escape sequences for bold or italic characters.

These devices are write-only; any attempt to read them will return an error. They are exclusive-use devices and may be opened by only one task at a time.

The system calls *ttyget* and *ttyset* can be used to examine device status and change operation settings.

Other Devices

The `/dev` directory contains other entries for devices supported by the operating system:

diskc	Raw system disk
disk1c..diskнc	Raw optional winchester disk
floppyc	Raw floppy disk drive
tapec	Raw streaming tape drive
null	Null device
pmem	Physical memory
smem	System memory
swap	Swap space on winchester disk

These devices are all character-oriented. The raw versions of the peripheral devices provide access to them as simple character streams without file systems. The null device may be used as a data sink. The memory devices can be used to inspect and modify the system's memory.

These devices, with the exception of */dev/null*, are reserved for use by system programs.

CAUTION

Be very careful when you use these devices because errors in programming them may crash the operating system and destroy the disk file structure.

DISPLAY, MOUSE, AND KEYBOARD SUPPORT

The display of the 4400 series products is dependent upon the hardware design. Refer to the appendices for a description of the display of each product. The operating system controls access to the display and is consistant throughout the product family. The operating system also supports the creation and movement of a display curser. Information about mouse and keyboard event processing can be found in the *4400 Series Assembly Language Reference.*

The 4400 series display uses Smalltalk-80 conventions. The upper-left corner of the display has coordinates (0,0), while the lower-right corner has coordinates (1023,1023). X coordinates increase toward the bottom of the screen; Y coordinates increase to the right.

Full program access to interactive event processing is supported through system calls to an event manager. Events include mouse movements, and up/down transitions of the mouse, keyboard, and joydisk contacts. The design of the event mechanism is patterned closely after a similar mechanism described on pages 648-650 of the book *Smalltalk-80: The Language and Its Implementation.*

Cursor and Mouse Tracking

The cursor is a 16 X 16 bit-map that is displayed by logically ORing it into the display bit-map. The contents of the area under the cursor are saved and they are restored when the cursor is moved. The operating system allows the cursor's position to track the motion of the mouse. When this feature is enabled, the operating system will automatically move the cursor whenever the mouse is moved.

The mouse position is not allowed to exceed certain bounds when the cursor is linked to the mouse. The default bounds are the virtual display coordinates. The user may change these bounds and allow the cursor to be moved off the virtual display.

FLOATING POINT SUPPORT

The operating system provides access to the floating point hardware. Floating point values are in IEEE format. Both 32-bit single precision and 64-bit double precision formats are supported. For more information about floating point support, refer to *4400 Series Assembler Language Programmers Reference.*

Appendix A
4404 HARDWARE DEPENDENCIES

This appendix is specifically for the 4404 Artificial Intellegence Machine, containing information about the display and memory organization. Section 6 describes how to access the hardware with the software in the 4400 series family.

DISPLAY SUPPORT

The 4404 display is a 1024 X 1024 virtual display viewed through a 640 X 480 physical display viewport. The relation between the virtual display and the display viewport is shown in Figure A-1. The operating system includes support that allows positioning and smooth panning of the viewport over the virtual display. The operating system also supports the creation and movement of a display cursor.

The 4404 display uses Smalltalk-80 conventions. The upper-left corner of the display has coordinates (0,0), while the lower-right corner has coordinates (1023,1023). X coordinates increase toward the bottom of the screen; Y coordinates increase to the right.

Full program access to interactive event processing is supported through system calls to an event manager. Events include mouse movements, and up/down transitions of the mouse, keyboard, and joydisk contacts. The design of the event mechanism is patterned closely after a similar mechanism described on pages 648-650 of the book *Smalltalk-80: The Language and Its Implementation.*

Display Panning

The operating system allows the 640 X 480 hardware display viewport to be positioned anywhere on the virtual display as long as the upper left corner has an X-coordinate less than 383 and a Y-coordinate less than 543.

The operating system supports the panning of the viewport over the virtual display under control of the mouse and joydisk. When joydisk panning is enabled, pushing the top of the joydisk causes the Y-coordinate to decrease by 5 pixels during each vertical sync interrupt, while pushing the bottom causes it to increase a like amount. Pushing the left side of the joydisk causes the X-coordinate to decrease 5 pixels per interrupt; while pushing the right side of the joydisk causes it to increase. Joydisk panning ceases in a particular direction when the coordinate for that direction reaches zero or its maximum value.

The cursor remains at a fixed position on the virtual display while the viewport is panned by the joydisk. When cursor panning is enabled, moving the cursor will also cause the viewport to pan so that the cursor is always located within the physical viewport. This allows the mouse to pan the viewport position because the cursor position is usually linked to mouse movement.

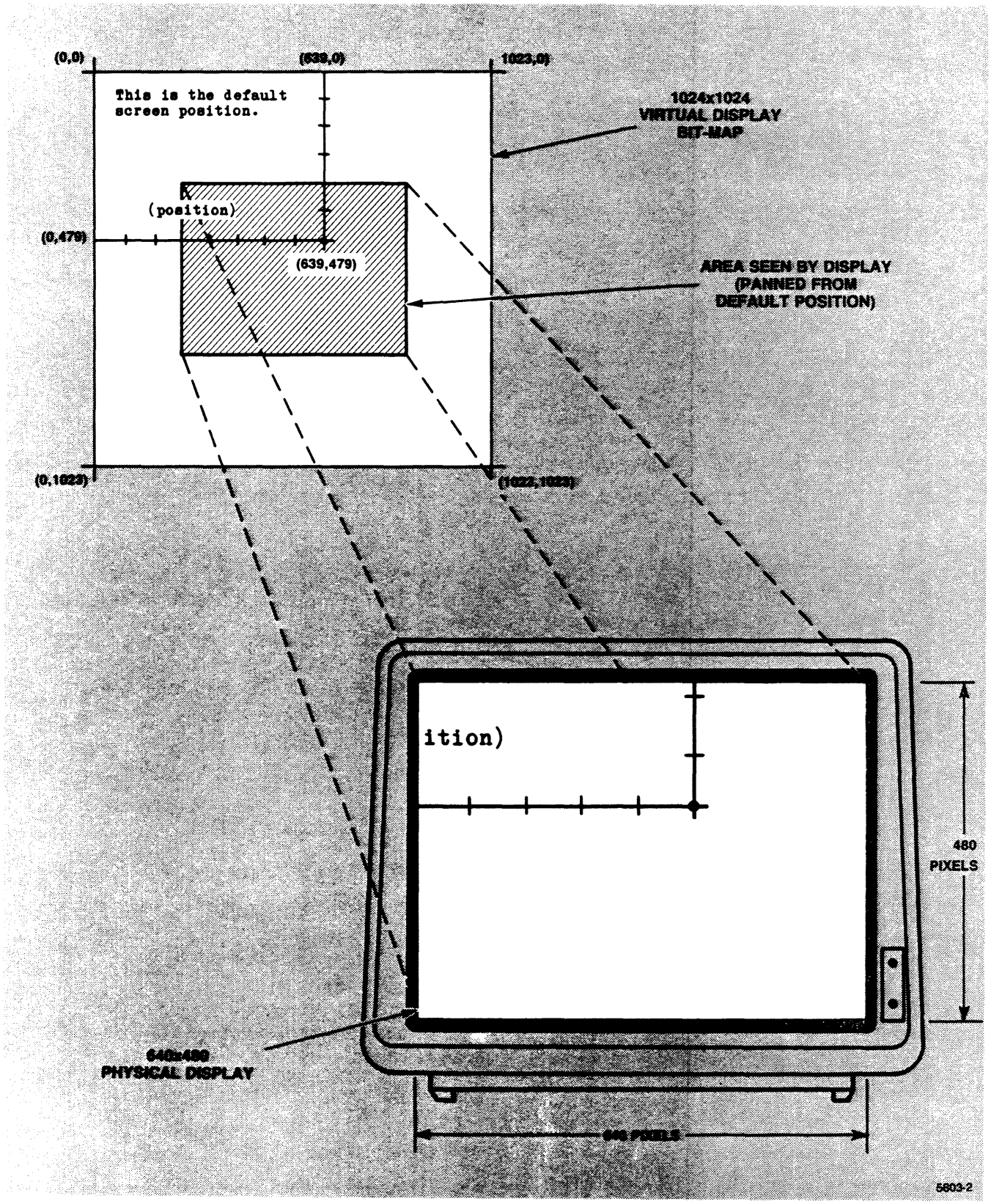

Figure A-1. 640 X 480 Window Into 1024 X 1024 Bit-Map.

MEMORY USE

Overall Address Space

The 68010 processor on the 4404 is capable of addressing 16 Mb of memory. Of this, the 4404 recognizes the lower 8 Mb. (All addresses given in this discussion will be hexadecimal unless stated otherwise.) The 4404 operating system uses a virtual memory scheme whereby 8 Mb of virtual memory is mapped into the 4404's physical memory in 4 Kb increments. To a programmer working through the operating system, it appears that the entire 8 Mb address space (ranging from 000000 through 1FFFFF) is available.

Physical Memory

The standard 4404 contains 1 Mb of physical RAM in addresses 000000 through 0FFFFF. Option 1 adds an additional 1Mb of physical memory in addresses 100000 through 2FFFFF.

Addresses 200000 through 5FFFFF are reserved for future expansion.

Display Memory

The 4404 display memory begins at address 600000 and occupies through address 6FFFFF. The virtual display begins in the upper left corner at address 600000 and proceeds in 1024 (decimal) lines of 64 (decimal) 16-bit (decimal) words. Each word has the most significant bit first, thus each word controls 16 pixels on the display.

I/O and ROM Memory Space

The memory segment from 700000 through 7FFFFF is dedicated to ROM, I/O, and various utilities. It consists of eight 128 Kb pages arranged as:

700000 -- 71FFFF	Spare 0
720000 -- 73FFFF	Spare 1
740000 -- 75FFFF	Boot ROM
760000 -- 77FFFF	Debug ROM space (for factory use)
780000 -- 79FFFF	Processor Board I/O (treated later)
7A0000 -- 7BFFFF	Peripheral Board I/O (treated later)
7C0000 -- 7FFFFF	EPROM application space

Processor Board I/O

780000 -- 781FFF	Map Control Registers
782000 -- 783FFF	Video Address Registers
784000 -- 785FFF	Video Control Registers
786000 -- 787FFF	Spare
788000 -- 789FFF	Sound
78A000 -- 78BFFF	Floating Point Hardware
78C000 -- 78DFFF	Debug ACIA
78E000 -- 78FFFF	Spare

Peripheral Board I/O

7A0000 -- 7AFFFF	Reserved for future expansion
7B1000 -- 7B1FFF	Diagnostic registers
7B2000 -- 7B3FFF	Printer
7B4000 -- 7B5FFF	Serial I/O
7B6000 -- 7B7FFF	Mouse
7B8000 -- 7B9FFF	Timer
7BA000 -- 7BBFFF	Calendar
7BC000 -- 7BDFFF	SCSI bus address registers
7BE000 -- 7BFFFF	SCSI

Appendix B
4405 HARDWARE DEPENDENCIES

This appendix is specifically for the 4405 Artificial Intellegence Machine, containing information about the display and memory organization. Section 6 describes how to access the hardware with the software in the 4400 series family.

DISPLAY SUPPORT

The 4405 display is a 1024 X 1024 virtual display viewed through a 640 X 480 physical display viewport. The relation between the virtual display and the display viewport is shown in Figure B-1. The operating system includes support that allows positioning and smooth panning of the viewport over the virtual display. The operating system also supports the creation and movement of a display cursor.

The 4405 display uses Smalltalk-80 conventions. The upper-left corner of the display has coordinates (0,0), while the lower-right corner has coordinates (1023,1023). X coordinates increase toward the bottom of the screen; Y coordinates increase to the right.

Full program access to interactive event processing is supported through system calls to an event manager. Events include mouse movements, and up/down transitions of the mouse, keyboard, and joydisk contacts. The design of the event mechanism is patterned closely after a similar mechanism described on pages 648-650 of the book *Smalltalk-80: The Language and Its Implementation.*

Display Panning

The operating system allows the 640 X 480 hardware display viewport to be positioned anywhere on the virtual display as long as the upper left corner has an X-coordinate less than 383 and a Y-coordinate less than 543.

The operating system supports the panning of the viewport over the virtual display under control of the mouse and joydisk. When joydisk panning is enabled, pushing the top of the joydisk causes the Y-coordinate to decrease by 5 pixels during each vertical sync interrupt, while pushing the bottom causes it to increase a like amount. Pushing the left side of the joydisk causes the X-coordinate to decrease 5 pixels per interrupt; while pushing the right side of the joydisk causes it to increase. Joydisk panning ceases in a particular direction when the coordinate for that direction reaches zero or its maximum value.

The cursor remains at a fixed position on the virtual display while the viewport is panned by the joydisk. When cursor panning is enabled, moving the cursor will also cause the viewport to pan so that the cursor is always located within the physical viewport. This allows the mouse to pan the viewport position because the cursor position is usually linked to mouse movement.

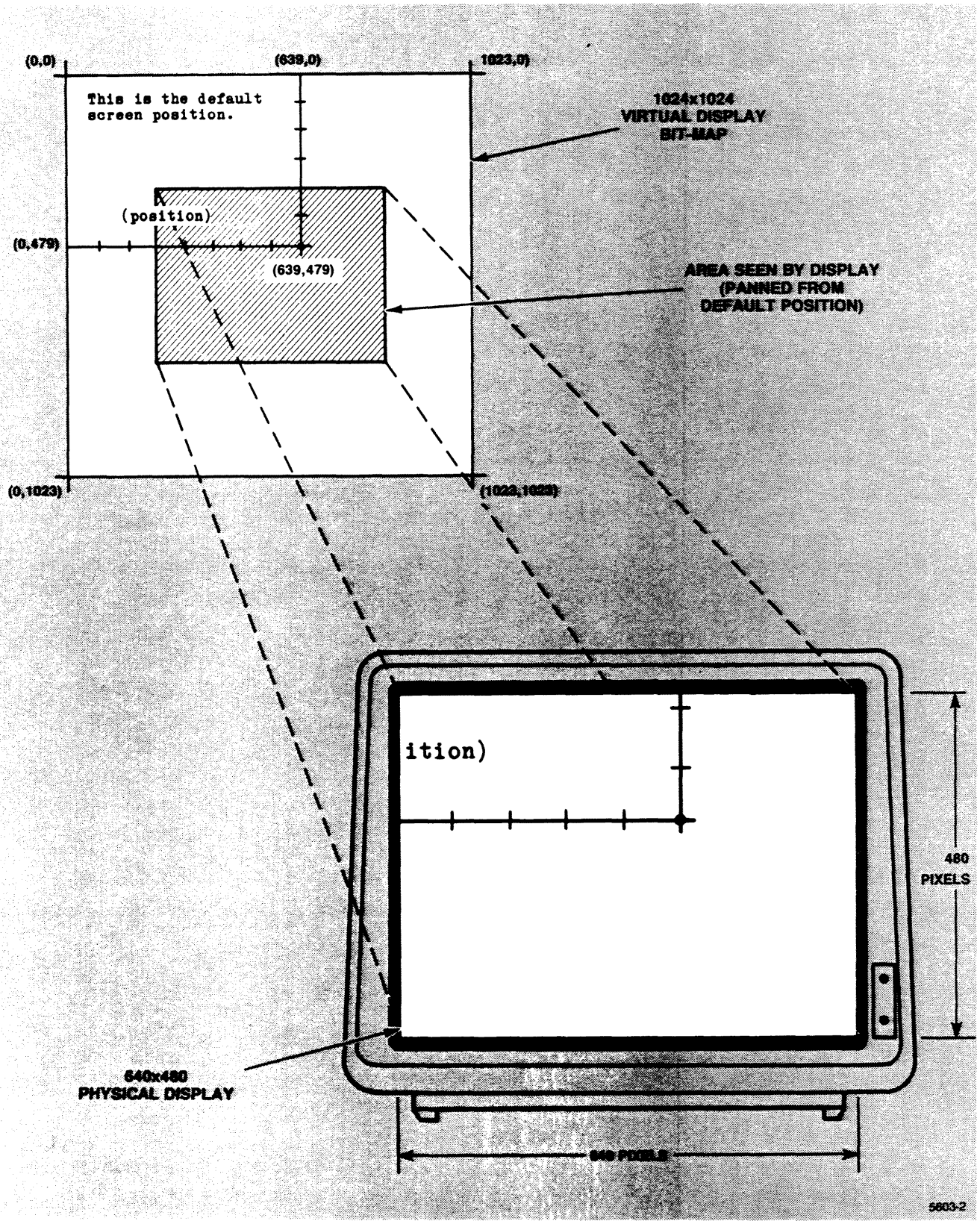

Figure B-1. 640 X 480 Window Into 1024 X 1024 Bit-Map.

MEMORY USE

Overall Address Space

The 68020 processor on the 4405 is capable of addressing a 32 Mb logical address range of memory. Of this, the 4405 recognizes the lower 16 Mb. (All addresses given in this discussion will be hexadecimal unless stated otherwise.) The 4405 operating system uses a virtual memory scheme whereby 32 Mb of virtual memory is mapped into the 4405's physical memory in 4 Kb increments. To a programmer working through the operating system, it appears that the entire 32 Mb address space (ranging from 000000 through FFFFFF) is available.

Physical Memory

The standard 4405 contains 2 Mb of physical RAM in addresses 000000 through 1FFFFF. Option 2/4 adds an additional 2/4 Mb of physical memory in addresses 200000 through 6FFFFF.

Addresses 700000 through DFFFFF are reserved for future expansion.

Display Memory

The 4405 display memory begins at address E00000 and occupies through address EFFFFF. The virtual display begins in the upper left corner at address E00000 and proceeds in 1024 (decimal) lines of 64 (decimal) 16-bit (decimal) words. Each word has the most significant bit first, thus each word controls 16 pixels on the display.

I/O and ROM Memory Space

The memory segment from F00000 through FFFFFF is dedicated to ROM, I/O, and various utilities. It consists of eight 128 Kb pages arranged as:

F00000 -- F1FFFF	Spare 0
F20000 -- F3FFFF	Spare 1
F40000 -- F5FFFF	Boot ROM
F60000 -- F7FFFF	Debug ROM space (for factory use)
F80000 -- F9FFFF	Processor Board I/O (treated later)
FA0000 -- FBFFFF	Peripheral Board I/O (treated later)
FC0000 -- FFFFFF	EPROM application space

Processor Board I/O

F80000 -- F81FFF	Map Control Registers
F82000 -- F83FFF	Video Address Registers
F84000 -- F85FFF	Video Control Registers
F86000 -- F87FFF	Spare
F88000 -- F89FFF	Sound
F8A000 -- F8BFFF	Floating Point Hardware
F8C000 -- F8DFFF	Debug ACIA
F8E000 -- F8FFFF	Spare

Peripheral Board I/O

FA0000 -- FAFFFF	Reserved for future expansion
FB1000 -- FB1FFF	Diagnostic registers
FB2000 -- FB3FFF	Printer
FB4000 -- FB5FFF	Serial I/O
FB6000 -- FB7FFF	Mouse
FB8000 -- FB9FFF	Timer
FBA000 -- FBBFFF	Calendar
FBC000 -- FBDFFF	SCSI bus address registers
FBE000 -- FBFFFF	SCSI

Appendix C
4406 HARDWARE DEPENDENCIES

This appendix is specifically for the 4406 Artificial Intellegence Machine, containing information about the display and memory organization. Section 6 describes how to access the hardware with the software in the 4400 series family.

DISPLAY SUPPORT

The 4406 display is a 1280 X 1024 raster display. The operating system supports the creation and movement of a display cursor.

The 4406 display uses Smalltalk-80 conventions. The upper-left corner of the display has coordinates (0,0), while the lower-right corner has coordinates (1023,1023). X coordinates increase toward the bottom of the screen; Y coordinates increase to the right.

Full program access to interactive event processing is supported through system calls to an event manager. Events include mouse movements, and up/down transitions of the mouse, keyboard, and joydisk contacts. The design of the event mechanism is patterned closely after a similar mechanism described on pages 648-650 of the book *Smalltalk-80: The Language and Its Implementation.*

MEMORY USE

Overall Address Space

The 68020 processor on the 4406 is capable of addressing a 32 Mb logical address range of memory. Of this, the 4406 recognizes the lower 16 Mb. (All addresses given in this discussion will be hexadecimal unless stated otherwise.) The 4406 operating system uses a virtual memory scheme whereby 32 Mb of virtual memory is mapped into the 4406's physical memory in 4 Kb increments. To a programmer working through the operating system, it appears that the entire 32 Mb address space (ranging from 000000 through FFFFFF) is available.

Physical Memory

The standard 4406 contains 2 Mb of physical RAM in addresses 000000 through 1FFFFF. Option 2/4 adds an additional 2/4 Mb of physical memory in addresses 200000 through 6FFFFF.

Addresses 700000 through DFFFFF are reserved for future expansion.

Display Memory

The 4406 display memory begins at address E00000 and occupies through address EFFFFF. The virtual display begins in the upper left corner at address E00000 and proceeds in 1024 (decimal) lines of 64 (decimal) 16-bit (decimal) words. Each word has the most significant bit first, thus each word controls 16 pixels on the display.

I/O and ROM Memory Space

The memory segment from F00000 through FFFFFF is dedicated to ROM, I/O, and various utilities. It consists of eight 128 Kb pages arranged as:

F00000 -- F1FFFF	Spare 0
F20000 -- F3FFFF	Spare 1
F40000 -- F5FFFF	Boot ROM
F60000 -- F7FFFF	Debug ROM space (for factory use)
F80000 -- F9FFFF	Processor Board I/O (treated later)
FA0000 -- FBFFFF	Peripheral Board I/O (treated later)
FC0000 -- FFFFFF	EPROM application space

Processor Board I/O

F80000 -- F81FFF	Map Control Registers
F82000 -- F83FFF	Video Address Registers
F84000 -- F85FFF	Video Control Registers
F86000 -- F87FFF	Spare
F88000 -- F89FFF	Sound
F8A000 -- F8BFFF	Floating Point Hardware
F8C000 -- F8DFFF	Debug ACIA
F8E000 -- F8FFFF	Spare

Peripheral Board I/O

FA0000 -- FAFFFF	Reserved for future expansion
FB1000 -- FB1FFF	Diagnostic registers
FB2000 -- FB3FFF	Printer
FB4000 -- FB5FFF	Serial I/O
FB6000 -- FB7FFF	Mouse
FB8000 -- FB9FFF	Timer
FBA000 -- FBBFFF	Calendar
FBC000 -- FBDFFF	SCSI bus address registers
FBE000 -- FBFFFF	SCSI

Appendix U
4400 Series O/S Reference Update

Please add the following subjects to your copy of the *4400 Series Operating System Reference Manual.* The information was not available when the manual was printed.

cc

The 'C' compiler, ***cc***, takes the additional option ***+Q***.

Additional Options

+Q —Suppress quad word alignment on 68020 code generation*

*68020 only

Explanation Of Options

The 'C' compiler, by default, aligns data structures on ***quad word*** (words consisting of four eight-bit bytes) boundaries. This, while allowing the 68020 to load and execute faster, causes "holes" in the data structures. The ***+Q*** option lets you suppress this alignment to allow close packing of data structures or compatilbility with data structures generated with non quad-aligned compilers, such as 68000 or 68010 compilers.

conset

You can choose several sizes and styles of monospaced fonts for your normal screen display on the 4405 and 4406.

Additional Options

selectFont=<name> -- small, smallBold, medium, mediumBold,
large, largeBold, extraLarge, extraLargeBold*

*68020 only.

Explanation Of Options

Conset allows you to select the size and style of fonts used by the terminal emulator. It allows you to select the suitable monospaced fonts from the selections in the directory ***/fonts***. The additional fonts in this directory are available via Smalltalk-80 or the graphics library.

headset

The ***headset*** command allows you to enable or disable the floating point processor signals, enable or disable demand-load operation, and block-align??? (questions here)...

Additional Options

+f/-f -- make file demand-load and block-aligned
+I/-I -- set or clear "floating-point signal" bit
+Z/-Z -- block-align text and data segments on 512 byte boundaries

Explanation Of Options

The ***+f/-f*** option makes the file block-aligned (see the ***+Z*** option) and causes it to be ***demand-load*** — that is, only pages that cause a page fault are loaded into physical memory. Unreferenced pages are not loaded.

The ***+I/-I*** option allows you to catch or ignore signals generated by the floating point processor. By default, floating point signals are ignored. If you want to catch these signals, enable them with the ***+I*** option.

The ***+Z/-Z*** option lets you force text and data segments to begin on 512 byte boundaries.

load

Additional Options

+q -- suppress quad word alignment of each segment*
+I -- enable floating point signal processing
+z -- align text and data segments on 512 byte boundaries

*68020 only

Explanation Of Options

Normally, the loader aligns the beginning of each segment on a ***quad word*** (a word consisting of four eight-bit bytes) boundary. The ***+q*** option allows you to suppress this alignment.

The loader normally does not enable floating point processor signals. If you are using, or need to catch, these signals, enable them with the ***+I*** option. You can enable or disable signals on a compiled and loaded program via the *headset* utility.

The ***+Z*** option lets you force text and data segments to begin on 512 byte boundaries.

Index