305-506
Issue 1

AT&T 3B2 Computer
UNIX™ System V Release 2.0
Utilities – Volume 3

NOTICE

The information in this document is subject to change without notice. AT&T assumes no responsibility for any errors that may appear in this document.

Copyright© 1985 AT&T
All Rights Reserved
Printed in U.S.A

Replace this

page with the

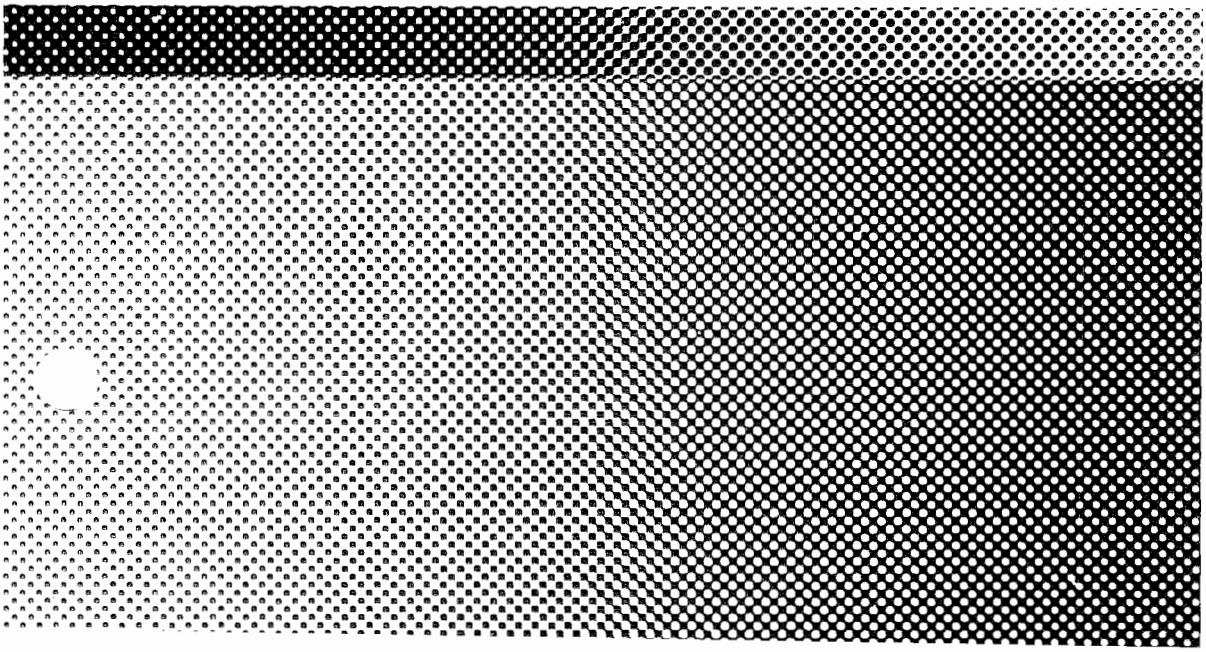
LINE PRINTER SPOOLING

tab separator.

AT&T

AT&T 3B2 Computer

UNIX™ System V Release 2.0 Line Printer Spooling Utilities Guide



CONTENTS

Chapter 1. INTRODUCTION

Chapter 2. USER COMMANDS

Chapter 3. ADMINISTRATIVE COMMANDS

Chapter 4. ADMINISTRATION

Appendix: ERROR MESSAGES

Chapter 1

INTRODUCTION

PAGE

GENERAL 1-1
GUIDE ORGANIZATION 1-2
SUPPORTING DOCUMENTATION 1-3
DEFINITION OF TERMS 1-4

Chapter 1

INTRODUCTION

GENERAL

This document describes the Line Printer (LP) Spooling Utilities. The LP Spooling Utilities is a set of commands that allows you to "spool" a file that you want to print. Spooling is the name given to the technique of temporarily storing data until another aspect of processing is ready for the data. For the Line Printer Spooling, a file (or several files) to be printed is stored in a queue until a printer becomes available. When the printer becomes available, the file in the queue is printed.

LP Spooling frees up your terminal so you do not have to wait until the file is printed before you can start using your terminal again. LP Spooling also allows one printer, or several printers, to be shared among many users. The flow of printing through the system is regulated by the LP Spooling Utilities.

The LP Spooling Utilities allows:

- Customizing the system so that it will spool to a pool of line printers. These printers may be all the same type or various types of printers.

- Grouping printers into logical classes to maximize the throughput of the printers. For example, grouping all letter-quality printers into one class and all high-speed dot matrix printers into another class.
- Queuing print requests, thus allowing a print request (job) to get printed on the next available printer in that class.
- Canceling print requests so one job that is no longer needed will not be printed.
- Starting and stopping LP from processing requests.
- Changing the configuration of printers.
- Finding the status of LP scheduler.
- Restarting any printing that was not completed if the system was powered down.

GUIDE ORGANIZATION

This guide is structured so you can easily find information without having to read the entire text. The remainder of this guide is organized as follows:

- Chapter 2, "USER COMMANDS," describes the format and use of the LP Spooling Utilities commands that are used by a general **UNIX*** System user.
- Chapter 3, "ADMINISTRATIVE COMMANDS," describes the format and use of the LP Spooling Utilities commands that are used to administer the LP system.

---

* Trademark of AT&T

- Chapter 4, "ADMINISTRATION," provides additional information on administering the LP system.

- Appendix, "ERROR MESSAGES," is a listing of the error messages associated with the LP commands. An explanation of the error message is provided, as well as the corrective action to take.

SUPPORTING DOCUMENTATION

Before reading this document you should be familiar with the *AT&T 3B2 Computer Owner/Operator Manual* that describes how to operate the computer and the *UNIX System V User Guide*. Other documents that you may need to use with the *LP Spooling Utilities Guide* are:

- *AT&T 3B2 Computer User Environment Utilities Guide* – explains how to set up a user environment. This utilities is required to use the LP Spooling Utilities.

- *AT&T 3B2 Computer Dot Matrix Printer Manual* – explains how to turn the DQP-10 printer on, load paper and ribbons, and how to set it up.

- *AT&T 3B2 Computer Letter Quality Printer Manual* – explains how to turn the LQP-40 printer on, load paper and ribbons, and how to set it up.

- *AT&T 3B2 Computer Expanded Input/Output Capability Manual* – explains how to add additional ports to the computer so you can connect additional printers.

DEFINITION OF TERMS

There are terms in this document that have a singular meaning within the context of the document. Knowing the meaning of these terms is critical to your understanding of the LP system. These terms are defined below:

device — Depending on its usage, the word *device* can refer to either an apparatus for obtaining a printout or a special UNIX System file found in the /dev directory. These special UNIX System files are called "device" files. The UNIX System uses these device files to access peripherals such as printers, terminals, tape drives, etc. Some typical device file names for accessing printers are tty11, tty12, tty13, and tty14.

printer — This is a logical name that represents a physical device. In this guide the printer names "dqp10" and "lqp40" are used extensively. The name "dqp10" is the logical name assigned to the dot matrix printer; "lqp40" is the logical name assigned to the letter quality printer.

class — *Class* is the name given to an ordered list of printers. Although a *class* is normally thought of as a group of printers, a *class* could contain only one printer. A printer does not have to be assigned to a *class*; but on the other hand, you can assign a printer to more than one *class*.

destination — This is the location where an LP Spooling output request is sent to be printed or to await printing. A *destination* may be a specific printer or a class of printers. An output request directed to a specific printer will only be printed by that printer; an output directed to a class will be printed by the first available printer in that class.

Chapter 2

USER COMMANDS

PAGE

GENERAL 2-1

COMMAND SUMMARY 2-2

HOW COMMANDS ARE DESCRIBED 2-3

COMMAND DESCRIPTIONS 2-5

- cancel — Stop a Request from Printing 2-5
- disable — Stop Printing Requests on Printers 2-7
- enable — Enable Printing Requests on Printers 2-9
- lp — Make an Output Request 2-11
- lpstat — Get LP Status 2-15

Chapter 2

USER COMMANDS

GENERAL

The Line Printer (LP) Spooling Utilities provides eleven UNIX System commands. These commands are divided into two categories. The commands in one category are for general use of the LP system; the commands in the other category are for administering the LP system. This chapter describes the general user commands. The commands for administering the LP system are covered in Chapter 3.

COMMAND SUMMARY

Figure 2-1 provides a summary of the general user commands.

COMMAND	DESCRIPTION
cancel	Cancels output requests.
disable	Prevents a printer from printing jobs that are in the queue.
enable	Allows a printer to print jobs that are in the queue.
lp	Routes jobs to a destination and places them in a queue. The destination may be a printer or a class of printers.
lpstat	Provides the status of anything that has to do with the LP Spooling system.

Figure 2-1. LP Spooling Utilities—User Command Summary

HOW COMMANDS ARE DESCRIBED

A common format is used to describe each of the commands. This format is as follows:

- **General:** The purpose of the command is defined. Any uncommon or special information about the command is also provided.
- **Command Format:** The basic command line format (syntax) is defined and the various arguments and options are discussed.
- **Sample Command Use:** Example command line entries and system responses are provided to show you how to use the command.

In the command format discussions, the following symbology and conventions are used to define the command syntax:

- The basic command is shown in bold type. For example: **command** is in bold type.
- Arguments that you must supply to the command are shown in a special type. For example: **command** *argument*
- Command options and arguments that do not have to be supplied are enclosed in brackets ([ ]). For example:
  command [*optional arguments*]
- The pipe symbol (|) is used to separate arguments when one of several forms of an argument can be used for a given argument field. The pipe symbol can be thought of as an exclusive OR function in this context. For example:
  command [*argument1* | *argument2*]

Refer to the *AT&T 3B2 Computer User Reference Manual* and the *AT&T 3B2 Computer System Administration Reference Manual* for the UNIX System V manual pages supporting the commands described in this guide. In the sample command discussions, and in most other instances, screen displays are used to show user inputs and system responses.

The following conventions are used within the screen displays:

```
This style of type is used to show system generated
responses displayed on your screen.
```

This style of bold type is used to show inputs entered from your keyboard that are displayed on your screen.

These bracket symbols, < > identify inputs from the keyboard that are not displayed on your screen, such as: <*CR*> carriage return, <*CTRL d*> control d, <*ESC g*> escape g, passwords, and tabs.

This style of italic type is used for notes that provide you with additional information.

Other conventions used in the sample command discussions that need to be mentioned are prompt symbols and printer names. Two different prompt symbols are used: a "**$**" and a "#". The "**$**" is used as the prompt for commands that can be entered by a general user; the "#" is the prompt for commands that can only be entered by "root" or a super-user.

The printer names that are used in many of the sample commands are the default printer names that the installation script assigns. Examples of default names are dqp10_1, dqp10_2, and lqp40_1. If default printer names are not used, then lp1, lp2, lp3, etc., are used as printer names.

COMMAND DESCRIPTIONS

cancel — Stop a Request from Printing

General

The **cancel** command is used to remove a job from the queue. This command can be invoked before a job starts printing or after a job has started printing. Only one job can be canceled at a time.

Any user is able to cancel another user's job. The reason that LP allows anyone to use the **cancel** command is to cover problems that may occur on an unattended printer. Some typical problems might be a printer spewing out unintelligent information or a printer with a paper jam that keeps printing the same line over and over. If **cancel** could only be invoked by the original requester, or someone who can log in as "lp" or "root," it might take a long time to find a person who can stop the job.

If you cancel another user's request, mail is sent to that user. Once a job is canceled, it has to be requested again using the **lp** command.

Command Format

The general format of the **cancel** command is as follows:

cancel [*printer–name* | *request identification–number*]

Two kinds of arguments may be given to the command — request identification numbers and printer names. A **cancel** *request identification number* cancels one job request. A **cancel** *printer–name* cancels only the job that is currently printing. No other requests in the queue for the named printer will be canceled. Both types of arguments may be intermixed. With no arguments, the **cancel** command cancels the job currently printing.

Sample Command Use

The following is an example of canceling a request that is now printing on printer dqp10_1:

```
$ cancel dqp10_1-51<CR>
request " dqp10_1-51" canceled
$
```

disable — Stop Printing Requests on Printers

General

The **disable** command prevents the printer from printing any jobs that are in the queue. You might want to disable a printer for things like malfunctioning hardware, paper jams, running out of paper, or end-of-day shutdowns. If a printer is busy at the time it is disabled, then the request that was printing will be reprinted in its entirety when the printer is enabled.

Job requests can be routed to a printer that is disabled. The jobs go into a queue, but they will not be printed until the printer is enabled.

Command Format

The general format of the **disable** command is as follows:

disable [**–c**] [**–r**[*reason*]] *printers*

The **-c** option causes the currently printing request on a printer to be canceled in addition to disabling the printer. This is useful if you have some strange output causing a printer to behave abnormally.

The **-r** option enables you to let other users know why the printer was disabled. *Reason* is a brief explanation of the purpose for disabling the printer. If the reason consists of several words separated by spaces, then enclose the reason statement in double quotes ("). This reason is reported to other users who may try to use the disabled printer.

Sample Command Use

Example 1

This is an example of how you would disable printer dqp10_1 if the paper is jammed:

```
$ disable -r"paper jam" dqp10_1<CR>
printer "dqp10_1" now disabled
$
```

Example 2

This example shows the response you get if you use the **lpstat** command to determine the status of printer dqp10_1:

```
$ lpstat -pdqp10_1<CR>
printer pdq10_1 disabled since July 18 10:15 -
        paper jam
$
```

enable — Enable Printing Requests on Printers

General

The **enable** command allows the printer to print jobs that are in the queue. A job in the process of printing, that is stopped by the **disable** command, will start printing again from the beginning after the **enable** command is invoked.

Command Format

The general format of the **enable** command is as follows:

enable *printers*

Sample Command Use

```
$ enable dqp10_2<CR>
printer " dqp10_2" now enabled
$
```

lp — Make an Output Request

General

The **lp** command routes a job request to a destination where it is placed in a queue to await printing. The destination may be a printer or a class of printers. If no destination is specified, the request is routed to the default destination.

Every time an **lp** request is made, a "request ID" is assigned to the job, and a record of the request is returned to you. The request ID is of the form:

dest-seqno

where *dest* is the destination to which the request was routed and *seqno* is a sequence number that is uncommon to the LP system. The request ID provides a means for keeping track of the status of individual job requests. It also provides a means for canceling job requests.

Command Format

The command format is as follows:

lp [*options*] *file(s)*

The following options are available with the **lp** command:

–c	This option will immediately create a copy of the file(s) to be printed. Using this option ensures that no changes will be made to the file even if some time elapses before the file is actually printed.
–d*dest*	This is the printer where you want the file(s) to be printed.

–m	When you select this option, mail is sent to you after the file(s) has been printed.
–n*number*	If no number is specified, one copy is printed. By specifying a number, multiple copies can be printed.
–o*option*	The *option* referred to is the optional modes of printing some printers have, for example, compressed print or expanded print.
–s	This option suppresses messages such as "request ID is . . ."
–t*title*	This will "banner" a title on the printout so you can distinguish your job from other jobs.
–w	The -w option will write a message on your terminal after the file(s) has been printed. If you log off before your file(s) is printed, then mail will be sent to you.

Sample Command Uses

There are several different ways to request a printout with the **lp** command. The first example shows four different ways to obtain a printout of the */etc/passwd* file. Since no destination is specified in the command line, the request will go to the system default destination.

Example 1

```
$ lp /etc/passwd<CR>
request id is dqp10_1-53 (1 file)

$ lp < /etc/passwd<CR>
request id is dqp10_1-54 (standard input)

$ cat /etc/passwd | lp<CR>
request id is dqp10_1-55 (standard input)

$ lp -c /etc/passwd<CR>
request id is dqp10_1-56 (1 file)
$
```

The LP system does not handle all the requests shown in the example in the same manner. All the command lines, except the first one, "freeze" a copy of the */etc/passwd* file as soon as the **lp** command is issued. This copy is held until a printer is ready. If the file is modified between the time the request is made and the time it is actually printed, the changes are not included in the output.

The first command line does not freeze a copy of the file. Any changes made to the */etc/passwd* file between the time the **lp** request is made and the time the file is actually printed are included in the printout.

When you use LP spooling, it is a good idea to know whether the method you use will freeze a copy of the file or if the printout will reflect changes that are made after the **lp** request was made.

Example 2

This example shows the command line that you would use to print file "xyz" on printer dqp10_1 and have a message sent back to you after it prints:

```
$ lp -ddqp10_1  -w xyz<CR>
request id is dqp10_1-65 (1 file)
$
```

Example 3

Here is an example showing how to print two copies of file *xyz* on printer dqp10_1 and title the output *myfile*:

```
$ pr xyz | lp -ddqp10_1 -n2 -t"myfile" <CR>
request id is dqp10_1-1 (standard input)
$
```

Note: The title of the file is bannered on the output.

lpstat — Get LP Status

General

The **lpstat** command gives you a report on such things as:

- Jobs in the queue
- A job that is printing
- Printers that are busy or idle
- LP scheduler status
- The system default destination.

In short, it provides a status report of anything that has to do with the LP Spooling system.

Command Format

The general format of the **lpstat** command is as follows:

lpstat [*options*]

Where options may be:

–a[*list*]	Report whether printers are accepting requests. *List* is a list of printer names and class names.
–c[*list*]	Report all class names and their members. *List* is a list of class members.
–d	Report the system default destination printer.

–o[*list*]	Report the status of requests. *List* is a list of printer names, class names, or request identification numbers.
–p[*list*]	Report the status of printers. *List* is a list of printer names.
–r	This option is used to determine if the LP scheduler is on or off.
–s	Print a status summary including the system default destination, a list of class names and their members, and a list of printers and their associated devices.
–t	Report all status information. Prints all the information that is given with a **-s** plus prints the acceptance and idle/busy status of all printers.
–u[*list*]	Report the status of requests for users. *List* is a list of login names.
–v[*list*]	Identifies the associated device for each LP printer. *List* is a list of printer names.

Sample Command Uses

Example 1

Invoke the **lpstat** command without any options. The status information that you receive includes:

- The request identification number
- The user's logname
- The total amount of characters to be printed

- The date and time the request was made.

```
$ lpstat<CR>
dqp10_1-25      pr2cms      1942   July 19 13:09
dqp10_1-26      pr2cms      3893   July 19 13:15
dqp10_1-27      pr2cms       942   July 19 14:09
$
```

Example 2

Request the status of printers, dqp10_1 and lqp40_1. Note that only a comma separates the name of the two printers. It is important that no spaces or tabs are between them.

```
$ lpstat -pdqp10_1,lqp40_1<CR>
printer dqp10_1 is idle.  enabled since July 19 08:40
printer lqp40_1 is idle.  enabled since July 19 09:16
```

Example 3

Execute the **lpstat** command with the **-t** option to print all the status information.

```
$ lpstat -t<CR>
scheduler is running
system default destination: dqp10_1
device for dqp10_1: /dev/tty11
device for lqp40_1: /dev/tty12
dqp10_1 accepting requests since July 19 13:44
lqp40_1 accepting requests since July 19 11:30
printer dqp10_1 disabled since July 18 15:07
        bad print wheel
printer lqp40_1 is idle.  enabled since July 19 11:30
dqp10_1-19     pr2cms                1616   July 19 17:13
$
```

Chapter 3

ADMINISTRATIVE COMMANDS

PAGE

GENERAL 3-1

COMMAND SUMMARY 3-2

COMMAND DESCRIPTIONS 3-3

accept — Allows Print Requests 3-3

reject — Prevent LP Requests 3-5

lpadmin — Configure Printers 3-7

lpmove — Move a Request to Another Printer 3-11

lpsched — Start the LP Scheduler 3-13

lpshut — Stop the LP Scheduler 3-15

Chapter 3

ADMINISTRATIVE COMMANDS

GENERAL

This chapter describes the commands that are used to administer the LP system. The administrative commands are described using the same format that was used to describe the general user commands in Chapter 2. To execute the administrative commands, you must be logged in as "root" or "lp." Administrative commands should only be used by experienced users.

COMMAND SUMMARY

Figure 3-1 provides a summary of the administrative commands.

COMMAND	DESCRIPTION
accept	Permits job requests to be queued for a particular destination.
reject	Prevents jobs from being queued at a particular destination.
lpadmin	Used to set up or change the LP configuration.
lpmove	Moves output requests from one destination to another.
lpsched	Starts the LP scheduler.
lpshut	Stops the LP scheduler.

Figure 3-1. LP Spooling Utilities — Administrative Command Summary

COMMAND DESCRIPTIONS

accept — Allows Print Requests

General

The **accept** command allows job requests to be placed in a queue at the named destination(s); destination being the name of a printer or class of printers.

Command Format

The general format of the **accept** command is as follows:

/usr/lib/accept *destination(s)*

Sample Command Use

The sample command line allows printer dqp10_1 to start receiving requests:

```
# /usr/lib/accept dqp10_1<CR>
destination " dqp10_1"  now accepting requests
#
```

reject — Prevent LP Requests

General

Sometimes it is necessary to stop **lp** from routing requests to a destination. For example, if a printer has been removed for repairs, or if too many requests are building at a destination, you may want to prevent new jobs from being queued at this destination. The **reject** command performs this function.

If requests are in the queue at the time the **reject** command is invoked, those requests will be printed as long as the printer is enabled. After the condition that led to denying requests has been corrected, use the **accept** command to allow requests to be received again.

Command Format

The general format of the **reject** command is as follows:

/usr/lib/reject [**–r**[*reason*]] *destinations*

The **-r** option enables you to let users know why requests are being rejected by the specified destination. *Reason* is a brief explanation of the purpose for rejecting requests. If the reason consists of several words separated by spaces, then enclose the reason in double quotes ("").

The *destinations* are the printers that are not to accept requests any longer.

Sample Command Use

The example given here is for a printer, lqp40_1, that had to be taken in for repair. While the printer is gone, you want to prevent **lp** from routing requests to lqp40_1:

```
# /usr/lib/reject -r"printer lqp40_1 needs repair" lqp40_1<CR>
destination "lqp40_1" is no longer accepting requests
#
```

Any users that try to route a job to lqp40_1 will receive the following message:

```
$ lp -dlqp40_1 filename<CR>
lp: can't accept requests for destination "lqp40_1" -
        printer lqp40_1 needs repair
$
```

lpadmin — Configure Printers

General

The **lpadmin** command is used to reconfigure the LP system as needed. Except for a few exceptions, the **lpadmin** command will not alter the LP configuration when the LP scheduler is running.

Command Format

Unlike most other UNIX System commands, the **lpadmin** command requires an option. Of the following three options one must always be included on the command line to execute **lpadmin**:

–d[*dest*]

–x*dest*

–p*printer*

The **–d**[*dest*] option is used to define a system default destination. The destination (*dest*) must already exist. This option can be invoked when the LP scheduler is running.

To remove a destination (*dest*), the **–x***dest* option is used with **lpadmin**. This option can *NOT* be invoked when the scheduler is running.

No other options are allowed with the **–d** and **–x** options. However, there are many options that are used with the **–p***printer* option. These options are as follows:

–c*class* This option assigns the printer specified in the **–p** option to the specified *class*.

–e*printer* This option allows you to use an existing interface program for a new printer that you are adding to the LP

system. When you select this option, the interface program for the printer specified in this option is copied to the new printer.

–h When adding a new printer, this option shows that the printer is hardwired to the 3B2 Computer.

–i*interface* Use this option if you are creating a new interface program for the printer specified in the **–p** option. *Interface* is the path name of the new program.

–l When adding a new printer, this option shows that the device associated with the printer is a login terminal.

–m*model* Several "model" interface programs are supplied with the LP Spooling Utilities. These model interface programs support some of the common printers that may be used with the 3B2 Computer. Use this option to select the model interface program that you want to use with the printer you are adding to the LP system.

–r*class* Use this option to remove a printer from a class.

–v*device* This option must be used when you add a new printer to the LP system. It associates the printer with the UNIX System file specified by *device*. The complete path name must be given for the file.

Sample Command Uses

Example 1

Make printer dqp10_1 the system default destination:

```
# /usr/lib/lpadmin -ddqp10_1<CR>
#
```

Example 2

Add a new printer called dqp10_2 and associate it with device */dev/tty11*. Use the dqp10 model interface program:

```
# /usr/lib/lpadmin -pdqp10_2 -v/dev/tty11 -mdqp10<CR>
#
```

Note: When you add a new printer, it is left in a disabled state and does not accept requests.

Example 3

Create a hardwired printer called lp1 on device */dev/tty13*. Add lp1 to a new class called cl1 and use the same interface program that is used with printer lqp40_1:

```
#  /usr/lib/lpadmin -plp1 -v/dev/tty13 -elqp40_1 -ccl1<CR>
#
```

More examples of using the **lpadmin** command can be found in Chapter 4.

lpmove — Move a Request to Another Printer

General

Occasionally, you may find it necessary to move output requests from one destination to another. For example, if you have a printer that was removed for repairs, you will want to move all the pending job requests to a destination with a working printer. This is done using the **lpmove** command. Be aware that job requests routed to a destination without a printer are automatically rejected.

Another use of the **lpmove** command is to move specific requests from one destination to another destination. However, a word of caution, **lpmove** refuses to move requests while the LP scheduler is running.

Command Format

The general format of the **lpmove** command is as follows:

/usr/lib/lpmove *requests dest*

Requests are the request identification numbers (request ID's) of jobs waiting to be printed, and *dest* is the destination to where the requests are to be moved. The destination can be a printer or a class of printers.

Sample Command Uses

Example 1

Move all the requests for printer lp1 to printer lp2. Moving the requests will rename the request ID's from lp1-nnn to lp2-nnn. After the requests are moved, destination lp1 will no longer be accepting requests:

```
# /usr/lib/lpmove lp1 lp2<CR>
#
```

Example 2

Move requests lp1-54 and lp2-55 to printer dqp10_1:

```
# /usr/lib/lpmove lp1-54 lp2-55 dqp10_1<CR>
total of 2 requests moved to dqp10_1
#
```

Note: The two requests are now renamed dqp10_1-54 and dqp10_1-55.

lpsched — Start the LP Scheduler

General

The **lpsched** command starts the LP scheduler. The LP scheduler takes the top job request off the queue and "hands" it to the appropriate interface program to be printed on a printer. The LP scheduler keeps track of the job progress and as soon as the job is completed it takes the next job request off the queue and repeats the same process. As long as the LP scheduler is running, jobs requested by **lp** will be printed. If the scheduler is not running, jobs will not be printed.

The LP scheduler is started automatically each time the system is turned on. This is done by an executable file called *lp* in the /etc/rc.d directory. The *lp* file is created when the LP Spooling Utilities is installed.

Every time the scheduler is started, **lpsched** creates a file called *SCHEDLOCK* in the /usr/spool/lp directory. As long as the *SCHEDLOCK* file is present, the system will not allow another scheduler to run. When the scheduler is stopped under normal conditions, the *SCHEDLOCK* file is removed. However, if the system is taken down abnormally, there is a possibility that the *SCHEDLOCK* file did not get removed. To ensure that the *SCHEDLOCK* file does not exist, the *lp* file contains a command line to first remove *SCHEDLOCK* before it attempts to start the scheduler.

Command Format

The general format of the **lpsched** command is as follows:

/usr/lib/lpsched

Sample Command Use

Start the LP scheduler:

```
# /usr/lib/lpsched<CR>
#
```

Notice that there is no response to let you know that the scheduler is running. To verify that the scheduler is running, use the **lpstat** command with the **–r** option:

```
# lpstat –r<CR>
scheduler is running
#
Note: If there are a large amount of job
      requests in the queue, it may take
      a while before the lpstat command
      reports that the scheduler is running.
```

lpshut — Stop the LP Scheduler

General

Occasionally, it is necessary to reconfigure the LP system using the **lpadmin** command. Many of the **lpadmin** command options cannot be executed unless the LP scheduler is stopped. The **lpshut** command stops the LP scheduler and ends all printing activity. All requests that were in the middle of printing will be reprinted in their entirety when the scheduler is restarted.

Command Format

The general format of the **lpshut** command is as follows:

```
/usr/lib/lpshut
```

Sample Command Use

Enter the **lpshut** command to stop the LP scheduler:

```
# /usr/lib/lpshut<CR>
scheduler stopped
#
```

Chapter 4

ADMINISTRATION

PAGE

GENERAL 4-1

ADDING AN LP PRINTER 4-1

TYPICAL ADMINISTRATIVE TASKS 4-8

- Change Existing Destinations 4-8
- Assign LP System Default Destination 4-13
- Removing Destinations 4-14

PRINTER INTERFACE PROGRAMS 4-15

- Model Interface Programs 4-15
- Writing Interface Programs 4-15

FILES AND DIRECTORIES 4-19

- /usr/spool/lp/FIFO 4-19
- /usr/spool/lp/default 4-19
- /usr/spool/lp/log 4-19
- /usr/spool/lp/oldlog 4-20
- /usr/spool/lp/outputq 4-20
- /usr/spool/lp/pstatus 4-20
- /usr/spool/lp/qstatus 4-21
- /usr/spool/lp/seqfile 4-21
- /usr/spool/lp/class 4-21
- /usr/spool/lp/interface 4-21
- /usr/spool/lp/member 4-21
- /usr/spool/lp/model 4-22
- /usr/spool/lp/request 4-22
- Lock Files 4-23

CLEANING OUT LOG FILES 4-24

Chapter 4

ADMINISTRATION

GENERAL

This chapter contains information to help you administer the LP system. Included is a procedure for adding a printer to the LP system and examples of using the **lpadmin** command to change existing destinations, assign a default destination, and remove destinations. Also included is information on printer interface programs and a description of the files and directories that make up the LP system.

ADDING AN LP PRINTER

When you install the LP Spooling Utilities, a printer can be easily added to the LP system by following the interactive installation script. If you need to add a printer at a later time, you have to add it manually. This section leads you through the steps required to manually add an LP printer.

> ***Note:*** It is assumed that an Expanded Input/Output Capability feature card, with available ports, is installed in the 3B2 Computer.

1. **Ensure that "lp" can write to device file**

   To avoid unwanted output from non-LP processes and to ensure that **lp** can write to the device, log in as root and enter the following commands:

   chown lp /dev/tty*xx*
   chmod 600 /dev/tty*xx*

   Note: *xx/f1 is the port number that the LP printer will be connected to.*

2. **Change Port Entry in /etc/inittab File**

   When adding an LP printer to the Input/Output (I/O) expansion ports, you may have to make changes to the port entry in the */etc/inittab* file. These changes can be made using Simple Administration subcommands. Enter the command **sysadm** to display the System Administration menu:

```
#sysadm<CR>

                SYSTEM ADMINISTRATION

 1 diagnostics   system diagnostics menu
 2 diskmgmt      disk management menu
 3 filemgmt      file management menu
 4 machinemgmt   machine management menu
 5 packagemgmt   package management
 6 softwaremgmt  software management menu
 7 syssetup      system setup menu
 8 ttymgmt       tty management menu
 9 usermgmt      user management menu

Enter a number, a name, the initial part of a name, or
? or <number>? for HELP, q to QUIT:
```

Enter an **8** to select the tty management menu. The response is:

```
                        TTY MANAGEMENT

  1 baud            change the baud rate on a tty line
  2 disable         turn off a tty line
  3 enable          turn on a tty line

Enter a number, a name, the initial part of a name, or
? or <number>? for HELP,  ^ to GO BACK,  q to QUIT:
```

Enter a **2** to turn off the getty process on a tty line. The response is:

```
        The following is a list of the changeable tty lines:

                NAME            STATUS          BAUD
                contty          respawn         9600
                tty11           respawn         9600
                tty12           off             1200
                tty13           off             1200
                tty14           off             4800

Enter tty line (11-14, 21-24, 31-34, 41-44, contty):
```

Note: If a tty line cannot be changed, it will not appear on this list. For example, a tty line being used for basic networking will not appear.

Enter the name of the tty line (port) that you are connecting to the printer. The response is:

```
This is the tty line before the change.
xx:2:respawn:/etc/getty ttyxx 9600

This is the tty line after the change.
xx:2:off:/etc/getty ttyxx 9600

Do you want to see the table again? [y, n]
```

Note: The variable xx is the port number that the LP printer is being connected.

Enter **n**. The response is:

Do you want to do more? [y, n]

Enter **n**. The response is:

Press the RETURN key to see the ttymgmt menu [?, ^, q]:

You are now finished changing the tty line; so enter a **q** to quit. The response is the shell prompt (#). The port line in the */etc/inittab* file should now look something like this:

12:2:off:/etc/getty tty12 9600

3. **Turn off LP scheduler**

To execute the **lpadmin** command, the LP scheduler cannot be running; so enter the command:

lpstat -r

to find out if the scheduler is running. If the scheduler is running, enter the following command to stop it.

/usr/lib/lpshut

4. **Introduce printer to LP system**

The **lpadmin** command is used to add a new printer to the LP system. The format of the **lpadmin** command is as follows:

lpadmin -p*printer* –v*device* [–e*printer*¦–i*interface*¦–m*model*]

When adding a printer, you need to furnish a printer name, a device file, and an interface program. The *printer* name must conform to the following rules:

- No longer than 14 characters.
- Consists solely of alphanumeric characters and underscores.
- The name must be uncommon and cannot be the name of an existing LP destination (printer or class).

Device is the path name of the special UNIX System device file that the printer is associated with, for example, /dev/tty11.

The interface program is chosen from the following:

- One model interface program supplied with the LP Spooling Utilities (**-m***model*).
- An interface program that is being used by an existing printer (**-e***printer*).
- An interface program that you have created (**-i***interface*).

The following is an example command line for adding printer dqp10_3 to the LP system. The interface program used in the sample command line is the dqp10 model interface program and the I/O port is tty12.

/usr/lib/lpadmin –pdqp10_3 –v/dev/tty12 –mdqp10

Other options that you can include when creating a new printer are the **-h**, **-l**, and **-c** options. The **-h** option shows that the device for the printer is hardwired, or the device is the name of a file (default). The **-l** option is used if the device is the path name of a login terminal. Using the **-l** option in the command line tells the LP scheduler to automatically disable the printer each time the scheduler starts running. The **-c** option adds the printer to an existing class or to a new class. Class names must conform to the same rules that apply to printer names.

5. **Start LP scheduler**

   Enter the command

   /usr/lib/lpsched

   to restart the LP scheduler.

6. **Allow printer to accept job requests**

   A printer that is added to the LP system when the LP Spooling Utilities is installed is enabled and accepting requests. However, when a printer is added to the LP system using the **lpadmin** command, it is disabled and does not accept requests routed to it. When you are ready for the printer to accept requests, enter the following command:

   /usr/lib/accept *printer-name*

7. **Enable printer**

   When you are ready to start printing, be sure that the printer is ready to receive output. For several printers, this means that the top of form has been adjusted and that the printer is on-line. Enter the command:

   enable *printer-name*

   to enable printing to occur on the printer.

TYPICAL ADMINISTRATIVE TASKS

This section covers some of the administrative type tasks that you may have to do on the LP Spooling system: for example, establishing a default destination, change existing destinations, and removing destinations.

Change Existing Destinations

Changes to existing destinations are done using the **lpadmin** command. Changes must always be made with respect to a printer name (**-p***printer*). The changes may be one or more of the following:

1. The device for a printer may be changed using the **-v** option (**-v***device*). If this is the only change being made, then this may be done while the LP scheduler is running. This ability to change devices while the scheduler is running helps changing devices for login terminals.

2. A printer interface program may be changed using the **-m**, **-e**, **-i** options (**-m***model*, **-e***printer*, **-i***interface*).

3. A printer may be specified as a hardwired printer (**-h**) or as a login terminal (**-l**).

4. A printer may be added to a new or existing class using the **-c** option (**-c***class*).

5. A printer may be removed from an existing class with the **-r** option (**-r***class*).

The next examples shows the use of the **lpadmin** options:

Example 1

This example shows how to change the device for printer dqp10_1 to */dev/tty13*. Since the device is the only change being made, it can be done while the LP scheduler is running:

```
# /usr/lib/lpadmin -pdqp10_1 -v/dev/tty13<CR>
#
```

Example 2

Change the interface program for printer lp1 to model interface program *dqp10*. The LP scheduler has to be stopped before the **lpadmin** command can be executed:

```
#/usr/lib/lpshut<CR>
scheduler stopped
#  usr/lib/lpadmin -plp1 -mdqp10<CR>
#
```

Example 3

Define printer dqp10_1 as a hardwired printer:

```
# /usr/lib/lpadmin -pdqp10_1 -h<CR>
#
```

Example 4

Add printer dqp10_2 to class cl1:

```
# /usr/lib/lpadmin -pdqp10_2 -ccl1<CR>
#
```

Printers that are added to a class are ordered according to the sequence that they are added. For example, assume that class cl1, in the example above, already had printers lp1 and lp2 as members. After adding printer dqp10_2, the printers would be lp1, lp2, and dqp10_2. If all three printers are available and a request is routed to class cl1, the request will be serviced by lp1. If all three printers are busy, the request will be serviced by the first available printer.

Example 5

Remove printers lp1 and lp2 from class cl1:

```
# /usr/lib/lpadmin -plp1 -rcl1<CR>
# /usr/lib/lpadmin -plp2 -rcl1<CR>
#
```

Example 6

The previous examples showed only one configuration change per command line. However, several configuration changes can be made in the same command line:

```
# /usr/lib/lpadmin -plp1 -mlqp40 -v/dev/tty14 -ccl2<CR>
#
```

The sample command line makes the following changes to printer lp1:

- Changes the interface program to the model interface program *lqp40*.
- Changes the device to */dev/tty14*.
- Adds the printer to a class called cl2.

Assign LP System Default Destination

The **lp** command determines the destination of a request by checking for a **-d***dest* option on the command line. If no **-d** option is present, it checks to see if the environment variable **LPDEST** is set. If **LPDEST** is not set, then the request is routed to the system default destination.

The system default destination for the LP system can be a printer or printer class. There are two ways you can assign a default destination: during the installation of the LP Spooling Utilities or using the **lpadmin** command with the **-d** option. To assign a default destination, the destination must already exist.

The following example shows the command line that is used to assign printer dqp10_2 as the destination printer. The **-d** option is a **lpadmin** option that can be invoked when the LP scheduler is running:

```
# /usr/lib/lpadmin –ddqp10_2<CR>
#
```

Setting the environment variable **LPDEST** allows a user to have a default destination other than the system default destination. The example below shows how to set **LPDEST**:

```
$ env LPDEST=lqp40_1<CR>
$
```

The default destination for this user is lqp40_1.

Removing Destinations

No destination (class or printer) may be removed if it has pending requests. The pending requests must either be canceled using the **cancel** command or moved to other destinations using the **lpmove** command before the destination can be removed.

Removing the last remaining member of a class causes the class to be deleted. If the destination removed is the system default destination, then the system will no longer have a default destination.

When the last remaining member of a class is removed, then the class is also removed. However, the removal of a class does not imply the removal of printers that were assigned to that class.

Example

Remove destination lp3 from the LP system:

```
# /usr/lib/lpadmin -xlp3<CR>
#
```

PRINTER INTERFACE PROGRAMS

Printers that are used as LP spooling printers must have a printer interface program. Every print request made with the **lp** command is routed through the appropriate printer interface program before the request is printed on a line printer. The printer interface program to use is defined by the **lpadmin** command.

Model Interface Programs

Each type of printer requires an uncommon interface program. Several interface programs, referred to as "model" interface programs, are furnished with the LP Spooling Utilities. The furnished interface programs support the DQP-10 printer, the LQP-40 printer, and several other popular printers. The model interface programs are written as shell procedures, but they can be written as C programs or any other executable program. These programs are located in the /usr/spool/lp/model directory.

Writing Interface Programs

If you have a printer that is not supported by a model interface program, you will have to write your own program. The shell script for a "dumb" printer interface program is provided in Figure 4-1. Use this interface program as a model and change it to meet your particular needs. The information that follows should provide some help in creating your interface program.

When the LP scheduler routes an output request to a printer, the interface program for the printer is invoked in the directory */usr/spool/lp* as follows:

interface/P *id user title copies options file ...<CR>*

Arguments for the interface program are:

P printer name

id	request ID returned by lp
user	logname of user who made the request
title	optional title specified by the user
copies	amount of copies requested by user
options	blank-separated list of class or printer-dependent options specified by user
file	full path name of a file to be printed.

When the interface program is invoked, its standard input comes from */dev/null* and both the standard output and standard error output are directed to the printing device. Interface programs format their output based on the command line arguments. You want to ensure that the interface program has the proper stty modes (terminal characteristics such as baud rate, output options, etc). You may do this by adding **stty** command lines of the form:

stty *mode options* **<&1**<*CR*>

This command line takes the standard input for the **stty** command from the device. An example of an **stty** command line that sets the baud rate at 1200 and sets some of the option modes is shown below:

```
stty -parenb -parodd 1200 cs8 cread clocal ixon 0<&1
```

Since different printers have different numbers of columns, make sure that header and trailer for your interface program correspond to your printer.

When printing is complete, it is the responsibility of the interface program to exit with a code that shows the status of the print job.

Exit codes are interpreted by **lpsched** as follows:

CODE	MEANING TO LPSCHED
0	The print job has completed successfully.
1 to 127	A problem was encountered in printing this particular request (for example, too many nonprintable characters). This problem will not affect future print jobs. The **lpsched** command notifies users by mail that there was an error in printing the request.
greater than 127	These codes are reserved for internal use by **lpsched**. Interface programs must not exit with codes in this range.

When problems occur that are likely to affect future print jobs (for example, a device filter program is missing) you should have your interface program disable printers so that print requests are not lost. When a busy printer is disabled, the interface program will be ended with signal 15.

```
# lp interface for dumb line printer
#
#       SCCS @(#)dumb   1.1

x=" XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
echo " \014\c"
echo " $x\n$x\n$x\n$x\n"
banner " $2"
echo " \n"
user='grep " ^$2:"  /etc/passwd ¦ line ¦ cut -d: -f5'
if [ -n " $user"  ] .
then
        echo " User: $user\n"
else
        echo " \n"
fi
echo " Request ID; $1    Printer: 'basename $0'\n"
date
echo " \n"
if [ -n " $3"  ]
then
      banner $3
fi
copies=$4
echo " \014\c"
shift; shift; shift; shift
files=$x"
i=1
while [ $i -le $copies ]
do
        for file in $files
        do
                cat " $file"  2>&1
                echo " \014\c"
        done
        i='expr $i + 1'
done
echo " $x\n$x\n$x\n$x\n$x\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n
echo " \n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n$x\n$x\n$x\n$x\n$x"
echo " $x\n$x\n$x\n$x\n$x\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n"
echo " \n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\" n
echo " $x\n$x\n$x\n$x\n$x\n$x\n$x\n$x\n$x\n$x\n"
exit 0
```

Figure 4-1. Dumb Line Printer Interface Program

FILES AND DIRECTORIES

This section describes the files and directories in the LP Spooling structure.

/usr/spool/lp/FIFO

FIFO is a special file that all the commands use to send messages to **lpsched**. Any of the LP commands may write to *FIFO*, but only **lpsched** may read it.

/usr/spool/lp/default

This file contains the name of the system default destination. If this file does not exist or if it is empty, the LP system has no default destination.

/usr/spool/lp/log

The purpose of the *log* file is to keep a record of all the printing activity that has taken place since the LP scheduler was last started. This file contains:

- *logname* of the user who made the request
- request ID
- name of the printer that the request was printed on
- date
- time that the printing started.

Any **lpsched** error messages that occur are also recorded. The first line of the log file shows the time that the LP scheduler was started.

/usr/spool/lp/oldlog

The *oldlog* file contains a record of what was in the *log* file. When the scheduler is stopped, the *log* file is closed. When the scheduler is restarted, all the information that had accumulated in the *log* file is copied to the *oldlog* file and a new *log* file is started. Any information that had been in the *oldlog* file is overwritten. The first line of the file identifies the time that the scheduler was turned on, and the last line shows the time the scheduler was turned off.

/usr/spool/lp/outputq

When an output request is made by the **lp** command, an entry is made in this binary file. The LP scheduler takes the job request and hands it to the appropriate interface program to be printed. After the job is completed, the job request is removed, and the scheduler takes the next job request from this file and has it printed. Only those requests that have been made since the last time the LP scheduler was started are contained in the file.

Entries in *outputq* may be modified by the **lpmove**, **disable**, and **lpsched** command. The **cancel**, **disable**, and **lpsched** commands can mark entries in this file "deleted." If a job request is deleted before the job is completed, the entry will remain in the file.

/usr/spool/lp/pstatus

The binary file *pstatus* contains status information for each printer. Entries are added and removed from this file by the **lpadmin** command and are modified by the **cancel**, **enable**, **disable**, and **lpsched** commands. When the **lpstat** command is invoked with the **-p** option, printer status information is obtained from this file.

/usr/spool/lp/qstatus

This binary file keeps track of whether a destination is accepting or rejecting requests. Entries are added or removed from this file by the **lpadmin** command and modified by the **accept**, **reject**, and **lpmove** commands. When the **lpstat** command is invoked with the **-o** option, the request status is obtained from this file.

/usr/spool/lp/seqfile

The *seqfile* file contains the sequence number of the last request ID that was assigned by the **lp** command. The sequence number is incremented by **lp** for each request. When the number 9999 is reached, the sequence number is reset to 1.

/usr/spool/lp/class

This is a directory that contains one file for each LP class that has been identified. (The name of the file is the same as the name of the class.) The file identifies each member, here an LP printer, that is assigned to the class. Class files are created, modified, and deleted by the **lpadmin** command. Every class file must always have at least one member.

/usr/spool/lp/interface

The interface directory contains one executable interface program for each printer that is in the LP system. The file name of the interface program is the same as the printer name. The interface program is invoked with its standard error output directed to the printer. Interface programs may be shell procedures or compiled C programs.

/usr/spool/lp/member

The member directory contains one file for each LP printer. The file name is the same as the printer name.

/usr/spool/lp/model

This is a directory that contains the printer interface programs that are distributed with the LP Spooling Utilities.

/usr/spool/lp/request

This directory contains one subdirectory for each destination in the LP system. The name of the subdirectory is the same as the name of the destination. When an **lp** request is made, a *request* file (or "r" file) and, usually, a *data* file (or "d" file) are created in the subdirectory of the destination to where the request is going. The *data* file stores the file to be printed until the scheduler is ready to print it. A *data* file is not created if the file to be printed cannot be linked to the request subdirectory.

The name of the request file is derived from the request identification number and is of the form *r-seqno*. The name of the data file is of the form *dn-seqno*, where *n* is a non-negative integer.

The request and data files are deleted by the **cancel**, **disable**, and **lpsched** commands and may be moved from one subdirectory to another by the **lpmove** command.

Lock Files

To guarantee LP commands exclusive access to data files, several "lock" files are maintained in the LP system. They are binary files that contain the process ID of the locking process. The lock files and their associated data files are:

Lock File	*Data File*
OUTQLOCK	outputq
PSTATLOCK	pstatus
QSTATLOCK	qstatus
SEQLOCK	seqfile

Lock files "expire" after a given interval and may be unlinked by any LP process. Thus, commands that lock a data file for longer than this interval must update the modification time on the lock file. The creation, updating, and unlinking of lock files is handled automatically by the LP low level file access routines.

Another lock file, SCHEDLOCK, is present while the LP scheduler is running to ensure that only one invocation of **lpsched** is active. Unlike other lock files, SCHEDLOCK has no expiration time.

CLEANING OUT LOG FILES

A history of LP printing activities is kept in the */usr/spool/lp/log* file. The information is stored in the *log* file until a shutdown and a restart of the LP scheduler occurs. When this happens, the *log* file is copied to */usr/spool/lp/oldlog*, and a new *log* file is started.

If the scheduler is not stopped for long periods of time and if you have a large amount of LP requests, the *log* file could grow to be a large file. You can manually remove the contents of this file, or you can let the system do it for you on a scheduled basis.

To have the system clean out the log file, create an entry in the files in the /usr/spool/cron/crontabs directory. One way to do this is to log in as "root" and use the **crontab** command. The other way is to edit the *crontabs* directory files.

The example below shows some typical **crontab** command lines. **Crontab** adds these command lines to the *root* file in the /usr/spool/cron/crontabs directory. Every Friday at 11:00 PM **cron** executes these commands. First, the contents of the *log* file are copied to the *oldlog* file, and then the *log* file is cleaned out:

```
# crontab<CR>
0 23 * * 5 /bin/su lp -c " cp /usr/spool/lp/log /usr/spool/lp/oldlog"
1 23 * * 5 /bin/su lp -c " >/usr/spool/lp/log"
<CTRL d>
#
```

The **crontab** command is covered in the *AT&T 3B2 Computer User Environment Utilities Guide* and the *AT&T 3B2 Computer System Administration Utilities Guide.*

Appendix

ERROR MESSAGES

This appendix provides a description of the error messages that are associated with LP commands. The following variables are used in the error messages:

file(s)	Shows the file or files that are to be printed.
dest	Shows the name of the destination printer.
printer-id	Shows the request identification of the printout. For example, *dqp10_2-46* is the printer name followed by the request identification number.
printer-name	Shows the name of the printer.
program-name	Shows the program name that was executed.
user	Shows the user who requested the printout.

Refer to the *AT&T 3B2 Computer User Reference Manual* for UNIX System V manual pages supporting the commands described in this appendix.

The messages are shown in bold type, and the variables are shown in italics. For example,

destination *dest* **non-existent**

Following each message is an explanation of the probable cause of the error and the corrective action to take. If you are not able to correct all the error conditions you encounter, call your service representative for help.

Some lengthy error messages, that appear all on one line on the display, are too long to be shown as one line in the documentation. When more than one line is required in the documentation to show a one line error message, a "\" is used to split the message.

''*dest***'' is an illegal destination name**

The *dest* you used is not a valid destination name. Use the **lpstat -p** command to list valid destination names.

"*file***" is a directory**

The file name you typed is a directory and cannot be printed.

"*xx***" is not a request ID or a printer**

The argument you used with the **cancel** command is not a valid request identification number or a printer name. Use the **lpstat -t** command to give you all the printers and requests waiting to get printed.

"*xx***" is not a request ID**

The request identification number you used with the **lpmove** command is not a valid request identification number. To find out what requests are valid, use the **lpstat -u** command.

"*xx*" not a request ID or a destination

You used an invalid request identification number or destination with the **lpstat** command. To find out what is valid, use the **lpstat -t** command.

dest **not accepting requests since** *date*

Requests to the printer that you are trying to use have been stopped by the **reject** command.

Can't access FIFO

The named pipe file /usr/spool/lp/FIFO is incorrect. The mode should be 600.

LP Administrator not in password file

You must have an entry in the /etc/passwd file for "lp," and you must belong to the group "bin."

acceptance status of destination "*printer-name***" unknown**

Use the **accept** command to enable the printer so that it will accept requests.

can't access file "*xx***"**

The mode could be wrong on your directory or the file that you are trying to access.

can't create class "*xx***"—it is an existing printer name**

The class name you are trying to use has already been given to a printer. You will have to use another name or remove the printer to use the class name.

can't create new acceptance status file

The mode may be wrong on the /usr/spool/lp directory. It should be 755 with the owner "lp" and the group "bin."

can't create new class file

The mode may be wrong on the /usr/spool/lp directory. It should be 755 with the owner "lp" and the group "bin."

can't create new interface program

The mode may be wrong on the /usr/spool/lp/interface directory. It should be 755 with the owner "lp" and the group "bin."

can't create new member file

The mode may be wrong on the /usr/spool/lp/member directory. It should be 755 with the owner "lp" and the group "bin."

can't create new printer status file

The mode may be wrong on the /usr/spool/lp/pstatus. It should be 644 with the owner "lp" and the group "bin."

can't create new request directory

The mode may be wrong on the /usr/spool/lp/request directory. It should be 755 with the owner "lp" and the group "bin."

can't create printer "*printer–name***" — it is an existing class name**

The printer-name you are trying to use has already been used as a class name. You will have to assign another name for the printer.

can't create new output queue

The mode on the file /usr/spool/lp/seqfile is incorrect. It should be 644, and the mode on the directory should be 755. The owner should be "lp," and the group should be "bin." This may be corrected by typing the command at a later time.

can't create new sequence number file

The mode on the file /usr/spool/lp/seqfile is incorrect. The mode of the file should be 644, and the mode of the directory should be 755. The owner should be "lp," and the group should be "bin." This may be corrected by typing the command at a later time.

can't create request file *xx*

The mode on the file /usr/spool/lp/request/*printer–name*/r–id is incorrect. *Printer–name* is the name of the printer such as dqp10, and r–id is the request identification number. The mode of the file should be 444, and the mode of the directory should be 755. The owner should be "lp," and the group should be "bin." This may be corrected by typing the command at a later time.

can't fork

You either have several processes running and are not allowed to run anymore, or the system has all the processes running that it can handle. You will have to rerun this command later.

can't lock acceptance status

This is a temporary file in /usr/spool/lp that prevents more than one "lp" request from being taken at any given instant. You will have to rerun this command later.

can't lock output queue

The file /usr/spool/lp/QSTATLOCK prevents more than one "lp" request from being printed on a printer at a time. You will have to rerun this command later.

can't lock printer status

The temporary file /usr/spool/lp/PSTATLOCK prevents more than one "lp" request from being printed on a printer at a time. You will have to rerun this command later.

can't lock sequence number file

The file /usr/spool/lp/SEQLOCK prevents more than one "lp" request from getting the next printer–id (request identification) number at a time. You will have to rerun this command later.

can't move request *printer–id*

Printer–id is the request identification number that cannot be moved. You will probably have to change the modes on the files and directories in /usr/spool/lp/request. Also, you will have to manually move the request from the disabled printer directory to the new destination after you shut down the LP scheduler.

can't open class file

The **lp** program is trying to access the list of classes for printers. One reason it may not be able to open the class file is that the system could have the maximum amount of files open that are allowed at any time. This can be corrected by typing the command at a later time.

can't open member file

The **lp** program is trying to access the list of members in the directory /usr/spool/lp/member. The system could have the maximum amount of files open that are allowed at any time. This can be corrected by typing the command at a later time.

can't open *xx* file in MEMBER directory

There are a couple of reasons why file *xx* in the /usr/spool/lp/member directory cannot be opened. The mode on the file could be incorrect. It should be 644. Another possibility is that the system could have the maximum amount of files open that are allowed at any time. This can be corrected by typing the command at a later time.

can't open *xx* file in class directory

One possibility why file *xx* cannot be opened is that the mode on the file or directory is incorrect. The file mode should be 644, and the directory mode should be 755. Another possibility is that the system has the maximum amount of files open that are allowed at any time. This problem can be corrected by typing the command at a later time.

can't open *xx*

You cannot print on printer *xx* because the mode is incorrect on /dev/tty. The mode should be 622.

can't open FIFO

The mode on the named pipe file /usr/spool/lp/FIFO may be incorrect. It should be 600. Or, the system could have the maximum amount of files open that are allowed at any time. This problem can be corrected by typing the command at a later time.

can't open MEMBER directory

The mode on the directory /usr/spool/lp/member could be incorrect. It should be 755. Another possibility is that the system could have the maximum amount of files open that are allowed at any time. If the maximum amount of files are open, try typing the command at a later time.

can't open acceptance status file

The mode on the file /usr/spool/lp/qstatus may not be correct. It should be 644. Another possibility is that the system could have the maximum amount of files open that are allowed at any time. This problem can be corrected by typing the command at a later time.

can't open default destination file

Check the mode on the file /usr/spool/lp/default. The mode should be 644. If the mode is okay, it could be that the system has the maximum amount of files open that are allowed at any one time. This can be corrected by trying the command at a later time.

can't open file *filename*

The *filename* was incorrectly typed or you don't have the correct modes set. The mode should be at least 400 if you are the owner.

can't open output queue file

Check the mode on the file /usr/spool/lp/outputq. It should be 644. This error message could also be generated if the system has the maximum amount of files open that are allowed at any one time. Try entering the command at a later time.

can't open printer status file

The mode on the file /usr/spool/lp/pstatus is incorrect. The mode should be 644. It could also be that the system has the maximum amount of files open that are allowed at any one time. This can be corrected by trying the command at a later time.

can't open request directory *directory name*

The mode on the directory /usr/spool/lp/request is incorrect. The mode should be 655. It could also be that the system has the maximum amount of files open that are allowed at any one time. This can be corrected by trying the command at a later time.

can't open request file *xx*

The mode on the file /usr/spool/lp/member/request/*xx* is incorrect. The mode should be 644. It could also be that the system has the maximum amount of files open that are allowed at any one time. This can be corrected by trying the **lpmove** command at a later time.

can't open system default destination file

The mode on the file /usr/spool/lp/default is incorrect. The mode should be 644. It could also be that the system has the maximum amount of files open that are allowed at any one time. This can be corrected by trying the command again at a later time.

can't open temporary output queue

The mode on the file /usr/spool/lp/outputq is incorrect. The mode should be 644. It could also be that the system has the maximum amount of files open that are allowed at any one time. This can be corrected by trying the command at a later time.

can't proceed — scheduler running

Many of the **lpadmin** command options cannot be executed while the scheduler is running. Stop the scheduler using the **lpshut** command and then try invoking the command again.

can't read current directory

The **lp** and **lpadmin** commands cannot read the directory containing the file to be printed. The directory name may be incorrect or you do not have read permission on that directory.

can't remove class file

The mode may be wrong on the /usr/spool/lp/class. It should be 755. The owner should be "lp," and the group should be "bin." Another possibility is the file in that directory may have the wrong mode. It should be 644.

can't remove printer

The mode may be wrong on the /usr/spool/lp/member directory. It should be 755, and the files in that directory should be 644. Both the directory and the files should be owned by "lp," and the group should be "bin."

can't remove request directory

The mode may be wrong on the /usr/spool/lp/request directory. It should be 755 and should be owned by "lp," and the group should be "bin." The directory may still have pending requests to be printed. These requests will have to be removed before the directory can be removed.

can't set user id to LP Administrator's user id

The **lpsched** and **lpadmin** commands can only be used when you are logged in as "lp" or "root."

can't unlink old output queue

The **lpsched** program cannot remove the old output queue. You will have to remove it manually by using the command **rm /usr/spool/lp/outputq**.

can't write to *xx*

The **lpadmin** command cannot write to device *xx*. The mode is probably wrong on the /dev/ttyxx file. It should be 622 and owned by "lp."

cannot create temp file *filename*

The system may be out of free space on the /usr file system. Use the command **df /usr** to determine the number of free blocks. Several hundred blocks are required to insure that the system will function correctly.

class "*xx***" has disappeared!**

Class *xx* was probably removed since the scheduler was started. The system may be out of free space on the /usr file system. Use the command **df /usr** to find out. Use the **lpshut** command to stop the scheduler and restore the class from a backup.

class "*xx***" non-existent**

The class *xx* may have been removed because the system is out of free space on the /usr file system. Use the command **df /usr** to find out how much free space is available. The class will probably have to be restored from a backup.

class directory has disappeared!

The /usr/spool/lp/class directory has been removed. The system may be out of free space on /usr; use the **df /usr** command to find out. The class directory contains all the data for each printer class. To restore this directory, get these files and directory from a backup.

corrupted member file

The /usr/spool/lp/member directory has a corrupted file in it. You should restore the directory from backup.

default destination "*dest***" non-existent**

Either the default destination is not assigned or the printer *dest* has been removed. Use the **lpadmin** to set up a default destination or set **LPDEST** to the value of the destination.

destination "*dest***" has disappeared!**

A destination printer, *dest* has been removed since **lpsched** was started. Use the **lpadmin** command to remove the printer.

destination "*printer-name***" is no longer accepting requests**

The printer has been disabled using the **reject** command. The printer can be re-enabled using the **accept** command.

destination *dest* **non-existent**

The destination printer you specified as an argument to the **accept** or **lpadmin** command is not a valid destination name, or it has been removed since the scheduler was started.

destination "*printer-name***" was already accepting requests**

The destination printer was previously "enabled." Once a printer is accepting requests, issuing any more **accept** commands to it are ignored.

destination "*printer-name***" was already not accepting requests**

A **reject** command was already sent to the printer. Use the **accept** command to allow the printer to start accepting requests again.

destination *printer-name* **is not accepting requests**
move in progress ...

The printer has been disabled by the **reject** command, and requests are being moved from the disabled printer to another printer. The printer can be enabled again by the **accept** command.

destinations are identical

When using the **lpmove** command, you need to specify a printer to move the print requests from and a different printer to move the requests to.

disabled by scheduler: login terminal

The login terminal has been disabled by the LP scheduler. The printer can be re-enabled by using the **enable** command.

error in printer request *printer–id*

Printer–id is the request identification number. The error was most likely caused by an error in the printer. Check the printer, and reset it if needed.

illegal keyletter "*xx*"

An invalid option, *xx*, was used.

keyletters "-" and "*–yy*" are contradictory

This combination of options to the **lpadmin** program cannot be used together.

keyletter "*xx*" requires a value

The option *xx* requires an argument. For example, in the command line

lpadmin -m*model*

the argument to the **-m** option is the name of a model interface program.

keyletters -e, -i, and -m are mutually exclusive

These options to the **lpadmin** command cannot be used together.

lp: *xx*

In this message the variable *xx* could be one of several arguments. Typically, it is a message telling you the default destination is not assigned.

member directory has disappeared!

The /usr/spool/lp/member directory has been removed. The system is probably out of free disk space in the /usr file system. You need to clean up the /usr file system, and then install the LP commands or retrieve them from a backup.

model "*xx*" non–existent

The name that you are using for a model interface program is not a valid one. A list of valid models is in the /usr/spool/lp/model directory.

new printers require -v and either -e, -i, or -m

A printer must have an interface program, and this is specified by **-e**, **-i**, or **-m** options. The **-v** option specifies the device file for the printer. For more information on these options, refer to the **lpadmin** in the *AT&T 3B2 Computer User Reference Manual.*

no destinations specified

There are no destination printers specified. Use the **lpadmin** command to set one up.

no printers specified

There are no printers specified. Use the **lpadmin** command to set one up.

non-existent printer *xx* **in PSTATUS**

A printer with the name *xx* is in the /usr/spool/lp/pstatus file, but no longer exists. The printer should be removed using the **lpadmin** command.

non-existent printer *printer–name* **in class** *xx*

The printer that you are trying to address in class *xx* has been removed from that class.

out of memory

The message implies that there is not enough memory to contain the text to be printed.

printer "*printer–name***" already in class "***xx***"**

The printer you are trying to move to class *xx* is already in that class. You cannot move a printer to a class that it is already in.

printer "*printer-name***" has disappeared!**
or **printer "***printer-name***" has disappeared**

The printer has been removed, and the **enable** command cannot find it. The printer was most likely removed since the machine was rebooted or since the scheduler was started.

printer "*printer–name***" non–existent**

Printer–name is the name of a printer that has been removed since the scheduler has been started. You must use the **lpadmin -x***printer–name*.

printer status entry for "*printer–name***" has disappeared**

The /usr/spool/lp/pstatus file must have been corrupted. You will have to resubmit the printer request.

printer "*printer-name***" was not busy**

The printer is not printing a request at this time. Either the request you wanted to cancel is finished printing, or you have specified the wrong printer.

request "*printer–id***" non-existent**

You are attempting to cancel a request that does not exist. You may have given the wrong printer name or wrong request identification number or the request may have finished printing.

request not accepted

The request was not accepted by **lp**. The scheduler may not be running. Use the **lpstat -t** command to find out more information.

requests still queued for "*printer–name***" — use lpmove**

Printer–name is the printer that still has requests waiting to get printed. You need to use the **lpmove** command to get those requests moved to another printer.

scheduler is still running — can't proceed

You cannot do this command while the scheduler is running. You will have to use the **lpshut** command first.

spool directory non-existent

The directory /usr/spool has been removed. You will have to use the **mkdir** command to restore the directory. This has probably removed some of the necessary LP files. You may have to reinstall the LP commands.

standard input is empty

You specified an invalid file name either by incorrectly typing a name or by specifying a nonexistent file. Nothing will be printed on the printers from this request.

this command for use only by LP Administrators

This command is restricted to someone logged in as "root" or "lp." Refer to Chapter 2 for the commands that can be used by general users, and refer to Chapter 3 for the commands that are used by "root" or "lp" to administer the LP System.

too many options for interface program

The **lp** command called the appropriate interface program with too many arguments. For more information on the options and arguments that can be used with the **lp** command, refer to the *AT&T 3B2 Computer User Reference Manual.*

unknown keyletter "*xx*" or unknown keyletter "-*xx*"

An invalid option was supplied to the **lp** or **lpadmin** commands.

unknown option "*xx*"

This message is displayed in response to an invalid option supplied to the **disable**, **lpstat**, or **reject** commands.

usage: disable [-c] [-r[reason]] printer ...

The syntax for the **disable** command is not correct. The valid options are: **-c** to cancel the currently printing request, and **-r** followed by the reason that you are disabling the printer.

usage: reject [-r[reason]] dest ...

The syntax for the **reject** command is not correct. The proper format is to specify the reason the printer is not taking any more print requests and to identify the destination printer.

usage: accept dest ...

The syntax for the **accept** command is to specify a destination printer. You are setting up a printer to accept requests, and you did not specify what printer should accept requests.

usage: enable printer ...

The syntax for the **enable** program is to specify a destination printer.

usage: cancel id .... printer ...

The syntax for the **cancel** command is not correct. The proper format is to specify the request identification number or the printer name.

```
usages: lpadmin -pprinter [-vdevice] [-cclass] [-rclass]
        [-eprinter | -iinterface | -mmodel] [-h | -l]
                    -or-
        lpadmin -d[destination]
                    -or-
        lpadmin -xdestination
```

The correct syntax for the **lpadmin** command is to specify at least one option mentioned above.

your printer request *printer-id* **was canceled by** *user*

The printer request did not finish printing because another *user* canceled it. Typically, you will get this message in your mail. One reason a person may cancel a request other than their own is because the request is not printing correctly.

Replace this

page with the

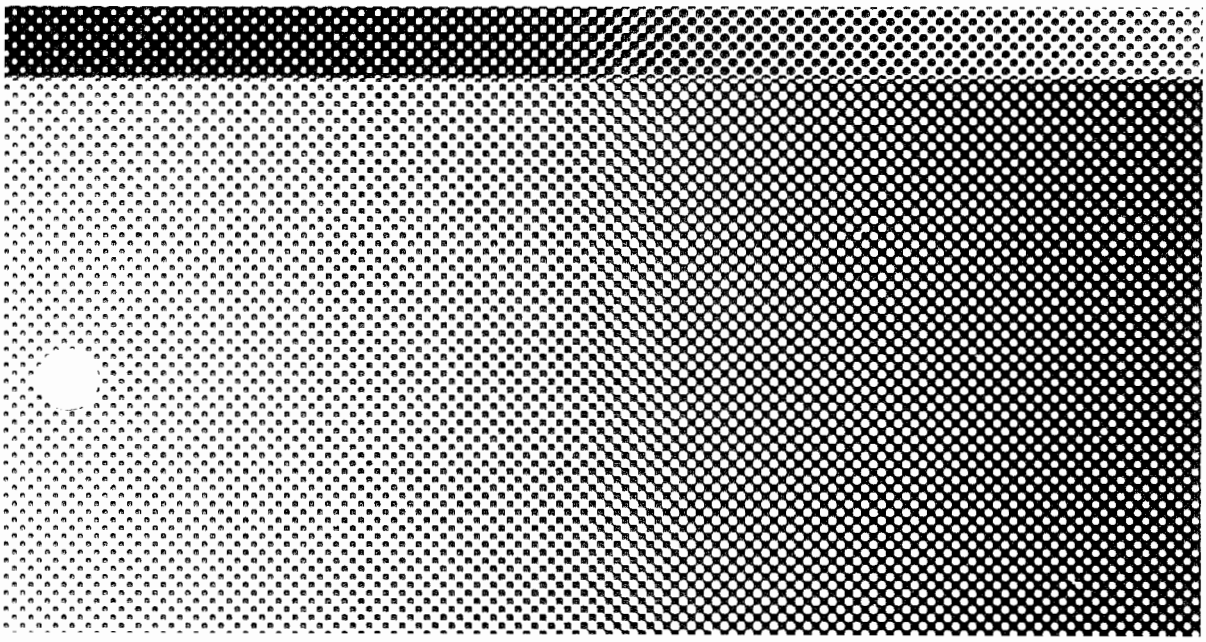
PERFORMANCE MEASUREMENTS

tab separator.

AT&T

AT&T 3B2 Computer

UNIX™ System V Release 2.0 Performance Measurements Utilities Guide



CONTENTS

Chapter 1. INTRODUCTION

Chapter 2. REQUIREMENTS

Chapter 3. COMMAND DESCRIPTIONS

Chapter 4. TUNING AND CONFIGURATION

Chapter 1

INTRODUCTION

PAGE

GENERAL 1-1
SYSTEM ACTIVITY 1-2
KERNEL PROFILING 1-3
GUIDE ORGANIZATION 1-4

Chapter 1

INTRODUCTION

GENERAL

This guide describes command formats (syntax) and use of Performance Measurement Utilities available for your AT&T 3B2 Computer.

The Performance Measurement Utilities are for use by the person responsible for administrating the system or software developers. To use any command beginning with **prf**, you must be logged in on the system as **root**. All other commands are available to all users.

The Performance Measurement Utilities address two areas: (1) system activity and (2) kernel profiling. The utilities provide information that can be used in load balancing, performance analysis, and system tuning. This guide describes the changes required to support and use these utilities on the 3B2 Computer. **UNIX*** System setup and command usages are also explained.

---

* Trademark of AT&T

SYSTEM ACTIVITY

In the area of system activity, the utilities support the collection of system-wide data and provide tools to generate several different types of reports. The data is collected internally by the UNIX System.

Areas to be measured include:

- CPU utilization
- Buffer and file access activity
- Terminal device activity
- Disk I/O activity
- System calls
- Process switching
- Swapping activity
- Queue activity
- Inter-Process Communications (IPC) activity.

The utilities consist of **sar, sadc, sag, sadp, timex,** and two shell scripts, **sa1** and **sa2,** for generating daily reports automatically.

Refer to the *AT&T 3B2 Computer User Reference Manual* for UNIX System V manual pages supporting the commands described in this guide.

KERNEL PROFILING

Kernel profiling is a mechanism by which it is possible to sample where the operating system is spending time during operation. It consists of a pseudo-device, */dev/prf,* the associated device driver, and user level commands to control the profiling process and to generate reports. The profiling mechanism samples the program counter on every clock interrupt and increments the counter corresponding to the function shown by that value of the program counter.

The profiling utilities consist of **prfld, prfstat, prfdc, prfsnap,** and **prfpr**. The manual pages for these utilities are included in the *AT&T 3B2 Computer User Reference Manual.*

GUIDE ORGANIZATION

Chapter 2, "REQUIREMENTS," describes the changes that the system administrator must make to certain files. The requirements also include other utilities that must be installed for the Performance Measurement Utilities commands to function properly.

Chapter 3, "COMMAND DESCRIPTIONS," describes the command formats (syntax) for each command in the Performance Measurement Utilities. The descriptions include the purpose of the command, a discussion of the command syntax and options, and examples of using each command.

Chapter 4, "TUNING AND CONFIGURATION," describes the major areas of the 3B2 Computer System that affect system performance. It contains information for tuning the UNIX System for minimum overhead and tuning the disk subsystem for maximum throughput. Also, included are workload analysis, performance tools, and housekeeping techniques for reducing peak load and estimating capacity.

Chapter 2

REQUIREMENTS

PAGE

GENERAL 2-1

REQUIREMENTS 2-2

- How to Restart Activity Counters From Zero 2-2
- How to Produce Records and Important Activities 2-2
- Timex Requirements 2-3
- How to Operate the System Profiler 2-5
- Sag Requirements 2-7

Chapter 2

REQUIREMENTS

GENERAL

After the Performance Measurement Utilities floppy disk has been installed, the person responsible for administrating the system, logged in as **root**, must execute initialization commands and change/add UNIX System files. Performance Measurement Utilities that rely on specific Software Utilities are also identified.

This chapter describes the necessary UNIX System changes that are required for the Performance Measurement Utilities to work properly. Setup procedures are discussed, explained, and elaborated on.

The **sadc, sadp, prfld,** and **prfpr** utilities, described in this Utilities Guide, require access to the symbol table of the UNIX System kernel. So, *root* is the only login ID able to use all the features.

REQUIREMENTS

How to Restart Activity Counters From Zero

The UNIX System kernel contains many counters that are incremented as various system activities occur. To mark the time at which the counters restart from zero, the person administrating the system must create a file *sa* under the directory */etc/rc.d/*. Then, add to file *sa* the following:

```
/usr/lib/sa/sadc /usr/adm/sa/sa`date +%d`
```

Each time the system is booted, this *sa* file will be executed. A file named *saXX*, where *XX* specifies the day of creation, will be created. The *saXX* file contains a special binary record containing system activity information.

How to Produce Records and Important Activities

Most users set up a *cron* file to run system activity data collection programs at specific intervals of time. If you are not familiar with the way *cron* interprets a **crontab** file, refer to the *AT&T 3B2 Computer System Administration Utilities Guide* before continuing.

To produce records (every 20 minutes during working hours and every hour, otherwise) and to report important activities (hourly, during the working day), the person administrating the system must add the following to the */usr/spool/cron/crontabs/root* file:

```
0 * * * 0,6 /usr/lib/sa/sa1
0 8-17 * * 1-5 /usr/lib/sa/sa1 1200 3
0 18-7 * * 1-5 /usr/lib/sa/sa1
5 18 * * 1-5 /usr/lib/sa/sa2 -s 8:00 -e 18:01 -i 3600 -A
```

The first line enters a single record every hour, Monday through Saturday, into */usr/adm/sa/saXX* (where *XX* is the current day). The second line enters a record every 20 minutes between 8:00 AM and 5:00 PM Monday through Friday. The third line enters a single record every hour between 6:00 PM and 7:00 AM, Monday through Friday. The last line will write a report at 6:05 PM, Monday through Friday; this report will be put in */usr/adm/sa/sarXX* (where *XX* is the current date). The **sa2** will use the data collected by **sa1** between the hours of 8:00 AM through 6:01 PM to produce a report with hourly statuses of all activities.

Timex Requirements

The timex command offers three options *-p, -o* and *-s*. If the *Accounting Utilities* are not present and enabled, the *-p* and *-o* options will return the following error message:

- Information from -p and -o options not available because process accounting is not operational.

The *-o* option reports on blocks read or written and the total characters transferred by the command being timed.

The *-p* option lists the process accounting records associated with the command being timed.

The *-o* and *-p* options are valuable for performance tuning during software development. If you do not have the *Accounting Utilities* you may wish to get it, even if it is only for tuning reasons.

Now, there is no *Accounting Utilities* available for the 3B2 Computer. If there is a large enough demand for accounting capabilities, this utilities will be available in the future. If you have a UNIX System source license for the 3B5 Computer, you could port the accounting utilities to the 3B2 Computer for your own use.

How to Operate the System Profiler

The system profiler (prf) is used to initialize the recording mechanism. It will generate a table containing the starting address of each system subroutine as extracted from the UNIX System kernel. The requirements for operating the system profiler (prf) are defined as follows:

- First — The person administrating the system should be in $HOME directory,
  enter: cd<*CR*>

- Second — The profiler must be initialized or loaded,
  enter: */etc/prfld*<*CR*>

- Third — The sampling mechanism must be turned on,
  enter: */etc/prfstat on*<*CR*>

- Fourth — The data must be collected and entered into a file, to be analyzed, every so many minutes and turned off at a specified time,
  enter: */etc/prfdc file ?? ??*<*CR*>

  Note: The *file* can be any file name. The first set of *??* is the number of minutes to run. The second set of *??* is the turn off hour (0 - 24). A snapshot of the data can also be taken by entering: */etc/prfsnap file*<*CR*>

- Fifth — To print the contents of the data, collected by *prfdc* or *prfsnap*,
  enter: */etc/prfpr file*<*CR*>

- Sixth — To turn the sampling mechanism off,
  enter: */etc/prfstat off*<*CR*>

- Seventh — At any time, to see what the status of the sampling mechanism is,
  enter: */etc/prfstat*<*CR*>.

These commands must be executed after every boot. Profiling must be set to *off* when the machine is shutdown; otherwise, the profiler will be working during the shutdown operation. If this happens, the system will hang just after it prints **System is down** on the maintenance console and the 3B2 Computer will remain powered up.

If you want the profiler to begin automatically when you boot the system, you can create a file in */etc/rc.d/* called *prf*. The contents of this file should be as follows:

```
#Load the profiler and enable operation.
/etc/prfld
/etc/prfstat on
```

This will load the profiler during the initialization procedures that occur when you power up your 3B2 Computer. The following will be printed on the console during boot up:

```
profiling enabled

XXX kernel text addresses
```

Note: Where XXX states how many kernel text addresses are in the present UNIX System kernel.

If you automatically start the profiler, make sure you add a file called *prf* in the */etc/shutdown.d* directory to turn the profiler off before shutting down the 3B2 Computer. The contents of *prf* should be as follows:

```
#Make sure the profiler is off before shutdown.
/etc/prfstat off
```

Note: This makes sure the profiler is turned off, avoiding the possibility of hanging the system during the shutdown process.

Sag Requirements

The **sag** command is dependent on the *Graphics Utilities* for display of system data. The **sag** command also requires the following type of known graphic equipment:

- 300 DASI 300
- 300S DASI 300s
- 450 DASI 450
- 5620 DMD 5620
- 4014 Tektronix* 4014
- ver Versatec† D1200A.

For detailed information of graphic equipment, see *tplot* in the *AT&T 3B2 Computer Graphics Utilities Guide*.

---

* Registered Trademark of Tektronix, Inc.

† Registered Trademark of Tektronix, Inc.

Chapter 3

COMMAND DESCRIPTIONS

PAGE

COMMAND SUMMARY 3-1

HOW COMMANDS ARE DESCRIBED 3-4

COMMAND DESCRIPTIONS 3-7

- prfdc — Profiler Data Collector 3-7
- prfld — Profiler Loader 3-9
- prfpr — Profiler Formatter 3-11
- prfsnap — Profiler Snapshot Data Collector 3-13
- prfstat — Profiler Status 3-15
- sadc — System Activity Data Collector 3-17
- sadp — Disk Access Profiler 3-19
- sag — System Activity Graph 3-21
- sar — System Activity Reporter 3-25
- sa1 — System Activity Report Package 3-35
- sa2 — System Activity Report Package 3-37
- timex — Time a Command; Report Process Data and System Activity 3-39

Chapter 3

COMMAND DESCRIPTIONS

COMMAND SUMMARY

The Performance Measurement Utilities provide twelve UNIX System commands. A summary of these commands are provided in Figure 3-1.

COMMAND	DESCRIPTION
prfdc	This command performs the data collection function of the profiler by copying the current value of all the text address counters to a file where the data can be analyzed.
prfld	This command is used to initialize the recording mechanism in the system.

Figure 3-1. Performance Measurement Utilities — Command Summary (Sheet 1 of 3)

COMMAND	DESCRIPTION
prfpr	This command formats the data collected by **prfdc** or **prfsnap**.
prfsnap	This command collects data (like **prfdc**) at the time of invocation only.
prfstat	This command is used to enable, disable, or check the status of the sampling mechanism.
sadc	This command is used to sample, save, and process the system activity data.
sadp	This command reports disk access location and seek distance in tabular or histogram form.

Note: Prfdc, prfld, prfpr, prfsnap, and **prfstat** form a system of programs to facilitate an activity study of the UNIX System.

Figure 3-1. Performance Measurement Utilities — Command Summary (Sheet 2 of 3)

COMMAND	DESCRIPTION
sag	This command graphically displays the system activity data stored in a binary data file by a previous **sar** command.
sar	This command samples cumulative activity counters in the operating system at specified intervals of time, and will save the samples in binary format.
sa1	This shell script, a variant of **sadc**, is used to collect and store data in binary file /urs/adm/sa/sadd_ where dd_ is the current day.
sa2	This shell script, a variant of **sar**, writes a daily report in file /usr/adm/sa/sardd_.
timex	When **timex** and a *command* are executed; the elapsed time, user time, and system time spent in execution of *command*, are reported in seconds.

Figure 3-1. Performance Measurement Utilities — Command Summary (Sheet 3 of 3)

HOW COMMANDS ARE DESCRIBED

A common format is used to describe each of the commands. This format is as follows:

- **General:** The purpose of the command is defined. Any uncommon or special information about the command is also provided.
- **Command Format:** The basic command line format (syntax) is defined and the various arguments and options discussed.
- **Sample Command Use:** Example command line entries and system responses are provided to show you how to use the command.

In the command format discussions, the following symbology and conventions are used to define the command syntax.

- The basic command is shown in bold type. For example: **command** is in bold type.
- Arguments that you must supply to the command are shown in a special type. For example: **command** *argument*
- Command options and arguments that do not have to be supplied are enclosed in brackets ([ ]). For example: **command** [*optional arguments*]
- The pipe symbol | is used to separate arguments when one of several forms of an argument can be used for a given argument field. The pipe symbol can be thought of as an exclusive OR function in this context. For example: **command** [*argument1* | *argument2*]

In the sample command discussions, the lines that you input are ended with a carriage return. This is shown by using <*CR*> at the end of the lines.

Refer to the *AT&T 3B2 Computer User Reference Manual* for UNIX System V manual pages supporting the commands described in this guide.

COMMAND DESCRIPTIONS

prfdc — Profiler Data Collector

General

The command **prfdc** performs the data collection function of the profiler by copying the current value of all the text address counters to a file where the data can be analyzed.

Command Format

The general format for the **prfdc** command is as follows:

/etc/prfdc *file [period [off hour]]*

Prfdc will store the counters into a *file* every *period* minutes and will turn off at *off hour*. Valid values for *off hour* are 0-24.

Sample Command Use

To copy the current value of all the text address counters into a file called *temp* every five minutes; and to turn off at four o'clock in the afternoon, enter the following:

```
#/etc/prfdc temp 5 16<CR>
#
```

prfld — Profiler Loader

General

The command **prfld** is used to initialize, or load, the recording mechanism in the system.

Command Format

The general format for the **prfld** command is as follows:

/etc/prfld *[namelist]*

Prfld generates a table, in memory, containing the starting address of each subroutine as extracted from *namelist*. The default of *namelist* is */unix*.

Sample Command Use

To generate the table containing the starting addresses of each subroutine, enter the following:

```
#/etc/prfld<CR>
#
```

prfpr — Profiler Formatter

General

The command **prfpr** formats the data that was collected by **prfdc** or **prfsnap**. Each text address is converted to the nearest text symbol (function) and the percentage of time used by that function is printed if the activity percentage is greater than the cutoff number specified by the user.

Command Format

The general format for the **prfpr** command is as follows:

/etc/prfpr *file [cutoff [namelist]]*

Prfpr will format the contents of *file* (created by **prfdc** or **prfsnap**). Each text address is converted and printed if the percent activity for that range is greater than that specified by *cutoff*. The range of *cutoff* would be 0% to 99%, where 0% would print all contents. The default of *cutoff* is 1%. The default of *namelist* is */unix*.

Sample Command Use

To print the data collected by **prfdc** or **prfsnap**, enter the following:

```
#/etc/prfpr temp 0<CR>

07/19/85 07:06
07/19/85 07:17

idle    92.3
fbzero  0.2
fbcopy  0.9
copyout 0.5
fubyte  0.1
subyte  0.1
        0.2
readi   0.1
bmap    0.2
systrap 0.1
exece   0.2
rdwr    0.1
getblk  0.1
bdflush 0.1
getc    0.1
putc    0.1
intOB   0.1
idsetup 0.1
user    2.1
#
```

Note: These are the function codes in the kernel.
For detailed information on function codes refer to
the AT&T 3B2 Computer Programmer Reference Manual.
See sections on Subroutines, System Calls, and/or Library.

prfsnap — Profiler Snapshot Data Collector

General

The command **prfsnap** performs the data collection function of the profiler by copying the current value of all the text address counters to a file where the data can be analyzed.

Command Format

The general format for the **prfsnap** command is as follows:

/etc/prfsnap *file*

Prfdc will store the counters into a *file* every specified number of minutes and will turn off at a specified hour. **Prfsnap** takes a "snapshot" of the system, collecting data at the time it is called, and appends the counter values to the *file*.

Sample Command Use

To copy the current value of all the text address counters into a file called *temp*, at the time the command is entered, enter the following:

```
#/etc/prfsnap temp<CR>
#
```

prfstat — Profiler Status

General

The command **prfstat** is used to enable or disable the sampling mechanism initialized by **prfld**. Profiler overhead is less than 1% as calculated for 500 text addresses.

Command Format

The general format for the **prfstat** commands are as follows:

/etc/prfstat *on*
/etc/prfstat *off*

Sample Command Use

To find out the status of the profiler, enter the following:

```
#/etc/prfstat<CR>
profiling (enabled or disabled)
#
```

To enable or turn on the recording mechanism, enter the following:

```
#/etc/prfstat on<CR>
profiling enabled
#
```

Sample Command Use (Continue)

To disable or turn off the recording mechanism, enter:

```
#/etc/prfstat off<CR>
profiling disabled
#
```

sadc — System Activity Data Collector

General

System activity data can be accessed (automatically and on a routine basis) at the special request of the user, assuming the person responsible for administrating the system has created the file shown on the next page under **Sample Command Use**. The operating system contains a number of counters that are incremented as various system actions occur. These counters are:

- CPU use
- Buffer usage
- Disk I/O activity
- Terminal device activity
- Switching and system-call
- File-access
- Queue activity
- Counters for inter-process communications.

Sadc is used to sample, save, and process data from the above activities.

Command Format

The general format for the **sadc** command is as follows:

/usr/lib/sa/sadc *[t n] [ofile]*

Sadc is the data collector, it samples the system data every *t* seconds for so many *n* times and writes in binary format to an *ofile* or to the standard output. If *t* or *n* are omitted, a single record is written.

Sadc is used at system boot time to mark the time at which the counters restart from zero.

Sample Command Use

To write the special record to the daily data file, the person administrating the system must first create a new file *sa* under directory */etc/rc.d*. Second, this person must add the following to the *sa* file:

```
/usr/lib/sa/sadc /usr/adm/sa/sa`date +%d`
```

Note: The person administrating the system must have root login.

sadp — Disk Access Profiler

General

Sadp reports disk access location and seek distance in tabular or histogram form. It will sample the disk activity every second for so many seconds and is repeated as many times as specified by the user. Cylinder usage and seek distance are recorded in units of eight cylinders.

Command Format

The general format for the **sadp** command is as follows:

sadp *[-th] [-d device[-drive]] s [n]*

The *-t* flag causes the data to be reported in tabular form. The *-h* flag causes the data to be produced in histogram form on a printer. If *-t* or *-h* is not specified, the report will be in tabular form. The *-d* option may be omitted, since only one or two *devices* are present.

Sadp samples disk activity once every second during an interval of *s* seconds. This is done repeatedly if *n* (a number) is specified, or once if not specified.

Sample Command Use

To generate a tabular report, describing cylinder usage and seek distance of disk drive during a five minute interval, enter the following.

```
#sadp 300<CR>

Fri July 19 12:45:28 1985
unix unix 2.0.3 2 3B2

CYLINDER ACCESS PROFILE
hdsk-0:
Cylinders        Transfers
 64 -  71        4
120 - 127        1
136 - 199        2
208 - 215        4
224 - 231        1
232 - 239        2
272 - 279        3
344 - 351        5
424 - 431        1
432 - 439        1
440 - 479        4
624 - 631        2
664 - 671        1
672 - 679        5

Sampled I/O = 36, Actual I/O = 683
Percentage of I/O sampled = 5.27

SEEK DISTANCE PROFILE
hdsk-0:
Seek Distance    Seeks
        0        2
  1 -   8        2
 25 -  32        1
 57 -  64        1
137 - 144        2
153 - 160        1
201 - 208        1
217 - 224        1
241 - 248        1
273 - 280        1
281 - 288        1
425 - 432        1
609 - 616        1
Total Seeks = 16
#
```

sag — System Activity Graph

General

The command **sag**, graphically displays the system activity data stored in a binary data file created by a previous **sar** run. Any of the **sar** data items may be plotted singly, or in combination; as cross plots, or versus time. The **sag** command is dependent on the *Graphics Utilities* for display of system data, and one of the following graphic devices:

- 300 DASI 300
- 300S DASI 300s
- 450 DASI 450
- 5620 DMD 5620
- 4014 Tektronix 4014
- ver Versatec D1200A.

For detailed information, see *tplot* in the *AT&T 3B2 Computer Graphics Utilities Guide*.

Command Format

The format for the **sag** command is as follows:

sag *[options]*

The options for **sag** are:

-s *time*	Select data later than -s *time* in the form of hh[:mm]. The default of *time* is 08:00.
-e *time*	Select data up to a -e *time*. Default is 18:00.
-i *sec*	Select data at intervals as close as possible to -i *sec* seconds.
-f *file*	Use -f *file* as the data source for **sar**. Default is the current daily data file (*/usr/adm/sa/sadd*).
-T *term*	Produce output suitable for terminal *term*. See *tplot* for known terminals. The default for *term* is $TERM.
-x *spec*	The x axis specification *spec* is in the form: name [op name]...[lo hi].
-y *spec*	The y axis specification *spec* is in the form: name [op name]...[lo hi].

Name is either a string that will match a column header in the **sar** report, or a number value.

Sample Command Use

To see todays CPU activity, enter the following:

```
$sag -i0<CR>
```

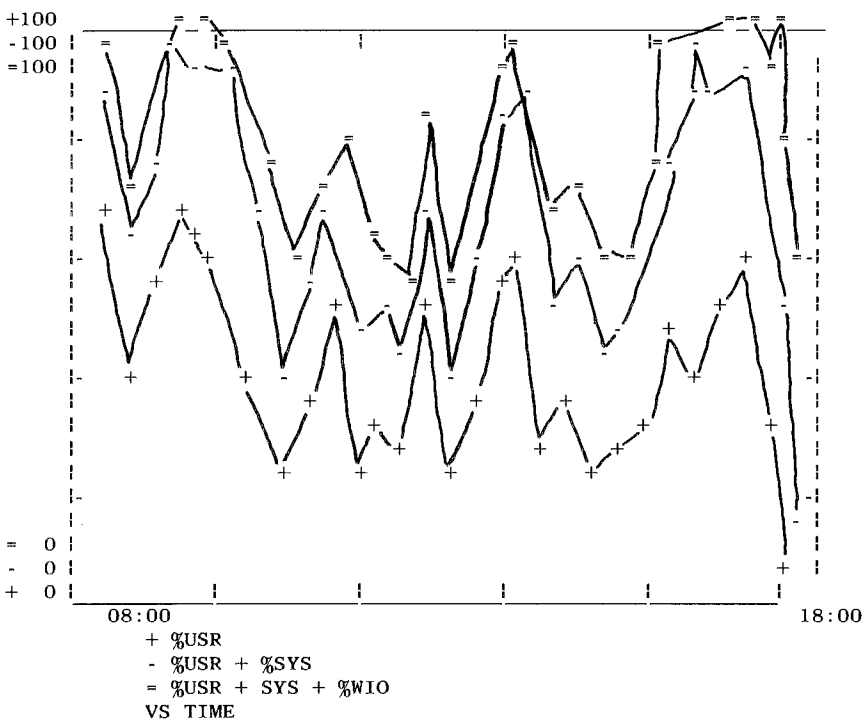


Figure 3-2. Example of sag Output

```
$
```

Note: Must have proper type of terminal.

To see activity of all users, over a 15 minute period, enter the following commands:

```
$TS='date +%H:%M'<CR>          (TS = start time for test)
$sar -o tempfile 60 15<CR>

unix unix 2.0.3 2 3B2    07/19/85
13:52:27        %usr    %sys    %wio    %idle
13:53:27           1       0       1       98
13:54:27           1       1       0       98
13:56:27           2       3       2       93

                (All data NOT shown)

14:06:27           1       4       0       95
14:07:27           1       2       1       97
Average            1       2       1       97
$

(Now enter this)

$TE='date +%H:%M'<CR>          (TE = end time of test)
$sag -f tempfile -s $TS -e $TE -i0 -y "%usr"<CR>

              (A Graphic Display with the above
            statistics will appear here)

$
```

Note: First, tempfile was placed in your directory containing this data. Second, the screen will show one time-frame every minute for 15 minutes. Third, the %usr will be displayed, graphically, on your terminal. You must have the proper terminal.

sar — System Activity Reporter

General

The **sar** command samples cumulative activity counters in the operating system. The reports can be stored in a *file* or *files*. The reports can be taken at a specified time, for a specified number of intervals, and for a specified number of seconds.

Command Format

There are two general formats for the **sar** command:

sar *[-ubdycwaqvmA] [-o file] t [n]*
sar *[-ubdycwaqvmA] [-s time] [-e time] [-i sec] [-f file]*

With the first format, **sar** samples cumulative activity counters, in the operating system at a specified number of *t* seconds for a specified number of *n* intervals. If the *-o* option is specified, it saves the samples in a *file* in binary format.
The options (*[-ubdycwaqvmA]*) specify what counters to sample.

With the second format, without specifying a sample interval, **sar** will extract data from a *file* (either the one specified by the *-f* option or the standard system activity daily data file). The starting and ending times of the report can be specified by the *-s* and/or *-e time* arguments. The *-i* option selects records at *sec* second intervals.

Other options of **sar**, to specify what counters to be sampled, are:

-u Report CPU use:

%usr = Portion of time running in user mode

%sys = Portion of time running in system mode

%wio = Waiting for block I/O

%idle = System idle.

-b Report buffer activity:

bread/s = Number of blocks read from disk per second

lread/s = Number of user reads

%rcache = Read % of cache hit ratios

bwrit/s = Number of blocks written to disk per second

lwrit/s = Number of user writes

%wcache = Write % of cache hit ratios

pread/s = Read transfers via raw (physical) device mechanism

pwrit/s = Write transfers via raw (physical) device mechanism.

-d Report activity for each block device, such as, hard disk drive:

device = Which device is being reported

%busy = Portion of time device was busy serving a transfer request

avque = Average number of requests outstanding during time device was busy

r+w/s = Number of data transfers from or to device

blks/s = Number of bytes transferred in 512-byte units (blocks)

avwait = Average time in milliseconds that transfer requests wait on queue

avserv = Average time to be serviced.

-y Report terminal device activity:

rawch/s = Input character rate -- rate at which characters arrive

canch/s = Input character rate processed by canon

outch/s = Output character rate

rcvin/s = Receive interrupt rate -- as a result of character arrivals

xmtin/s = Transmit interrupt rate

mdmin/s = Modem interrupt rate.

-c Report system calls:

scall/s = System calls of all types

sread/s = Read system calls

swrit/s = Write system calls

fork/s = Fork system calls

exec/s = Execute system calls

rchar/s = Characters transferred by read system calls

wchar/s = Characters transferred by write system calls.

-w Report system swapping and switching activity:

swpin/s = Number of programs moved from auxiliary storage into main memory

bswin/s = Number of 512-byte blocks swapped in

swpot/s = Number of programs moved from main memory into auxiliary storage

bswot/s = Number of 512-byte blocks swapped out

pswch/s = Number of process switches.

-a Reports on the use of file access system routines:

iget/s = Inode gets per second

namei/s = Inode names returned

dirbk/s = Inode conversions.

-m Report message and semaphore activities:

msg/s = Messages per second

sema/s = Semaphores per second.

-v Report status of text, process, i-node, and file tables:

text-sz = Text size for each table, evaluated once at sampling point

ov = Overflows occurring between sampling points

proc-sz = Process size for each table, evaluated once at sampling point

ov = Overflows occurring between sampling points

inod-sz = Inode size for each table, evaluated once at sampling point

ov = Overflows occurring between sampling points

file-sz = File size for each table, evaluated once at sampling point

ov = Overflows occurring between sampling points

lock-sz = Lock size for each table, evaluated once at sampling point

fhdr-sz = File Header size for each table, evaluated once at sampling point.

-q Report average queue length while occupied, and % of time occupied:

runq-sz = Run queue of processes in memory and runnable

%runocc = The percent of time run queue occupied

swpq-sz = Swap queue of processes swapped out, but ready to run

%swpocc = The percent of time swap queue occupied.

-A Report all data. Equivalent to *-udqbwcayvm*.

Sample Command Use

The following example shows how to see todays CPU activity up to the present time:

```
$sar<CR>

unix unix 2.0.3 2 3B2    07/19/85

00:00:03        %usr    %sys    %wio    %idle
01:00:04           0       1       1       98
02:00:02           1       1       1       98
03:00:02           0       1       1       98
04:00:02           0       1       1       98
05:00:01           1       2       1       97
06:00:01           1       4       0       95
07:00:02           2       3       2       93
08:00:01           2       7       1       90
Average            1       2       1       96

Note: This will only show on the standard output.
$
```

To sample the CPU every 60 seconds; for 10 intervals, and to put the report in a file called *tempfile*, enter the following:

```
$sar -o tempfile 60 10<CR>

unix unix 2.0.3 2 3B2    07/19/85

17:26:47    %usr    %sys    %wio   %idle
17:27:47       1       0       0      99
17:28:47       1       0       0      99
17:29:47       2      11       3      84
17:30:47       2       1       0      97
17:31:47       1       2       0      97
17:32:47       2       1       0      97
17:33:47       3       9       0      88
17:34:47      17      63       0      20
17:35:47      20      80       0       0
17:36:47      20      79       0       0

Average        7      25       0      68
$
```

Note: This will show on the standard output; plus, a file tempfile will be placed in your directory containing this data.

The next example shows how to create a file, for later review, of the disk activity.

```
$sar -d -o tmp<CR>

unix unix 2.0.3 2 3B2    07/19/85
00:00:03   device   %busy   avque   r+w/s   blks/s   avwait   avserv

01:00:04 hdsk-0         1     2.0       0        1     32.7     32.7

02:00:02 hdsk-0         1     1.7       0        1     24.4     33.0

03:00:02 hdsk-0         1     1.7       0        1     22.0     32.1

04:00:02 hdsk-0         1     1.9       0        1     30.4     32.4

05:00:01 hdsk-0         1     1.7       0        1     22.5     30.3

06:00:01 hdsk-0         1     1.6       0        1     19.5     33.0

07:00:02 hdsk-0         3     2.0       1        2     35.9     34.6

08:00:01 hdsk-0         2     1.6       0        1     26.0     40.3

08:20:01 hdsk-0         2     1.7       1        1     22.8     31.8

08:40:01 hdsk-0         3     2.7       1        2     54.5     32.7

09:00:02 hdsk-0         1     1.6       0        1     20.2     32.1

Average  hdsk-0         2     1.9       0        1     29.8     33.6
$
```

Note: This will show on the standard output; plus, a file tmp will be created, saving the data in binary format.

sa1 — System Activity Report Package

General

System activity data can be accessed, automatically and on a routine basis, at the special request of the user, assuming the person administrating the system has added to the file shown (see next page) under the first **Sample Command Use**.

The operating system contains a number of counters that are incremented as various system actions occur. These counters are:

- CPU use
- Buffer usage
- Disk I/O activity
- Terminal device activity
- Switching and system-call
- File-access
- Queue activity
- Counters for inter-process communications.

Sa1, like **sa2**, is used to sample, save, and process this data. **Sa1** writes to file */usr/adm/sa/sadd*, where *sa2* writes to file */usr/adm/sa/sardd*.

Note: *Sadd* file is for the minute, *sardd* file is for the hour.

Command Format

The general format for the **sa1** command is as follows:

/usr/lib/sa/sa1 *[t n]*

The shell script **sa1**, is like that of **sadc**, it is used to collect and store data in binary file */usr/adm/sa/sadd*. The *dd* in sa*dd* is the current day. The arguments *t* and *n* cause records to be written *n* times at an interval of *t* seconds, or once if omitted.

Sample Command Use

To produce records, every 20 minutes during working hours and every hour otherwise, the person administrating the system must add the following to the file *root* in directory */usr/spool/cron/crontabs/*:

```
0 * * * 0,6 /usr/lib/sa/sa1
0 8-17 * * 1-5 /usr/lib/sa/sa1 1200 3
0 18-7 * * 1-5 /usr/lib/sa/sa1
```

Note: The person administrating the system must be logged in as **root**.

To produce a record, in binary file /usr/adm/sa/sadd, every five minutes for three times, enter the following:

```
$/usr/lib/sa/sa1 300 3<CR>
$
```

Note: To see the results of this command, on standard output, see the sar command.

sa2 — System Activity Report Package

General

System activity data can be accessed, automatically and on a routine basis, at the special request of the user, assuming the person administrating the system has added to the file shown (see next page) under the first *Sample Command Use*.

The operating system contains a number of counters that are incremented as various system actions occur. These counters are:

- CPU use
- Buffer usage
- Disk I/O activity
- Terminal device activity
- Switching and system-call
- File-access
- Queue activity
- Counters for inter-process communications.

Sa2, like **sa1**, is used to sample, save, and process this data. **Sa2** writes to file */usr/adm/sa/sardd*, where *sa1* writes to file */usr/adm/sa/sadd*.

Note: *Sardd* file is for the hour, *sadd* file is for the minute.

Command Format

The general format for the **sa2** command is as follows:

/usr/lib/sa/sa2 *[-ubdycwaqvmA] [-s time] [-e time] [-i sec]*

Sa2 will write a report using options *[-ubdycwaqvmA]* (explained in **sar**) starting at *-s* time, ending at *-e* time, and as close to *-i* seconds as possible, to the daily report file **/usr/adm/sa/sardd**.

Sample Command Use

To report important activities hourly, during the working day, the person administrating the system must add the following to the file *root* in directory */usr/spool/cron/crontabs/*:

```
5 18 * * 1-5 /usr/lib/sa/sa2 -s 8:00 -e 18:01 -i 3600 -A
```

Note: The person administrating the system must be logged in as **root**.

To write a report showing terminal device activity, from 1:00 PM to 4:00 PM on the hour, enter the following:

```
$/usr/lib/sa/sa2 -y -s13:00 -e16:00 -i3600<CR>
$
```

Note: Nothing will be output on the screen, but the report will be added to the daily report file.

timex — Time a Command; Report Process Data and System Activity

General

Process accounting data for a given *command* (such as: **spell**, **pwd**, **ls -l**, **sort**, **sar**,) and all its children, can be listed or summarized by using **timex**. The total system activity, during the execution interval of a *command*, can also be reported by using **timex**.

The given *command* is executed; the elapsed (real) time, user time, and system time spent in execution are reported in seconds.

The output of **timex** is written on standard output.

Command Format

The general format for the **timex** command is as follows:

timex *[-p -o -s] command*

The options of **timex** are:

-p Lists process accounting records for *command* and all of its children. Suboptions *f*, *h*, *k*, *m*, *r*, and *t*, change the data items reported. The suboptions are explained under **acctcom** of the UNIX System Users Manual. The number of blocks read or written and the number of characters transferred are always reported with the *-p* option.

-o Reports the total number of blocks read or written and total characters transferred by *command* and all its children.

Note: Options *-o* and *-p* will only work if *System Accounting* is operational on the system. Now, *System Accounting* is not available for the 3B2 Computer.

-s Reports total system activity, not just what was called for in *command*, but, everything that occurred during the execution interval of *command*. All the data items listed in **sar** are reported.

Sample Command Use

To find out how many seconds it takes to display a long list of the contents of your current directory, enter the following:

```
$ timex ls -l<CR>
total 160
-rw-r--r--   1 markm    PR3B        3754 July 19 13:56 chapter1
-rw-r--r--   1 markm    PR3B        8037 July 19 14:00 chapter2
-rw-r--r--   1 markm    PR3B       33941 July 19 14:17 chapter3
-rw-r--r--   1 markm    PR3B       36891 July 19 14:35 chapter4
-rw-r--r--   1 markm    PR3B         830 July 13 14:16 trademarks

real                0.80
user                0.25
sys                 0.20
$
```

To list the process accounting record of sorting a file *tmp*, and redirecting the output to a file *tmp1*, first create a file **tmp** and add words or numbers in a list. Then, enter the following:

```
$ timex sort tmp > tmp1<CR>
real     0.52
user     0.04
sys      0.15
$
```

Note: Only the time of execution will be listed, and a file tmp1 will be created containing the sorted data from tmp.

Chapter 4

TUNING AND CONFIGURATION

PAGE

TUNING A 3B2 COMPUTER SYSTEM 4-1

SETTING KERNEL CONFIGURATION PARAMETERS 4-2

IMPROVING DISK USEAGE 4-3

- Selecting Tunable Parameters 4-3
- Setting Text-bit (Sticky-bits) 4-4
- File System Organization 4-5

PERFORMANCE TOOLS 4-7

- Introduction 4-7
- Internal Activity 4-7

LOAD MANIPULATION AND HOUSEKEEPING 4-30

- Introduction 4-30
- ps 4-30
- User $PATH Variables 4-31

Chapter 4

TUNING AND CONFIGURATION

TUNING A 3B2 COMPUTER SYSTEM

This section describes the major areas of the AT&T 3B2 Computer System that affect system performance. It contains information for tuning the kernel for minimum overhead and tuning the disk subsystem for maximum throughput. Also included are workload analysis, performance tools, and housekeeping techniques for reducing peak load and estimating capacity. A description of the tunable parameters may be found in the *AT&T 3B2 Computer System Administration Utilities Guide*.

SETTING KERNEL CONFIGURATION PARAMETERS

Most kernel parameters have minimum values that are required for normal kernel operation. Increasing the values of heavily loaded parameters can increase system performance significantly.

With the tools provided in this Utilities Guide, you can measure system load and determine what parameters to increase and what to lower. Lowering the values of parameters can save valuable main storage space (smaller kernel size), providing a larger area for user programs and reduced swapping activity.

The command **/etc/sysdef** will output the configuration of your system, including the values set for the tunable parameters. Refer to the *AT&T 3B2 Computer System Administration Utilities Guide* to locate the file(s) that contain the values for the tunable parameters and for information on how to change values and rebuild your UNIX System.

IMPROVING DISK USEAGE

Disk input/output may cause a bottleneck in system performance. There are three steps in tuning the disk subsystem for better usage.

- Select the proper tunable parameters.
- Setting Text-bit (Sticky-bits).
- Organizing the file systems to minimize disk activity.

Selecting Tunable Parameters

The *3B2 Computer System Administration Utilities Guide* has a table for recommended beginning values for tunable parameters located in the System Reconfiguration section. You should use the values given in the table as a starter. When you become familiar with the tools provided in this utilities, you may want to change parameters that will increase system performance for your specific application.

Setting Text-bit (Sticky-bits)

Setting the text-bit makes the I/O of a select group of commands more efficient. When the text-bit is set on a command, a copy of the text segment of the command will be loaded into a contiguous area on the swap device. Whenever the program is executed, the text segment is swapped in with one I/O operation. This reduces disk and CPU load, resulting in faster command start-up and yielding improved response time. (When the text-bit is not set, the program is copied into memory from the file system one block at a time).

System resources (one i-node and text table entry plus swap space) for each text-bit program residing on the swap device are used. It may not be necessary to set the text-bit on programs that are being run most of the time (**sh**, **getty**... etc.) since these programs would already have their text segments in memory.

After setting text-bits, observe the number of text and i-node entries with **sar -v**. If the tables are nearly full, it may be necessary to reconfigure the system with more i-node and text table slots.

The commands on which the text-bit is set must be pure executable (text segment shared and write protected). Use the *file* command to determine if the text-bit can be set on a command.

The text-bit should only be used for commands that are used heavily. Usually, there are only 4 to 6 commands with the text-bit set. This makes sure that swap space is not taken up solely by text-bit commands; too many programs with the text-bit set could cause the system to spend all its time swapping (thrashing) programs in and out. The **crash** command (smap) is useful for displaying the number of free segments on the swap device. See the *AT&T 3B2 Computer Crash Analysis Guide* for more information on **crash** abilities.

File System Organization

This section describes several actions that can be taken to reduce the overhead of file access. As file systems are used, they tend to become disorganized and I/O becomes less efficient. This disorganization yields poor ordering of blocks with files and poor directory structure.

Organization of File System Free List

The 3B2 Computer file systems are set up to allocate free blocks in a way that allows the files to be read or written with efficiency. A free list array is created when a file system is created with **/etc/mkfs**. The free list is set up with the optimal rotational gap of 7, as specified by **mkfs** options. The difference between successive block numbers in the free list is the rotational gap. For example, a file created on a system with a rotational gap of 7 may consist of blocks 500, 507, and 514. When the file is read, I/O requests are sent to the disk drive to read blocks 500, 507, and 514. As soon as the drive finishes reading block 500 and has started to process the second request, block 507 will be moving over the read/write head just as the drive is ready to read that request. This makes for an efficient I/O operation.

However, as you start changing files (changing size or removing), the efficiency starts to decrease. When several files are being created at once, they will be contending for blocks from the free list. Some of the blocks allocated to the files will be out of sequence. As you can see, the free list becomes disorganized.

Directory Organization

Free space in directories can decrease I/O performance. When a file is removed from a directory, the i-node number is nulled out. This leaves an unused slot of 8-bytes. *Directories retain the largest size they have ever achieved.* This may result in many empty file slots. If you have a directory with 100 files in it and you remove the first 99 files, an **ls** done on this directory will require four I/O operations [three for the empty directory blocks (32 file slots = one directory block) and one for the block with the file entry].

Restoring Good File System Organization

There is no "automatic" way to solve these problems; however, you can use manual intervention by reconfiguring the system. Running **fsck -s** will reorganize the free list.

Directories larger than 320 files (5120 bytes) are inefficient because of file system indirection. Use the following command to find directories with more than 320 entries.

Enter:

```
$find / -type d -size +10 -print<CR>
```

If you find directories of this size, consider breaking them up into smaller directories.

PERFORMANCE TOOLS

Introduction

The categories of performance tools for the UNIX System internal activities described in this section are:

sar	Samples cumulative activity counters internal to the UNIX System and provides reports on various system-wide activities.
sag	Graphically displays the information collected by *sar*.
sadp	Produces profiles of disk access location and seek distance.
timex	Reports both system-wide and per-process activity during the execution of a command or program.

Examples for these tools are provided in the following sections. These examples are from a system with 1-megabyte of main memory and a 30-megabyte integral hard disk. Command outputs are typical values received while benchmarking a specific function of the UNIX System. Values you receive may be different from values in the examples, depending on your application or benchmark. While tuning your systems, it is recommended to use a benchmark or have the system under normal load for your application. This will allow you to tune directly toward your specific application.

Internal Activity

Internal activity is measured by many counters contained in the UNIX System kernel. Each time an operation is performed, the counter monitoring its function is incremented. The functions monitored by **sar** are discussed in the following paragraphs.

sar

In this chapter, **sar** options are described with an analysis of sample outputs of the options. **Sar** can be used as an active process or be used to extract system activity data from the data collected by *sa1* and *sa2*. These data files are created by the entries made in the *crontab* files. Refer to Chapter 2, REQUIREMENTS for an explanation of these entries. Refer to Chapter 3 for a detailed command description of **sar**.

The following examples are listed in alphabetical order according to the available options of the **sar** command.

sar -a

The **sar -a** option reports the use of file access operations. The UNIX System routines reported are as follows:

iget/s Number of files located by its i-node entry per second.

namei/s Number of file system path searches per second. Namei calls iget, so iget is always larger than namei.

dirbk/s Number of directory block reads issued per second.

An example of **sar -a** output follows:

```
unix unix 2.0.3 2 3B2    07/19/85

12:41:40  iget/s namei/s dirbk/s
12:42:10       4       1       3
12:42:43       2       1       1
12:43:14       5       2       3

Average        4       1       3
```

The larger the values reported, the more time the UNIX System kernel is spending in accessing user files. If these numbers raise above 10, you may want to clean the file system. This can be done, but is time consuming when there is only one hard disk and limited free space available. You can use *cpio* to copy a directory structure to a temporary location, then remove the original structure, then copy the temporary structure back to the original location. By doing this, you remove the unused but allocated i-node entries. Refer to the section on Directory Organization for an explanation. This will remove all the nulled out entries from the directory structure.

sar -b

The **-b** option reports the following buffer activity.

bread/s	Average number of physical blocks read between the system buffers and the disk drive (or other block devices) per second.
lread/s	Average number of logical blocks read from system buffers per second.
%rcache	Fraction of logical reads found in buffer cache.
%wcache	Fraction of logical writes found in buffer cache.
bwrit/s	Average number of physical blocks written between the system buffers and the disk drive (or other block devices) per second.
lwrit/s	Average number of logical blocks written to system buffers per second.
pread/s	Average number of physical read requests per second.
pwrit/s	Average number of physical write requests per second.

The entries that you should be most interested in are **%rcache** and **%wcache** (cache hit ratios). If **%rcache** falls below 90, or **%wcache** falls below 65, it may be possible to improve performance by increasing the number of buffers.

The ratio of physical (block) I/O to logical (character) I/O is a common measure of the effectiveness of the system buffering.

An example of **sar -b** output follows:

```
unix unix 2.0.3 2 3B2     07/19/85

16:32:57 bread/s lread/s %rcache bwrit/s lwrit/s %wcache pread/s pwrit/s
16:33:07      16      241      93       1      16      91       0       0
16:33:17      18      206      91       2      16      87       0       0
16:33:27      28      102      76       3       7      64       0       0

Average       21      182      90       2      13      84       0       0
```

This example shows that the buffers are not causing any bottlenecks, because all data is within acceptable limits.

sar -c

The **-c** option reports system calls in the following categories:

scall/s	All types of system calls per second, generally about 80 per second on a busy 2 to 6 user system.
sread/s	Read system calls per second.
swrit/s	Write system calls per second.
fork/s	Fork system calls per second, about 1 per second on a 2 to 6 user system. This number will increase if running shell scripts.
exec/s	Exec system calls per second (If (**exec/s**) / (**fork/s**) is greater than 3, inefficient $PATHs are being used.)
rchar/s	Characters transferred by read system calls per second.
wchar/s	Characters transferred by write system calls per second.

Typically, reads plus writes account for about half of the total system calls; although this varies greatly with the activities that are being performed by the system.

An example of **sar -c** output follows:

```
unix unix 2.0.3 2 3B2    07/19/85

18:33:04 scall/s sread/s swrit/s fork/s exec/s rchar/s wchar/s
18:33:35      81      12      61   0.03   0.03   33506   31052
18:34:05      77       9      58   0.07   0.07   29704   28919
18:34:35      82      14      40   0.17   0.17   24095   19632

Average       80      12      53   0.09   0.09   29146   26604
```

This example shows that operating system call activity varies with each type of program(s) running.

sar -d

The **sar -d** option reports the activity of block devices.

device	Name of the physical entity that **sar** is monitoring.
%busy	Percent of time the device was servicing a transfer request.
avque	The average number of requests outstanding during the period of time.
r+w/s	Number of read and write transfers to the device per second.
blks/s	Number of 512 byte blocks transferred to the device per second.
avwait	Average time in milli-seconds that transfer requests wait in the queue.
avserv	Average time in milli-seconds for a transfer request to be completed by the device (for disks this includes seek, rotational latency and data transfer times).

An example of **sar -d** is as follows:

```
unix unix 2.0.3 2 3B2    07/19/85

13:46:28  device %busy avque r+w/s blks/s  avwait  avserv

13:46:58 hdsk-0      6   1.6     3      5    13.8    23.7
         fdsk-0     93   2.1     2      4   467.8   444.0

13:47:28 hdsk-0     13   1.3     4      8    10.8    32.3
         fdsk-0    100   3.1     2      5   857.4   404.1

13:47:58 hdsk-0     17    .7     2     41      .6    48.1
         fdsk-0    100   4.4     2      6  1451.9   406.5

Average  hdsk-0     12   1.2     3     18     8.4    34.7
         fdsk-0     98   3.2     2      5   925.7   418.2
```

The above example was taken while transferring data from hard disk (hdsk-0) to floppy disk (fdsk-0). Looking at the results tells us why hard disk technology has improved the performance of supermicros.

sar -m

The **sar -m** option reports on inter-process communication activities. Now, only message and semaphore calls are reported. Shared memory will be a future enhancement. Reported values are defined as follows:

msg/s Number of message operations (sends and receives) per second.

sem/s Number of semaphore operations per second.

An example of **sar -m** output follows:

```
unix unix 2.0.3 2 3B2    07/19/85

15:16:58   msg/s  sema/s
15:17:32  156.38  129.47
15:18:02  202.59  139.58
15:18:32  193.66  145.86

Average   182.76  137.86
```

This was recorded during an inter-process communication benchmark. You will probably never see values at this level. Unless you are running application messages, these figures will always be zero (0.00). You can refer to the *AT&T 3B2 Computer Inter-Process Communication Utilities Guide* for more information.

sar -q

The **sar -q** option reports the average queue length while the queue is occupied and percent of time occupied.

runq-sz Run queue of processes in memory, typically, this should be less than 2.

%runocc The percentage of processes in memory that are runnable, the larger this value is the better.

swpq-sz Swap queue of processes swapped out, the smaller this number is the better.

%swpocc The percentage of swapped out processes that are ready to run, the smaller this value is the better.

An example of **sar -q** follows:

```
unix unix 2.0.3 2 3B2    07/19/85

11:00:56 runq-sz %runocc swpq-sz %swpocc
11:01:07     1.7      98     1.5      36
11:01:17     1.0      63     1.0      31
11:01:27     1.0      58     1.0      49

Average      1.3      74     1.2      39
```

In this example, the processor (**%runocc**) varies between 63 and 98 percent, while the fraction of time the swap queue is not empty (**%swpocc**) is 31 to 49 percent. This means that memory is not causing a major bottleneck in the system throughput, but more memory would help reduce the swapping. You will not normally see output like this, a disk intensive benchmark was running when this was recorded.

If **%runocc** is greater than 90 and **runq-sz** is greater than 2, the CPU is heavily loaded and response is degraded. If **%swapocc** is greater than 20, more memory would help reduce swapping activity.

sar -u

The CPU usage is listed by **sar -u** (default). At any given moment the processor will be either busy or idle. When busy, the processor will be in either user or system mode. When idle, the processor will either be waiting for input/output completion or has no work to do. The **-u** option of **sar** lists the percent of time that the processor is in system mode (**%sys**), user mode (**%user**), waiting for input/output completion (**%wio**), and idle time (**%idle**).

In typical timesharing use, **%sys** and **%usr** are about the same value. In special applications, either of these may be larger than the other without anything being abnormal. If you can keep **%sys** below 25 percent average, you have obtained perfect operating system time sharing.

The following are clues to CPU usage, values are averages over a period of time:

- **%usr** —— If greater than 50, users are getting excellent service.
- **%sys** —— If greater than 60, kernel needs to be tuned.
- **%wio** —— If greater than 30, disk bottleneck.
- **%idle** —— If greater than 10, system is not being fully used.

sar -v

The **-v** option reports the status of text, process, i-node, file, shared memory record, and shared memory file tables. From this report you know when the system tables need to be modified.

text-sz	Number of text table entries now being-used/allocated in the kernel.
proc-sz	Number of process table entries now being-used/allocated in the kernel.
inod-sz	Number of i-node table entries now being-used/allocated in the kernel.
file-sz	Number of file table entries now being-used/allocated in the kernel.
ov	Number of entries over the allocated amount, from the previous field.
lock-sz	The number of shared memory record table entries now being-used/allocated in the kernel.
fhdr-sz	The number of shared memory file table headers now being-used/allocated in the kernel.

The values are given as *level/table size*. An example of **sar -v** follows:

```
unix unix 2.0.3 2 3B2     07/19/85

17:36:05 text-sz ov proc-sz ov inod-sz ov file-sz ov lock-sz fhdr-sz
17:36:35    9/ 80  0  17/ 40  0  39/ 80  0  29/ 80  0    0/ 50    0/  5
17:37:05   11/ 80  0  19/ 40  0  46/ 80  0  35/ 80  0    0/ 50    0/  5
17:37:35   10/ 80  0  18/ 40  0  43/ 80  0  34/ 80  0    0/ 50    0/  5
```

This example shows that all tables are large enough to have no overflows, sizes could be reduced to save main memory space if these are the highest values ever recorded.

sar -w

The **-w** option reports swapping and switching activity. The following are some target values and observations.

swpin/s Number of transfers into memory per second.

bswin/s Number of 512-block units (blocks) transferred for swap-ins (including initial loading of some programs) per second.

swpot/s Number of transfers from memory to the disk swap area per second. If greater than 1, memory may need to be increased. Since **swpin/s** also include program loads, **swpot/s** provides the better measure of the number of swaps. At system start-up **swpot/s** may be larger than **swpin/s** because of the loading of sticky-bit programs.

bswot/s Number of blocks transferred for swap-outs per second.

pswch/s Process switches per second. This should be 30 to 50 on a busy 2 to 6 user system.

An example of **sar -w** output follows:

```
unix unix 2.0.3 2 3B2    07/19/85

19:53:44 swpin/s bswin/s swpot/s bswot/s pswch/s
19:53:58    3.41    56.0    1.23    15.7      37
19:54:14    1.59    23.9    0.89    17.8      39
19:54:24    1.34    28.4    0.21     1.6      39

Average     2.17    36.3    0.84    13.0      38
```

This example shows swap activity at an acceptable level.

sar -y

The **-y** option monitors terminal device activities, it is useful if you have many terminal I/Os. Activities recorded are defined as follows:

rawch/s Input characters (raw queue) per second.

canch/s Input characters process by canon (canonical queue) per second.

outch/s Output characters (output queue) per second.

rcvin/s Receiver hardware interrupts per second.

xmtin/s Transmitter hardware interrupts per second.

mdmin/s Modem interrupts per second.

An example of **sar -y** output follows:

```
unix unix 2.0.3 2 3B2    07/19/85

16:50:11 rawch/s canch/s outch/s rcvin/s xmtin/s mdmin/s
16:50:41     112      15     653     103     102       0
16:51:11     107       7     654     104     105       0
16:51:41      99       5     641      99     105       0

Average      106       9     649     102     104       0
```

sar -A

The **sar -A** option is equivalent to **sar -udqbwcayvm**. All reportable kernel operations will be reported, these are described in the previous **sar** examples. This option is helpful for doing overall system performance, single report options are recommended for tuning specific areas. After tuning the kernel for individual functions, use the -A option to verify that you did not lose performance in the areas where the system spends most of its time.

An example of **sar -A** follows:

```
unix unix 2.0.3 2 3B2    07/19/85

18:42:47  %usr    %sys    %wio   %idle
        device   %busy   avque   r+w/s  blks/s  avwait  avserv
       runq-sz %runocc swpq-sz %swpocc
       bread/s lread/s %rcache bwrit/s lwrit/s %wcache pread/s pwrit/s
       swpin/s bswin/s swpot/s bswot/s pswch/s
       scall/s sread/s swrit/s  fork/s  exec/s rchar/s wchar/s
        iget/s namei/s dirbk/s
       rawch/s canch/s outch/s rcvin/s xmtin/s mdmin/s
       text-sz ov proc-sz ov inod-sz ov file-sz ov lock-sz    fhdr-sz
         msg/s  sema/s


18:43:11    56      34      10       0
        hdsk-0      47     1.8      22      68    17.9    21.7
           1.5      87     1.2      17
            20      69      71       1       3      72       0       0
          0.57    19.6    0.20     6.3      33
            68       8      46    0.10    0.13   24513   23285
            17       6      13
             0       0     210       0       4       0
        12/ 80  0  18/ 40  0  43/ 80  0  36/ 80  0   0/ 50      0/  5
          0.00    0.00
```

sag

The command **sag** graphically displays the system activity data stored in a binary data file by **sar**. Any of the **sar** data items may be plotted separately or in combination. **Sag** operates by invoking **sar** and string matching the data column header. Running **sar** will show what data is available. An example of **sag** output with default options are shown in the following figure:

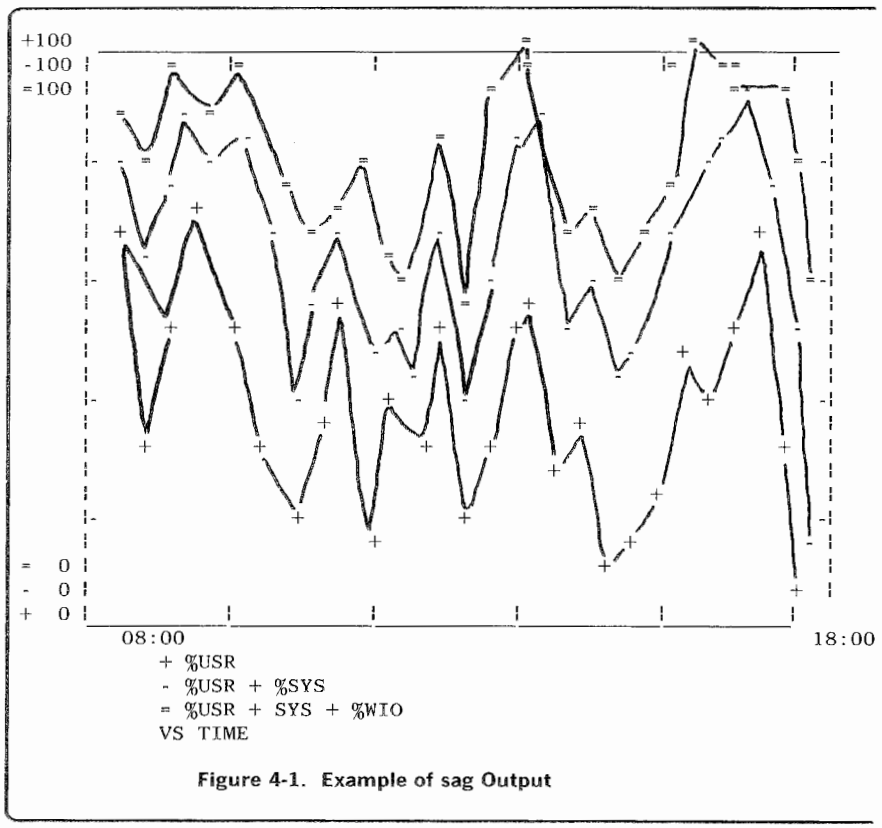


Figure 4-1. Example of sag Output

In Figure 4-1, the processor is completely used over three intervals 9–10 AM, 1–2 PM, and 4:30–5:30 PM. Remember the fraction of time that the processor is busy is the sum of user (**%usr**) mode time and system (**%sys**) mode time. When this approaches 100 percent, the processor is running at its maximum capacity as configured. The sum of **%usr** + **%sys** + **%wio** is about the same as the sum of **%usr** + **%sys** (**%wio** is low). This means that the disk subsystem is able to handle all requests that the processor generates with little delay. From this example, the first place to look to reduce any bottleneck is in reducing processor load.

> ***Note:*** The **sag** command is only useful if you have a standard output device that can read plotting instructions. Refer to the **tplot** manual page in the *AT&T 3B2 Computer User Reference Manual* for a list of known terminals with this capability.

timex

The **timex** command times a command and reports the system activities that occurred during the time the command was executing. The options that may be used with **timex** are described in Chapter 3. If no other programs are running, then **timex** can give you a good idea of which resources a specific command uses during its execution. System consumption can be collected for each application program and used for tuning the heavily loaded resources.

For our example, the *date* command is used. Enter the following:
$timex -s date<*CR*>

```
Fri July 19 20:46:33 EDT 1985

real             0.20
user             0.01
sys              0.15

unix unix 2.0.3 2 3B2   07/19/85

20:46:33         %usr              %sys              %wio             %idle
20:46:34           10                90                 0                 0

20:46:33 bread/s lread/s %rcache bwrit/s lwrit/s %wcache pread/s pwrit/s
20:46:34       0     151     100       0      13     100       0       0

20:46:33 device %busy avque r+w/s blks/s avwait avserv
20:46:3   dsk-0    22   1.2     3     15   10.1   18.4

20:46:33 rawch/s canch/s outch/s rcvin/s xmtin/s mdmin/s
20:46:34       0       0      44       0       3       0

20:46:33 scalls/s sreads/s swrit/s fork/s exec/s rchar/s wchar/s
20:46:34      176       18       6   4.41   5.88   23076     454

20:46:33 swpin/s bswin/s swpot/s bswot/s pswch/s
20:46:34    0.00     0.0    0.00     0.0      13

20:46:33 iget/s namei/s dirbk/s
20:46:34     43      15      29

20:46:33 runq-sz %runocc swpq-sz %swpocc
20:46:34     1.0     100

20:46:33 text-sz ov proc-sz ov inod-sz ov file-sz ov lock-sz fhdr-sz
20:46:34    8/30  0   12/60  0  30/100  0  19/100  0   0/100    0/20

20:46:33 msg/s sema/s
20:46:34  0.00   0.00

$
```

This is not the best example for the following reason: the date command is not a major user of system resources. Date was used for its simplicity.

Timex can be used in the following way:

$timex -s <application program><*CR*>

Your application program will operate normally, when you finish and exit, the **timex** result will be printed on your screen. This can be extremely interesting, you might see resources being used that your program never reflected. Such as, disk usage, etc.

sadp

The **sadp** command (**sadp** [**-th**] [**-d** *device*[*-drive*]] **s** [*n*]) reports disk access locations and seek distance in tabular (**-t**) or histogram (**-h**) form. Disk activity is sampled once every second, during a specified interval, repeatedly *n* times. Refer to Chapter 3 for details.

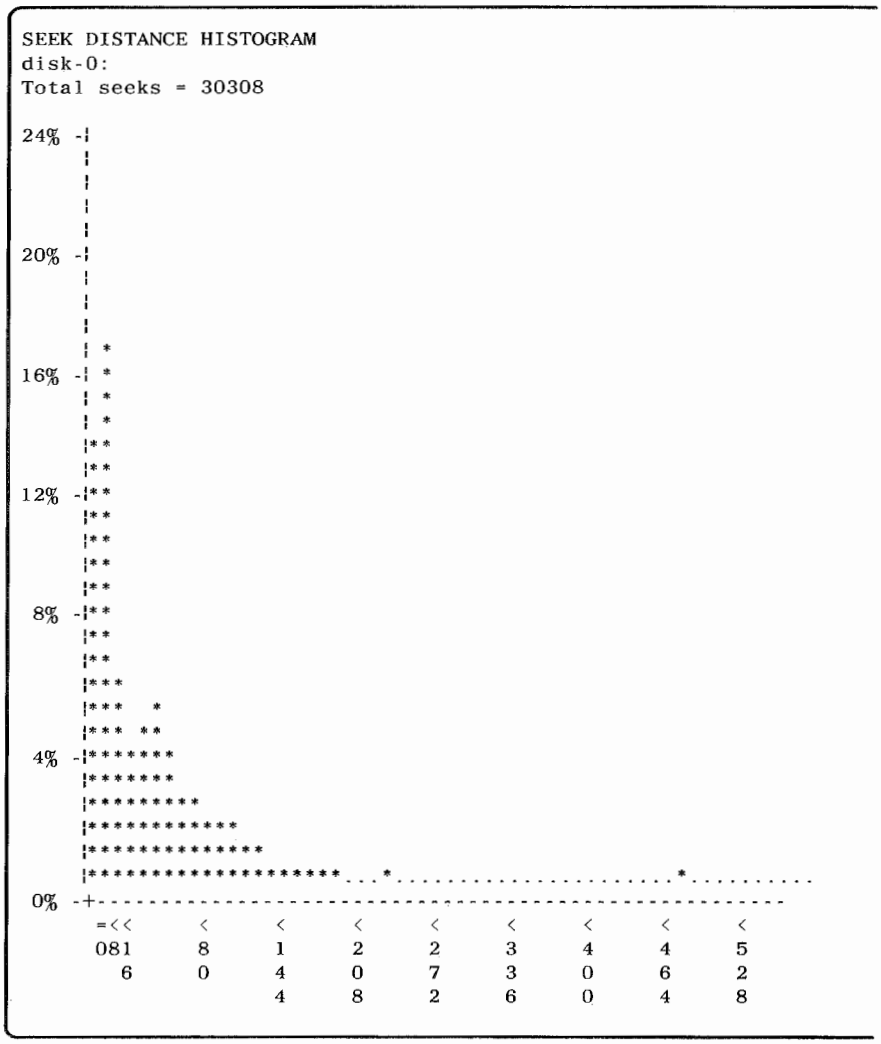
Using the **sadp** output along with the output of **/etc/mount** and the table of disk sections, you can identify the file systems with a large amount of I/O activity. Try to move areas of high activity close together. This will reduce the amount of seeks over large distances.

An example of getting a **sadp** output for hard disk drive 0 follows. The first graph shows excellent disk cylinder locality of references. Most references occur for files near cylinders 450 to 600, with a few for files around cylinder 250. The second graph shows that most physical seeks are under ten cylinders, all within the area of cylinders 450 to 600. This is an excellent file system configuration.

Enter the following command:

$sadp -h -d hdsk-0 3600<*CR*>

```
CYLINDER ACCESS HISTOGRAM
disk-0:
Total transfers = 33291

12% -|
     |
     |                                        *
     |                                        *
     |                                        *
10% -|                                        *
     |                                        *
     |                                        *
     |                                        *    *
     |                                        *    *
 8% -|                                        *    *
     |                                        *    *
     |                                       **    *
     |                                       **    *
     |                                       **    * *
 6% -|                                       **    * *
     |                                       **    * *   *
     |                                       **    * *   *
     |                                       **    * *   *
     |                                       **    * *   *
 4% -|                                       **    * *   *
     |                                       **    * *   **
     |                                       **    ****  **  *
     |                                       ***   **** ***  *
     |                                       ***   ********* *  * *
 2% -|                    *                  ***  ************  * *
     |                    *                  ***  *****************
     |*                   *                  ***  *****************
     |*                   *                  **** *****************
     |**                  *                  **** *****************
     |**.....*.......... ..**...*.....*.....*******************
  0%-+-------------------------------------------------------------
            Cylinder number, granularity=8
      081    8    1    2    2    3    4    4    5
        6    0    4    0    7    3    0    6    2
                  4    8    2    6    0    4    8
```


SEEK DISTANCE HISTOGRAM
disk-0:
Total seeks = 30308
24%
20%
16%
12%
8%
4%
0%
=<<
081
6
8
0
1
4
2
0
2
7
4
8
3
2
3
6
4
0
4
6
5
2
8

LOAD MANIPULATION AND HOUSEKEEPING

Introduction

After the kernel and the system activities are tuned, and the file systems organized, the next step for improving system performance is to do some housekeeping activities and to check whether prime time load can be reduced. The person responsible for administrating the system should check for:

- Less important jobs interfering with more important jobs.
- Unnecessary things being done.
- Scheduling jobs for when the system is not so busy.
- The efficiency of user-defined features such as *.profile* and $PATH.

ps

The **ps** command is used to obtain information about active processes. Without any *options*, information is printed about the terminal processes used to input the request. The **ps** command gives a "snapshot" picture of what is going on. This is useful when trying to identify what processes may be loading the system. Things will probably change by the time the output appears; however, the entries that you should be interested in are **TIME** (minutes and seconds of CPU time used by processes) and **STIME** (time when process started).

User $PATH Variables

$PATH is searched on each command execution. Before outputting "not found" the system must search everything in $PATH. These searches require both processor and disk time. If there is a disk or processor bottleneck, changes here can help performance.

Some things that you should check for in user $PATH variables are:

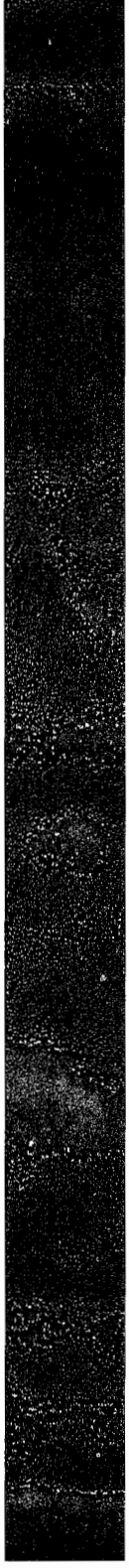
Path Efficiency
: $PATH is read left to right, so the most likely places to find the command should be first in the path (*/bin* and */usr/bin*). Make sure that a directory is not searched more than once for a command.

Convenience and Human Factors
: Users may prefer to have the current directory listed first in the path (*:/bin*).

Path Length
: In general, $PATH should have the least amount of required entries needed.

Large Directory Searches
: Searches of large directories should be avoided if possible. Put any large directory paths at the end of $PATH.

The person responsible for administrating the system should know if the system is being misused.



Replace this

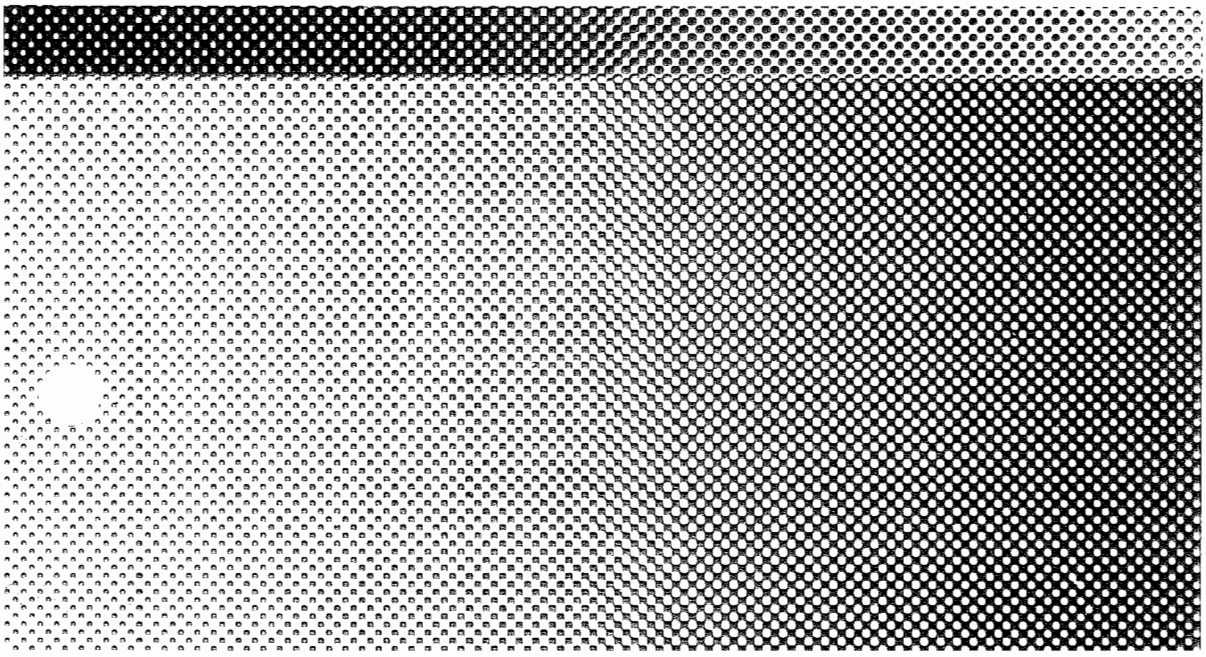
page with the

SPELL

tab separator.

AT&T 3B2 Computer

UNIX™ System V Release 2.0
Spell Utilities Guide



CONTENTS

Chapter 1. INTRODUCTION

Chapter 2. COMMAND DESCRIPTION

Chapter 3. SPELL UTILITIES ADMINISTRATION

Chapter 1

INTRODUCTION

PAGE

GENERAL 1-1
GUIDE ORGANIZATION 1-2
PREREQUISITES 1-2

Chapter 1

INTRODUCTION

GENERAL

This document describes the Spell Utilities command available with the AT&T 3B2 Computer. The **spell** command helps you to correct the spelling errors in your source files. When you run the **spell** program on a source file, all words contained in the source file are checked against another file that contains a dictionary. Output is printed on the terminal monitor or can be directed to a printer for hard copy or to another file using a file name of your own invention.

The **spell** program will neither correct the misspelled words of the source file nor will it give you a list of the corrected misspelled words. Words found by the **spell** program are simply those that it cannot find in its dictionary. Once the misspelled words have been found, you may correct the spelling by editing the source file.

GUIDE ORGANIZATION

This guide is structured so you can easily find desired information without having to read the entire text. This document describes how to use the **spell** command and also provides examples to aid you.

The information in this document is organized on a chapter basis as follows:

- Chapter 2, "COMMAND DESCRIPTION," describes the format and use of the command provided by the Spell Utilities.
- Chapter 3, "SPELL UTILITIES ADMINISTRATION," provides a description of the files used to monitor and update the performance of **spell**.

PREREQUISITES

Before continuing with this document, you should be familiar with the *AT&T 3B2 Computer Owner/Operator Manual* that describes how to operate the computer. You should also be familiar with the **UNIX*** *System V User Guide* that explains how to use the UNIX System.

* Trademark of AT&T

Chapter 2

COMMAND DESCRIPTION

PAGE

HOW COMMANDS ARE DESCRIBED 2-1

COMMAND FORMAT 2-3

SAMPLE COMMAND LINES 2-4

- Using spell By Itself 2-4
- Using spell On Text Files 2-5
- Using the -b Option 2-7
- Using the -i Option 2-8
- Using the -l Option 2-10
- Using the -v Option 2-12
- Using the -x Option 2-13
- Using the +local_file Option 2-14
- Combining Options 2-17

Chapter 2

COMMAND DESCRIPTION

This chapter provides the casual 3B2 Computer user with a tutorial of the **spell** command. The command is described when used alone and when used with options. The given examples show how to apply the **spell** command.

HOW COMMANDS ARE DESCRIBED

The **spell** command is described on a function or task level. Within the function there exists the following sections:

- **Command Format:** The basic command line format (syntax) is defined and the various arguments and options are discussed.
- **Sample Command(s):** Example command line entries and system responses are provided to show you how to use the command. A **$** represents the shell prompt.

In the command format discussion, the following symbology and conventions are used to define the command syntax:

- The basic command is shown in bold type. For example, **command** is in bold type.
- Arguments that you must supply to the command are shown in a special type. For example: **command** *argument*

In the sample command discussions, user inputs and 3B2 Computer response examples are shown as follows:

```
This style of type is used to show system generated
responses displayed on your screen.
```

This style of bold type is used to show inputs entered from your keyboard that are displayed on your screen.

These bracket symbols, < > identify inputs from the keyboard that are not displayed on your screen, such as: <*CR*> carriage return, <*CTRL d*> control d, <*ESC g*> escape g, passwords, and tabs.

This style of italic type is used for notes that provide you with additional information.

Refer to the *AT&T 3B2 Computer User Reference Manual* and the *AT&T 3B2 Computer System Administration Reference Manual* for UNIX System V manual pages supporting the commands described in this guide.

Because of its simplicity, the **ed** text editor is used for editing examples of text.

COMMAND FORMAT

The **spell** command can be used alone on a command line (to check individual words) or with arguments (to check files). When used with arguments, the general format for the **spell** command is:

spell [*options*] [+*local_file*] *filename*

The *filename* argument is either the source file(s) or a path name to a source file to be checked for spelling errors. The *local_file* is a user-generated file that enables the **spell** program to recognize special words not in the spelling list.

The options are as follows:

-b	Checks the source file or words for British spelling.
-i	Ignores all chains of files (pathnames) within a source file.
-l	Checks spelling throughout all chains of files (such as ".so" and ".nx" request lines) within a source file.
-v	Identifies all inflected words (containing prefixes or suffixes) and words not found in the spelling list.
-x	Lists all words and roots of inflected words within a source file. Also, shows the words not found in the spelling list.

SAMPLE COMMAND LINES

When using the **spell** command by itself or in the command format, the results are displayed on the terminal monitor by default. The examples show that you can redirect the results into a file for later viewing.

Using spell By Itself

The **spell** command can be used to check the spelling of one word or many words once they are entered on the terminal. This saves you from having to generate a new file, add words to the file, and check the file through **spell**. The following shows the use of the **spell** command directly from the terminal.

```
$ spell<CR>
mispell<CR>
hello<CR>
console<CR>
<CTRL d>
mispell
$
```

The **spell** command is entered on the terminal keyboard followed by a carriage return. Each word in question is entered, one per line. When you finish entering the words, enter a *<CTRL d>* (control d) on the terminal keyboard. The **spell** program will then check the spelling of each word entered and print the misspelled words on the terminal.

Using spell On Text Files

We can further analyze how the **spell** command works by looking at example files before executing the command, then examining the results after execution.

> ***Note:*** To duplicate the following examples, you will need to create similar "example" files on your computer.

First, cat the file "example1" to examine its contents, and then do a spelling check as follows:

```
$ cat example1<CR>
color
program
looked
have
prepared
mispell
$ spell example1<CR>
mispell
$
```

As you can see, the only word found misspelled by the **spell** program is "mispell." Using the **ed** text editor, delete the word "mispell" from the file example1, and run another spelling check.

```
$ ed example1<CR>
43
/mispell<CR>
mispell
d<CR>
w<CR>
35
q<CR>
$ spell example1<CR>
$
```

The $ prompt is returned to the monitor indicating no misspelled words in the list. The result would have been the same if you had used **ed** to correct the misspelled word. *Note that the* **spell** program does not correct the spelling for you. The **spell** program simply looks in its spelling list to find a word that matches a word from the source file. If a matching word is found, the **spell** program concludes the word is spelled correctly. However, if no match is found, the **spell** program displays the word on the terminal monitor.

> ***Note:*** A word printed on the terminal monitor does not always mean the word is misspelled. Some technical words, for example, are not in the spelling list of the **spell** program.

If you would like to view spelling errors at a later time, you can redirect the **spell** program output into another file. The following gives an example of how you might redirect the **spell** output.

```
$ spell bigfile > errors<CR>
$
```

The **spell** program is complete when you receive the system prompt. Any spelling errors that may have been found are now located in the "errors"

file. You can cat or edit the errors file at your convenience. Note that if you redirect output to an existing file, the original contents of the file will be erased.

Using the -b Option

Let's look at another example.

```
$ cat example2<CR>
colour
programme
looked
have
prepared
$ spell example2<CR>
colour
programme
$
```

The words "colour" and "programme" are shown misspelled by the **spell** program. Colour and programme are spelled correctly as far as the British are concerned. The **spell** program, as run in the previous examples, uses an American spelling list. Colour and programme in America are the words color and program, respectively.

Once the use of British words has been determined, the -b option can be used. With this option a British spelling list is used instead of the American spelling list. Only British spelling is considered correct. Using the -b option with the **spell** program will result in the following:

```
$ spell -b example2<CR>
$
```

With the -b option, the **spell** program finds no errors and the **$** prompt is returned to the monitor. The words colour and programme are now considered correct spelling.

Using the -i Option

When creating text, you may have used the ".so" or ".nx" macros. These macros specify a file to be read whenever the original file is executed. When the **spell** command is executed on a file containing .so or .nx macros, the contents of these macro-specified files are also considered (there is one exception — see -l option). The -i option, however, will cause **spell** to ignore any .so or .nx macros within the source file. The following shows how the -i option can be used.

First examine the file "example3":

```
$ cat example3<CR>
.so /usr/bin/example4
mispell
$
```

Note the .so macro that specifies the file example4. Now, cat the file example4 that is located in the /usr/bin directory:

```
$ cat /usr/bin/example4<CR>
corect
$
```

Without the -i option, execute **spell** on the file example3:

```
$ spell example3<CR>
corect
mispell
$
```

Note that the results include misspellings from both the example3 and example4 files.

You may not want the **spell** program to include files specified by the .so or .nx macros. For example, the macro-specified file could be a program that contains some non-text information. If example4 was such a file, you could use the -i option as follows:

```
$ spell -i example3<CR>
mispell
$
```

Here, the example4 file specified by the .so macro was ignored by the **spell** program.

Using the -l Option

When the **spell** command without the -l (and -i) option is executed on a file containing .so or .nx macros, all words within the original file and macro-specified file are checked. However, an exception to this rule does exist. If these macros specify a file located in the /usr/lib directory, **spell** will ignore this file. Here, only the words in the original file will be checked for spelling.

With the -l option, **spell** can search all macro-specified files for misspellings, including those located within the /usr/lib directory. The following example will illustrate the use of the -l option.

Examine the file "example5":

```
$ cat example5<CR>
.so /usr/lib/example6
chpter
$
```

Note the macro-specified file "example6" located in the /usr/lib directory. Now, cat the example6 file:

```
$ cat /usr/lib/example6<CR>
incorect
$
```

Without the -l option, run spell on example5:

```
$ spell example5<CR>
chpter
$
```

Notice that only "chpter" from the example5 file was found. Using the -l option:

```
$ spell -l example5<CR>
chpter
incorect
$
```

Both "chpter" from example5 and "incorect" from /usr/lib/example6 were found.

Using the -v Option

The -v option identifies the inflected words (words containing prefixes and suffixes) of a source file. The file being spelled is checked against the spelling list. Prefixes, suffixes, and other inflections of each inflected word are output preceded by a plus (+) sign. All words not in the spelling list are also output; however, they are not preceded by the plus sign. The words without plus signs are words the **spell** program considers misspelled. The following shows what happens when you use the -v option:

```
$ spell -v example1<CR>
+ed      looked
+d       prepared
$
```

In the above example, "ed" is the suffix of "looked." The word "prepared" also contains a suffix.

Using the -x Option

The file being spelled is checked against the spelling list. All words of the source file are output preceded by an equal sign (=). The root of any inflected words (words containing prefixes or suffixes) found within the source file are also displayed with an equal sign. The words not found in the spelling list are also output; however, they are not preceded by the equal sign. Look at the following example that utilizes the -x option:

```
$ spell -x example1<CR>
=color
=have
=looked
=looke
=look
=prepared
=pared
=prepare          Note that "mispell" was deleted
=program          earlier in this chapter.
$
```

In the above example, the plausible roots of the word in question are analyzed and printed on the terminal monitor. Using **ed**, reintroduce the misspelled word "mispell" into the example1 file.

```
$ ed example1<CR>
35
a<CR>
mispell<CR>
.<CR>
w<CR>
43
q<CR>
$
```

Now, when executing **spell**, the results are:

```
$ spell -x example1<CR>
=color
=have
=looked
=looke
=look
=mispell
=pell
=prepared
=pared
=prepare
=program
mispell
$
```

Note that the word "mispell" is displayed both with and without an equal sign.

Using the +local_file Option

The text of technical subjects often contains words that are not common English words. These words, which are not in the American or British spelling list, will show up as being misspelled. A user-generated spelling list (local_file) which contains these technical words will help you to solve this problem.

Acronyms, user-tailored macros, or any other user-defined text can be used in this special spelling list. The user simply stores the user-defined words in the local_file. How to create such a file is shown below. When using the local_file with **spell**, it must be preceded by a plus sign (+).

Note: When using the local_file option, you must be careful to put all uppercase words in front of the file. If this is not done, the first lowercase word seen by the **spell** program will inhibit the program from recognizing any uppercase words. If these instructions are not followed, the **spell** program may not run to completion.

When **spell** is run, both the American/British (depending on the option) spelling list and the user-generated local_file are used to form an overall spelling list. First examine and then check the file "example7" for spelling errors.

```
$ cat example7<CR>
UART
computerese
mispell
$ spell example7<CR>
UART
computerese
mispell
$
```

The contents of the example7 file are found to be incorrectly spelled. The words "UART" and "computerese" are terms associated with computers and are actually correctly spelled. By using **ed**, a local_file called "fixit" can be generated in order that we can consider these words as correctly spelled.

```
$ ed fixit<CR>
?fixit
a<CR>
UART<CR>
computerese<CR>
mispell<CR>          Note that the uppercase word UART
.<CR>                is placed at the front of the file.
w<CR>
25
q<CR>
$
```

Now, by running the **spell** program with the local_file fixit, you get:

```
$ spell +fixit example7<CR>
$
```

The fixit file finds the three words and considers them correctly spelled. As you can see, even misspelled words (like "mispell") can be considered correct if a user-generated spell list is used.

Combining Options

The following examples are shown here to show the various combinations of options.

Once again, **spell** is executed on the file example2, but with the -v option included.

```
$ spell -v example2<CR>
colour
programme
+ed       looked
+d        prepared
$
```

With only the -v option specified, **spell** defaults to the American spelling list. Options can be combined as in this example:

```
$ spell -b -v example2<CR>
+ed     looked
+d      prepared
$
```

Further examples of how the options can be combined are shown below:

```
$ spell -x -v example1<CR>
=color
=have
=looked
=looke
=look
=mispell
=pell
=prepared
=pared
=prepare
=program
mispell
+ed      looked
+d       prepared
$
```

```
$ spell -x -v -b example1<CR>
=color
=have
=looked
=looke
=look
=mispell
=pell
=prepared
=pared
=prepare
=program
color
mispell
program
+ed      looked
+d       prepared
$
```

```
$ spell -x -v -b example2<CR>
=colour
=have
=looked
=looke
=look
=prepared
=pared
=prepare
=programme
+ed        looked
+d         prepared
$
```

Chapter 3

SPELL UTILITIES ADMINISTRATION

PAGE

/usr/lib/spell/compress 3-2
/usr/lib/spell/hashcheck 3-2
/usr/lib/spell/hashmake 3-2
/usr/lib/spell/spellhist 3-2
/usr/lib/spell/spellin *n* 3-3
/usr/lib/spell/hlista 3-3
/usr/lib/spell/hlistb 3-3
/usr/lib/spell/hstop 3-3

Chapter 3

SPELL UTILITIES ADMINISTRATION

This chapter describes the Spell Utilities files that are used to monitor and update the performance of **spell**. These files consist of hashed (encoded) word lists and special programs. You can monitor the performance of **spell** by reviewing a file containing a history of misspelled words. You can also adjust the utilities if a misspelled word goes undetected or if there are words you would like to add to the spelling list.

> ***Note:*** The administrative files described here are intended to be used by root or the system administrator. To change the Spell Utilities files, requires some experience in working with the UNIX System. If the casual user finds a problem with **spell** that could possibly be remedied through these administrative files, the user should notify the system administrator.

The following paragraphs provide an explanation of each administrative file.

/usr/lib/spell/compress

Each time **spell** is executed, the misspelled words found are appended to a file called spellhist. The spellhist file, therefore, contains a history of all misspelled words found during all executions of **spell**. Identical entries are made owing to the same word being misspelled during different executions of **spell**. The spellhist file is useful since it can be checked for determining the accuracy of the **spell** program. The "compress" file, when executed, deletes redundant misspelled words, therefore reducing the size of the spellhist file and making it much easier to analyze.

/usr/lib/spell/hashcheck

The "hashcheck" program recreates 9-digit hash codes for a compressed spelling list. The compressed spelling list can be an existing spelling list (hlista or hlistb) or a list created or modified by the "spellin" program. By using hashcheck on a spelling list and hashmake on a file of selected words, you can determine whether the words are present in the list.

/usr/lib/spell/hashmake

The "hashmake" program generates a unique hashcode for each word. Each hashcode consists of a 9-digit octal number. This conversion is made to speed up the execution of **spell**.

/usr/lib/spell/spellhist

Each time **spell** is executed, an entry is made of the misspelled words into the spellhist file. Since all entries are appended, a history of all past misspelled words is collected. Through examining the file, you may find entries that are not misspelled. In this instance a new entry into the **spell** dictionary would be needed. The spellhist file is useful in following the performance of the **spell** program.

/usr/lib/spell/spellin *n*

The "spellin" program reads 9-digit hashcodes created through hashmake and compresses them into a spelling list. The *n* argument specifies the number of hashcodes to be read. The spellin program is used to add words to an existing spelling list or to create a new spelling list.

/usr/lib/spell/hlista

This file contains the American dictionary used by **spell**. Each word in the dictionary is represented by its unique hashcode. Each hashcode consists of a 9-digit octal number. When **spell** is executed on a file, the words in the file are checked against the "hlista" American dictionary. Unless an option specifies otherwise, **spell** will use the American dictionary by default.

/usr/lib/spell/hlistb

This file contains the British dictionary used by **spell**. Each word in the dictionary is represented by its unique hashcode. Each hashcode consists of a 9-digit octal number. When **spell** with the -b option is executed on a file, the words in the file are checked against the "hlistb" British dictionary. The **spell** program will use the American dictionary by default.

/usr/lib/spell/hstop

The "hstop" file is used by **spell** to filter out misspelled words that would normally go undetected. In combining the possible inflections with the word, the **spell** program may wrongly conclude that a word is spelled correctly. For example, **spell** can derive the misspelled word "thier" by subtracting the "y" from "thy" and adding "ier." You can stop or prevent this from occurring by making an entry into the hstop file or by using the -v option.

Replace this

page with the

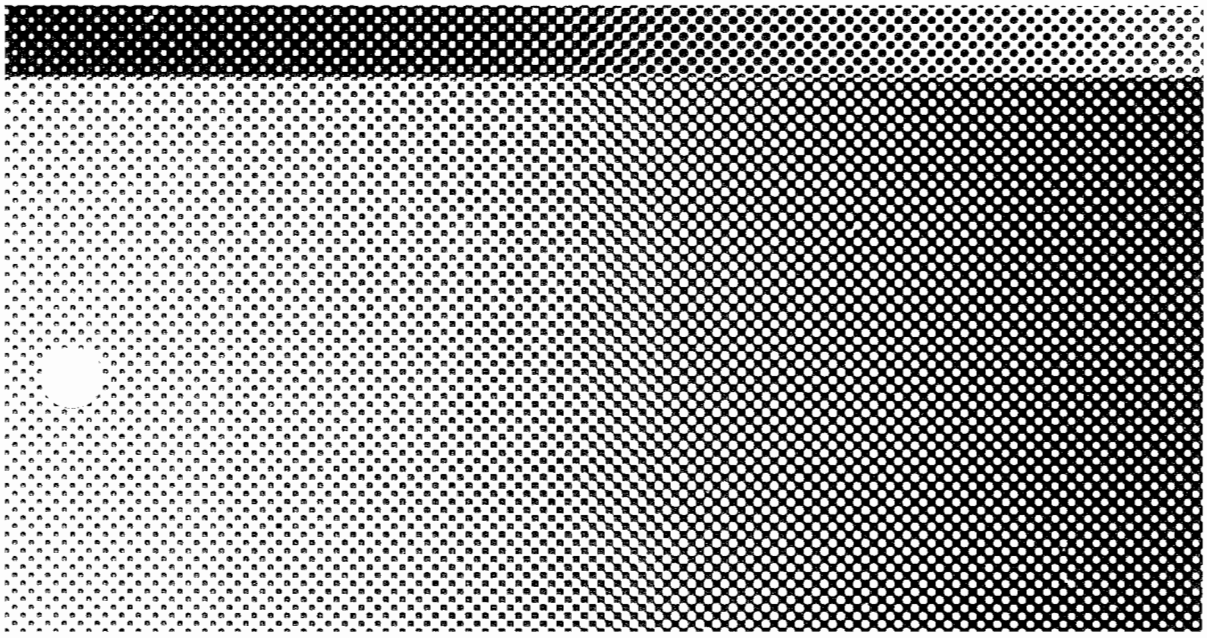
TERMINAL FILTERS

tab separator.

AT&T

AT&T 3B2 Computer

UNIX™ System V Release 2.0
Terminal Filters
Utilities Guide



CONTENTS

Chapter 1. INTRODUCTION

Chapter 2. COMMAND DESCRIPTIONS

Chapter 1

INTRODUCTION

PAGE

GENERAL 1-1

GUIDE ORGANIZATION 1-4

Chapter 1

INTRODUCTION

GENERAL

This manual describes the Terminal Filter Utilities available with the AT&T 3B2 Computer. These utilities allow a variety of terminals to do NROFF and manual page (MAN) text formatting. NROFF is a text processor that formats text for typewriter-like terminals. MAN is a text processor used to format the **UNIX*** System command manual pages.

A terminal filter is exactly what the name implies. The filter takes the incoming data and filters it through certain processes to check the compatibility with the associated terminal. For example, NROFF uses certain characters to identify operations within its format process. Sometimes, these characters mean different operations to different terminals. An "ESCAPE 9" on one terminal may be a "margin set." On another terminal, it may be a "form feed" which advances your paper, or screen, by one page length.

---

* Trademark of AT&T

If this difference is not resolved, the NROFF formatted version of the file will be somewhat mixed up.

The terminal filters reinterpret the incoming data. These filters equate the incoming characters to compatible characters for the associated terminal. For example, a filter would read the "ESCAPE 9" mentioned earlier and equate that to the character on the associated terminal that will do the function that NROFF expects "ESCAPE 9" to perform.

Special characters are simulated by overstriking, if necessary and if possible. This means that if a character is not a common character to the terminal, but the character can be produced by a combination of two characters, the terminal filter will force the printer or terminal to strike one character, backspace, and strike another character. For example, a bullet is a solid circle (•). Most printers and terminals are not able to produce a solid circle. However, the letter "o" overlapped with a plus (+) makes a character that looks similar to a bullet (o+).

By using these terminal filters, NROFF and MAN are made compatible to all terminals handled by the seven terminal filter utilities. These filters produce a consistent output for the associated terminals. They are required for terminals/printers that require special actions involving backspaces, underlines, and bolding. Most new terminals/printers use a standard type of backspace that does not require a terminal filter.

NROFF was designed to operate on a TELETYPE* Model 37 terminal. This is known as the default terminal. NROFF also works on the TELETYPE Model 5410. Any other terminal that uses NROFF may require a terminal filter associated with the NROFF command line.

---

* Trademark of AT&T

The following is a list of the terminal filter utilities and some of their associated terminals:

TERMINAL FILTERS	ASSOCIATED TERMINALS
300	DASI 300, XEROX 820, DIABLO* 620 & 630, BROTHER 620
300s	DASI 300s
450	DASI 450, DIABLO 1620, TELERAY†, XEROX 1700
4014	TEKTRONIX‡ 4014 and 4014 Emulator, Retrofit-graphics equipped terminals
greek	(Refer to the manual pages in the *AT&T 3B2 Computer User Reference Manual*)
hp	HEWLETT-PACKARD 2640 and 2621-series
hpio	HEWLETT-PACKARD 2645A

* Registered Trademark of XEROX Corporation

† Trademark of Research Incorporated

‡ Registered Trademark of Tektronics

GUIDE ORGANIZATION

The remainder of this guide, Chapter 2 -- "COMMAND DESCRIPTIONS," describes the format and use of the commands provided by the Terminal Filter Utilities.

Chapter 2

COMMAND DESCRIPTIONS

PAGE

COMMAND SUMMARY 2-1

HOW COMMANDS ARE DESCRIBED 2-2

COMMANDS 2-5

300,300s — Handle Special Functions of DASI 300 and DASI 300s Terminals 2-5

4014 — Paginator for the TEKTRONIX 4014 Terminal 2-8

450 — Handles Special Functions of the DASI 450 Terminal 2-11

greek — Select Terminal Filter 2-13

hp — Handles Special Functions for HEWLETT-PACKARD 2640- and 2621-Series Terminals 2-15

hpio — HEWLETT-PACKARD 2645A Terminal Tape File Archiver 2-17

Chapter 2

COMMAND DESCRIPTIONS

COMMAND SUMMARY

The Terminal Filter Utilities provide seven UNIX System commands. A summary of these commands is provided in Figure 2-1.

COMMAND	DESCRIPTION
300, 300s	Handle special functions of the DASI 300 and DASI 300s terminal.
4014	Paginator for the TEKTRONIX 4014 terminal.
450	Handles special functions of the DASI 450 terminal.
greek	Used in the command line to select the correct terminal filter for your terminal.
hp	Handles special functions of the HEWLETT-PACKARD 2640- and 2621-series terminals.
hpio	Tape file archiver for the HEWLETT-PACKARD 2645A terminal.

Figure 2-1. Terminal Filter Utilities — Command Summary

HOW COMMANDS ARE DESCRIBED

Each terminal filter is either a command or an option to a command. These filters are described in this chapter by the following format:

- **General:** The purpose of the command is defined. Any special or uncommon information about the command is also provided.
- **Command Format:** The basic format (syntax) of the command line is defined and the various fields and options are discussed.

- **Sample Command(s):** Example command lines and system responses are provided to show you how to use the command.

In the command format discussions the following symbology and conventions are used to define the command syntax.

- The basic filter command is shown in bold type. For example, **filter** is in bold type.
- Fields that you must supply to the command are shown in a special type. For example: **filter** *field*.
- Command options and fields that do not have to be supplied are enclosed in brackets []. For example: **filter** [*optional fields*].

In the sample command discussions, user inputs and 3B2 Computer response examples are shown as follows:

```
This style of type is used to show system generated
responses displayed on your screen.

This style of bold type is used to show inputs
entered from your keyboard that are displayed on your
screen.

These bracket symbols, < > identify inputs from the
keyboard that are not displayed on your screen, such
as: <CR> carriage return, <CTRL d> control d, <ESC g>
escape g, passwords, and tabs.

This style of italic type is used for notes that
provide you with additional information.
```

Also in the sample command discussion, the dollar sign ($) is used as the system prompt. You will not enter the dollar sign. The pipe symbol (|) is used to show that the NROFF command is piped (sent) through the process following the pipe symbol.

This is shown by the following example:

nroff (file) ! process.

The sample commands shown in this chapter are designed to show the terminal filter portions of an NROFF command. The formatted output will be displayed on the screen and then disappear. To save the formatted version of the file, you must direct the output into another file using the greater than symbol (>). An example of this is:
nroff -T300 chap1 > **CHAP1**<*CR*>.
The formatted version of **chap1** will be saved in **CHAP1**.

Refer to the *AT&T 3B2 Computer User Reference Manual* for UNIX System V manual pages supporting the commands described in this guide.

COMMANDS

300,300s — Handle Special Functions of DASI 300 and DASI 300s Terminals

General

The **300** filter reinterprets special characters to use the DASI 300 (GSI 300 and DTC 300) terminal: **300s** filter performs the same functions for the DASI 300s (GSI 300s and DTC 300s) terminal.

These filters also attempt to draw Greek letters and other special characters. They permit convenient use of 12-pitch text and reduce the time needed for printing.

> ***Caution: If your terminal has a PLOT switch, make sure it is turned ON* before 300/300s is used.**

Command Format

The general format for the **300** and **300s** filter commands is as follows:

300 [*+12*] [*-n*]
300s [*+12*] [*-n*]

The optional entries can be used in any combination, if at all.

+12	Permits the use of 12-pitch, 6 lines per inch text. To obtain the 12-pitch, 6 lines per inch combination, you should turn the PITCH switch to 12 and use the *+12* option.

-n Controls the size of half-line spacing. This allows for individual taste in the placement and appearance of subscripts and supersubscripts. Half-line is 4 vertical increments by default.

Sample Commands

The **300** and **300s** filters are used as options within an NROFF command line. There are two ways to run NROFF through a terminal filter, and both ways are shown in the following examples.

The following command lines show how to format a file named **text** for a DASI 300 terminal:

```
$ nroff -T300 text<CR>

or

$ nroff text | 300<CR>
```

The following command lines show how to format a file called **book** using 12-pitch, 6 lines per inch on a DASI 300s:

```
$ nroff -T300s-12 book<CR>

or

$ nroff book | 300s +12<CR>
```

The following command lines show how to format a file named **doc** using quarter-lines instead of half-lines (half-lines equal 4 vertical increments by default) on a DASI 300:

```
$ nroff -T300 -2 doc<CR>

or

$ nroff doc | 300 -2<CR>
```

4014 — Paginator for the TEKTRONIX 4014 Terminal

General

The **4014** filter arranges for 66 lines to fit on the TEKTRONIX 4014 terminal screen. It also divides the screen into a specified amount of columns and puts an eight-space margin on the left side of the page, if the output is a single column.

At the end of each page, **4014** waits for a new-line signal from the keyboard before going to the next page. In this wait state, commands can be sent to the shell by using "!3! *command*."

Command Format

The general format for the **4014** filter is as follows:

4014 [*-t*] [*-cN*] [*-pL*] [*file*]

The optional entries are used to specify what type of format is to be used on the screen.

-t	Alters the output so that it does not wait for a new-line signal between pages. This is useful when formatting into another file.
-cN	Divides the screen into the number of columns specified by the number *N*.
-pL	Sets the length of the page to *L*. *L* can be expressed in inches (*i*) or lines (*l*); no input means lines.
file	Specifies the file that the filter is to reinterpret for display on the screen.

Sample Commands

The **4014** filter is used as an option within the NROFF command line, or as an outside command to which the NROFF formatting process is piped.

The following command lines show how to format a file named **text** for a TEKTRONIX 4014 terminal:

```
$ nroff -T4014 text<CR>

or

$ nroff text | 4014<CR>
```

The following command lines show how to format a file named **book** without waiting for a new-line between pages:

```
$ nroff -T4014 -t book<CR>

or

$ nroff book | 4014 -t<CR>
```

The following is a sample command line to show how to format a file named **group** and make the output pages 30 lines long with two columns:

```
$ nroff -T4014 -c2 -p30 group<CR>

or

$ nroff group | 4014 -c2 -p30<CR>
```

450 — Handles Special Functions of the DASI 450 Terminal

General

The **450** filter reinterprets NROFF output to use the DASI 450 terminal. This filter attempts to draw Greek letters and other special symbols.

> ***Caution: Make sure that the PLOT switch of your terminal is** ON* **before using 450**. The SPACING switch should be put in the desired (10- or 12-pitch) position.

Command Format

The general format for the **450** filter is:

450 [*+12*]

The *+12* option permits the use of 12-pitch text. To obtain the 12-pitch text, the SPACING switch on your terminal should be set to 12 and the *+12* option used.

Sample Commands

The **450** filter is used as an option within the NROFF command line, or as an outside command to which the NROFF formatting process is piped.

The following command lines show how to format a file named **text** for a DASI 450 terminal:

```
$ nroff -T450 text<CR>

or

$ nroff text | 450<CR>
```

The following command lines show how to format a file named **final** using a 12-pitch format on a DASI 450:

```
$ nroff -T450-12 final<CR>

or

$ nroff final | 450 +12<CR>
```

greek — Select Terminal Filter

General

Greek is a filter that reinterprets the character set used in NROFF for various other terminals. Special characters are simulated by overstriking, if necessary and if possible. If the terminal is not specified, **greek** attempts to use the **$TERM** variable set in your environment.

Command Format

The general format for a **greek** command is as follows:

greek [**-T***terminal*]

The **-T***terminal* option is how you specify what type of terminal you have. The **-T** followed by one of the following arguments specifies the terminal filter required.

ARGUMENTS	TERMINALS
300	DASI 300
300-12	DASI 300 (12-pitch)
300s	DASI 300s
300s-12	DASI 300s (12-pitch)
450	DASI 450
450-12	DASI 450 (12-pitch)
4014	TEKTRONIX 4014
hp	HEWLETT-PACKARD 2621, 2640, and 2645

Sample Commands

The following command shows how to format the file named **final** for 12-pitch text on a DASI 300s terminal:

```
$ nroff final | greek -T300s-12<CR>
```

hp — Handles Special Functions for HEWLETT-PACKARD 2640- and 2621-Series Terminals

General

The **hp** filter is designed to produce accurate representations of most NROFF output for HEWLETT-PACKARD 2640- and 2621-series terminals.

Regardless of the hardware options on your terminal, **hp** tries to accurately do underlines and reverse line-feeds.

Command Format

The basic command format for **hp** is as follows:

hp [*-e*] [*-m*]

The options available are used to enhance the output on the screen.

-e Assumes that your terminal has the "display enhancements" feature. Overstruck characters are presented in the *underline* mode. Superscripts are shown in *half-bright* mode, and subscripts in *half-bright*, *underline* mode. If this flag is omitted, **hp** assumes that your terminal does not have the "display enhancement" feature. Thus, all overstruck characters, subscripts, and superscripts are displayed in *inverse video*, that is dark-on-light, rather than the usual light-on-dark.

-m Requests removal of new-lines in the output. Any amount of successive blank lines is reduced to produce a single blank line in the output.

Sample Commands

The **hp** filter is used by piping (sending) the NROFF output through the filter.

The following command line shows how to NROFF a file named **doc** for a HEWLETT-PACKARD 2640 terminal:

```
$ nroff doc | hp<CR>
```

The following command line shows how to NROFF a file named **text** for a HEWLETT-PACKARD 2621A terminal. The command will minimize the blank lines in the output:

```
$ nroff text | hp -m<CR>
```

hpio — HEWLETT-PACKARD 2645A Terminal Tape File Archiver

General

The **hpio** command was designed to take advantage of the tape drives on HEWLETT-PACKARD 2645A terminals. Up to 255 UNIX System files can be archived onto a tape cartridge, depending on the size of the files, for off-line storage or transfer to another UNIX System.

Hpio always leaves the tape at a position after the last file affected on the tape. Tapes should always be rewound before the terminal is turned off.

Command Format

The general command format for **hpio** is as follows:

hpio -o[*rc*] *file* [*file*] [*file*] ...
hpio -i[*rta*] [*-n count*]

The **hpio -o** (copy out) copies the specified *file(s)*, together with path name and status information, to a tape drive on your terminal. The left tape drive is used by default. Each file is written to a separate tape file and ended with a tape mark.

The other options available are *r* and *c*. If *r* is entered after the **-o**, the right tape drive is used. The *c* will include a checksum at the end of each file. These options (*r* and *c*) can be used in any combination or not at all.

The **hpio -i** (copy in) extracts a file, previously written by an **hpio -o**, from a tape drive. The next file on the left tape drive will be extracted by default.

The *r,t,* and *a* options can be used after **-i** in any combination or not at all. A description of each option is as follows:

r	Uses the right tape drive instead of the left.
t	Prints a table of contents only. No files are created. Information in the table of contents includes file size, file names, access modes, and whether a checksum has been included.
a	Asks before creating a file. The file size and name are printed, followed by a question mark (?). Any response entered by you beginning with a **y** or **Y** will cause the file to be copied in as usual. Any other response will cause the file to be skipped.

The *-n count* option shows the number of input files to be extracted. For example, *-n 3* would extract 3 input files and stop. If this option is not given, *count* defaults to 1. If all files on a tape are needed, you may choose an arbitrarily large *count*.

Sample Commands

The following command line shows how to copy a file named **table** to a tape drive on your terminal:

```
$ hpio -o table<CR>
$
```

The following command line shows how to copy a file named **frame** out to the "right" tape drive of your terminal:

```
$ hpio -or frame<CR>
$
```

The following command line shows how to copy a file in from a tape drive on your terminal. The file **test** is the next file on the tape.

```
$ hpio -i<CR>
2 bytes: test
test: read complete
$
```

The following command line shows how to copy a file in from the "right" tape drive and have the terminal ask if you want the file to be copied or skipped. The next file on the tape is **chap1**.

```
$ hpio -ira<CR>
4 bytes: chap1?
```

If you enter a **y**, the file will transfer as usual. Anything else will skip the file **chap1** and go to the start of the next file.

The following command line shows how to copy in the next four files from the "right" tape drive. The names of the next four files are:

chap1, chap3, chap4, appen

```
$ hpio -ir -n 4<CR>
3 bytes: chap2
chap2: read complete
7 bytes: chap3
chap3: read complete
4 bytes: chap4
chap4: read complete
1 byte: appen
appen: read complete
$
```

Replace this

page with the

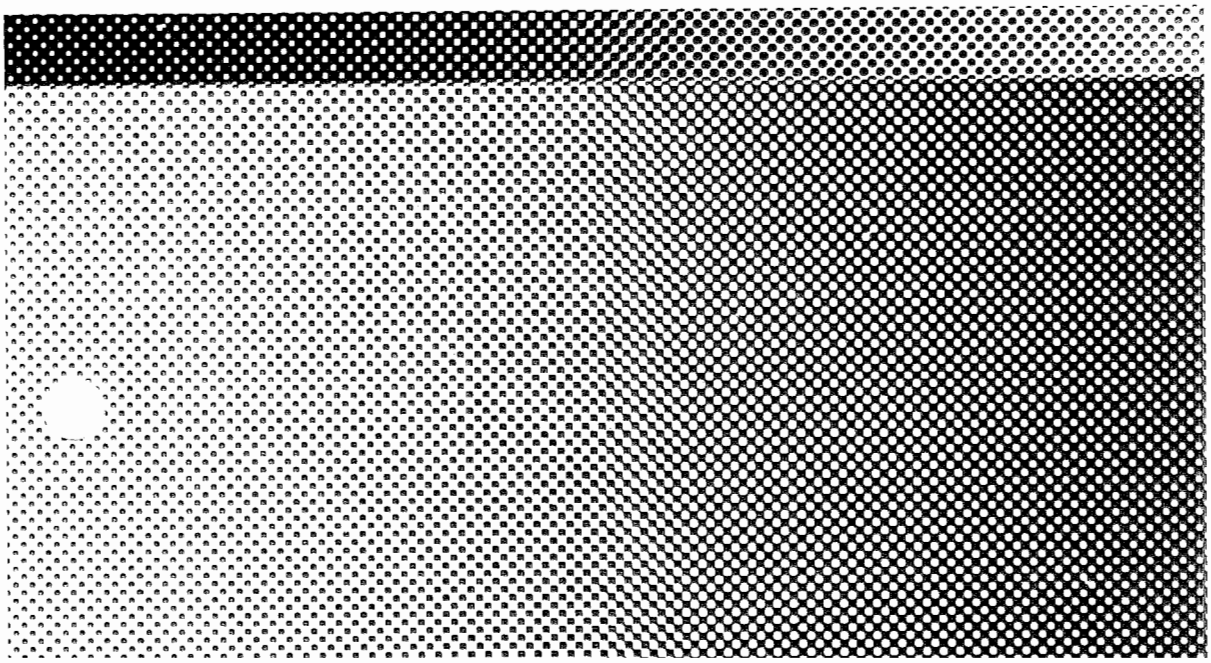
TERMINAL INFORMATION

tab separator.

AT&T

AT&T 3B2 Computer

UNIX™ System V Release 2.0
Terminal Information
Utilities Guide



CONTENTS

Chapter 1. INTRODUCTION

Chapter 2. SCREEN MANIPULATION

Chapter 3. WINDOW MANIPULATION

Chapter 4. MULTIPLE TERMINALS

Chapter 5. PORTABILITY FUNCTIONS

Chapter 6. LOWER LEVEL FUNCTIONS

Chapter 7. TERMINFO DATABASE

Appendix: CURSES EXAMPLES

Chapter 1

INTRODUCTION

PAGE

FEATURE DESCRIPTION 1-1

GUIDE ORGANIZATION 1-2

SPECIAL NOTATIONS 1-3

Functions 1-3

Comments 1-4

Chapter 1

INTRODUCTION

This guide describes the Terminal Information Utilities available with the AT&T 3B2 Computer. These utilities use the **terminfo** database and the **curses** library to improve the output of data to a video display terminal.

FEATURE DESCRIPTION

The **terminfo** database contains the descriptions of over 150 terminals. The terminals are described by giving a set of terminal capabilities and by describing how operations are performed. It is based on the **termcap** database, but contains many improvements and extensions.

The **curses** library is a collection of routines with the major function of "cursor optimization." **Curses** uses the **terminfo** database to obtain the specific information about the characteristics of any terminal that may be using the optimization function.

The capabilities described in this guide are intended for the sophisticated user with C Language programming experience who must write a screen-oriented program using the **curses** routines.

The **curses** programs are compiled as C Language programs. The general command line to compile a **curses** program is as follows:

cc *filename.c* **-lcurses -o** *FILENAME*

The *filename.c* variable is the name of the C Language program. The executable output of the compiled program is written to *FILENAME*.

GUIDE ORGANIZATION

This guide is structured so you may easily find the information that you are looking for without having to read the entire text. The remainder of this guide is organized as follows:

- Chapter 2, "SCREEN MANIPULATION," describes the screen manipulation capabilities provided by this utilities using examples of routines.
- Chapter 3, "WINDOW MANIPULATION," describes the window manipulation functions of the Terminal Information Utilities. Included in this chapter is a description of the pad manipulation capabilities.
- Chapter 4, "MULTIPLE TERMINALS," describes the procedures for accessing and using multiple terminals with the **curses** library functions.
- Chapter 5, "PORTABILITY FUNCTIONS," provides a description of the portability functions associated with **curses**.
- Chapter 6, "LOWER LEVEL FUNCTIONS," gives a brief explanation of those functions that do not need the screen optimization capabilities of **curses**. These functions are considered to be "lower level functions."

- Chapter 7, "TERMINFO DATABASE," describes the format and construction process of a **terminfo** entry.

- Appendix, "CURSES EXAMPLES," contains some larger, more useful examples of **curses** programs.

SPECIAL NOTATIONS

The following special notations and naming conventions are used throughout this guide.

Functions

The function names are shown in bold type followed by parentheses. The parentheses are used to designate the parameters to the function. The variables found in the parentheses represent the information that dictates where or how the function acts. The common variables used inside the parentheses and a description of what they represent are as follows:

bf Represents a Boolean flag with a value of *TRUE* or *FALSE* to show whether to enable or disable the function. Most functions that use this variable are initially *FALSE*.

ch Represents a character entry.

win Represents the window appointed for the function. When this variable appears, you must specify what window is to receive the action.

y, x Represent the row and column, respectively, for the cursor position.

str Represents a string of characters to be input.

Any other function variable that is encountered will be explained in the associated discussion.

Also, many function names include the main function with a prefix. The prefix is used to show the form of the function used. For instance, **printw** is a standard **curses** print routine. The **wprintw** function is a window print routine. A prefix of *mv* shows a starting position to be given to the routine. Thus, **mvprintw** will begin printing at the appointed cursor position and **mvwprintw** will begin printing at the appointed cursor position inside the appointed window.

Comments

Example programs contained in this guide include comments to help explain the procedures in the program. The comments are contained by the characters "/*" and "*/." Comments are ignored by program executions.

Chapter 2

SCREEN MANIPULATION

PAGE

STRUCTURE 2-1

INITIALIZATION 2-3

- General 2-3

INPUT/OUTPUT FUNCTIONS 2-4

- General 2-4
- Output 2-6
- Input 2-15
- Delays 2-17

TERMINAL SETTINGS 2-18

- Terminal Mode Setting 2-18
- Option Setting 2-21

Chapter 2

SCREEN MANIPULATION

Through **curses**, a terminal screen can be scrolled, cleared, or divided into separate windows. The most basic function of a **curses** program is screen manipulation. The structure of a **curses** program is always the same. The simplicity or complexity of the program function has no influence on the program structure.

STRUCTURE

All programs using **curses** should include the file *curses.h*. This file defines several **curses** functions as macros and defines several global variables and the datatype *WINDOW*. References to windows are always of the type *WINDOW *name-of-window*. Curses also defines *WINDOW ** constants *stdscr* and *curscr*. The *stdscr* (standard screen) constant is used as a default value to routines expecting a window. The *curscr* (current screen) constant is used only for certain low-level operations like clearing and redrawing a garbaged screen. Integer constants, that dictate the size of the screen, for *LINES* and *COLS* are defined. Constants *TRUE* and *FALSE* are defined with values 1 and 0, respectively. Additional constants that are values returned from most **curses** functions are *ERR* and *OK*. The *OK* value is returned if the function could be properly completed, and *ERR* is

returned if there was some error such as moving the cursor outside a window.

The include file *curses.h* automatically includes *stdio.h* and an appropriate tty driver interface file, currently either *termio.h* or *sgtty.h*.

> ***Note:*** The driver interface *sgtty.h* is a tty driver interface used in other versions of the **UNIX*** System.

Including *stdio.h* again is harmless but wasteful; including *termio.h* again will usually result in a fatal error.

A program using **curses** should include the loader option *–lcurses* in the makefile. This is true for both the **terminfo** level and the **curses** level.

---

* Trademark of AT&T

INITIALIZATION

General

The following functions are called when initializing a program.

initscr( )

The first function called should always be **initscr**. This will determine the terminal type and initialize **curses** data structures. The **initscr** function also arranges that the first call to **refresh** will clear the screen.

endwin( )

A program should always call **endwin** before exiting. This function will restore tty modes, move the cursor to the lower left corner of the screen, reset the terminal into the proper nonvisual mode, and tear down all appropriate data structures.

The following is a sample of the smallest possible **curses** program. The only thing this program will do is clear (refresh) the screen.

```
#include    <curses.h>

main( )
{
    initscr( );
    refresh( );
    endwin( );
}
```

INPUT/OUTPUT FUNCTIONS

General

printw(fmt, args)
wprintw(win, fmt, args)
mvprintw(y, x, fmt, args)
mvwprintw(win, y, x, fmt, args)

These functions correspond to **printf**. The characters that would be output by **printf** are instead output using **waddch** on the given window. Never use **printf** in a **curses** program because **curses** must have total control of the terminal.

refresh( )
wrefresh(win)

These functions must be called to get any output on the terminal, since other routines merely manipulate data structures. The **wrefresh** function copies the named window to the physical terminal screen, taking into account what is already there to do optimization. The **refresh** function is the same, using **stdscr** as a default screen. Unless **leaveok** has been enabled, the physical cursor of the terminal is left at the location of the cursor for that window. (Refer to the "Option Setting" heading in the "TERMINAL SETTINGS" section of this chapter for a description of **leaveok**.)

move(y, x)
wmove(win, y, x)

The cursor associated with the window is moved to the given location. This does not move the physical cursor of the terminal until **refresh** is called. The position specified is relative to the upper left corner of the window. Thus, if you have a window that is not in the upper left corner of the screen, you would have to specify the screen coordinates as a distance from the upper left corner of the window. The upper left corner of the window is position **(0, 0)**.

The following sample program shows the use of the **printw** and **move** functions. This program will print "Hello World" in the center of the

screen. The center of the screen is determined as the half-screen point both vertically (LINES / 2) and horizontally (COLS / 2).

```
#include    <curses.h>

main( )
{
    initscr( );
    move( LINES / 2, COLS / 2 );
    printw( "Hello World" );
    refresh( );
    endwin( );
}
```

The output of this program will appear as follows:

```
                    Hello World

$
```

erase( )
werase(win)
These functions copy blanks to every position in the window.

clear( )
wclear(win)
These functions are like **erase** and **werase**, but they also call **clearok**, arranging that the screen will be cleared on the next call to **refresh** for that window.

clrtobot( )
wclrtobot(win)
All lines below the cursor in this window are erased. Also, the current line to the right of the cursor is erased.

clrtoeol( )
wclrtoeol(win)
The current line to the right of the cursor is erased.

The following sample program shows the use of the **clrtobot** function. This program will print a full screen of lines, move the cursor to the fifth line (cursor position 5, 0), and clear all lines below the cursor.

```
#include    <curses.h>

main( )
{
    int    i;

    initscr( );
    for( i=0; i<LINES; i++ )
    printw( "This is line %d of the standard screen.\n", i );
    refresh( );
    sleep(5);
    move( 5, 0 );
    clrtobot( );
    refresh( );
    endwin( );
}
```

Output

A program using **curses** always starts by calling **initscr( )**. Other modes can then be set as needed by the program. Possible modes include **cbreak( )** and **idlok(stdscr, TRUE)**. These modes will be explained later in this section. During the execution of the program, output to the screen is done with routines such as **addch(ch)** and **printw(fmt,args)**. (These routines behave just like **putchar** and **printf** except that they go through

curses.) The cursor can be moved with the call **move(row,col)**. These routines only output to a data structure called a *window*, not to the screen. A window is a representation of a terminal screen containing such things as an array of characters to be displayed on the screen, a cursor, a current set of video attributes, and various modes and options. You don't need to worry about windows unless you use more than one of them, except to realize that a window is buffering your requests to output to the screen.

To determine how to update the screen, **curses** must know what is always on the screen. This requires **curses** to clear the screen in the first call to **refresh** and to always know the cursor position and screen contents. To send all accumulated output, it is necessary to call **refresh( )**. (This can be thought of as a **flush**.) Remember, before the program exits, it should call **endwin( )**, that restores all terminal settings and positions the cursor at the bottom of the screen.

See the program **scatter** in the Appendix for an example program that uses many of the output routines.

No output to the terminal actually happens until **refresh** is called. Instead, routines such as **move** and **addch** draw on a window data structure called **stdscr** (standard screen). **Curses** always keeps track of what is on the physical screen as well as what is in **stdscr**.

When **refresh** is called, **curses** compares the two screen images and sends a stream of characters to the terminal that will turn the current screen into what is desired. **Curses** considers many different ways to do this, taking into account the various capabilities of the terminal and the similarities between what is on the screen and what is desired. **Curses** usually outputs as few characters as is possible. This function is called *cursor optimization*, and it is the source of the name of the **curses** utilities.

> ***Note:*** Owing to the hardware scrolling of some terminals, writing to the lower right-hand character position could be impossible.

Bells and Flashing Lights

utilities( )
flash( )

These functions are used to signal the programmer. The **beep** function will sound the audible alarm on the terminal, if possible; and if not possible, the **beep** function will flash the screen (visible bell), if that is possible. The **flash** function will flash the screen; and if that is not possible, the **flash** function will sound the audible signal. If neither signal is possible, nothing will happen. Nearly all terminals have an audible signal (bell or beep), but only some can flash the screen. In the program **scatter** note the call to **flash( )**, that flashes the screen if possible, and otherwise rings the bell. Flashing the screen is intended as a bell replacement, and flashing is particularly useful if the bell bothers someone within hearing distance of the user's terminal. The routine **beep( )** can be called when a real beep is desired. (If for some reason the terminal is unable to beep but able to flash, a call to **beep** will flash the screen.)

Inserting and Deleting Text

insertln( )
winsertln(win)

A blank line is inserted above the current line. The bottom line is lost. This does not necessarily imply use of the hardware insert line feature.

deleteln( )
wdeleteln(win)

The line under the cursor in the window is deleted. All lines below the current line are moved up one line. The bottom line of the window is cleared. This does not necessarily imply use of the hardware delete line feature.

Writing One Character

addch(ch)
waddch(win, ch)
mvaddch(y, x, ch)
mvwaddch(win, y, x, ch)

The character **ch** is put in the window at the current cursor position of the window. If **ch** is a tab, newline, or backspace the cursor will be moved appropriately in the window. If **ch** is a different control character, it will be drawn in the ^X notation. The position of the window cursor is advanced. At the right margin, an automatic newline is performed. At the bottom of the scrolling region, if **scrollok** is enabled, the scrolling region will be scrolled up one line, and **wrefresh** will be called. If **scrollok** isn't enabled, the cursor will print the next characters on the last line of the window. (Refer to the "Option Setting" heading in the "TERMINAL SETTINGS" section of this chapter for the description of **scrollok**.)

The **ch** parameter is actually an integer, not a character. Video attributes can be combined with a character by OR-ing them into the parameter. This will result in these video attributes also being set, in addition to any current attributes in the window. (The intent here is that text including attributes can be copied from one place to another with **inch** and **addch**.)

Writing a String

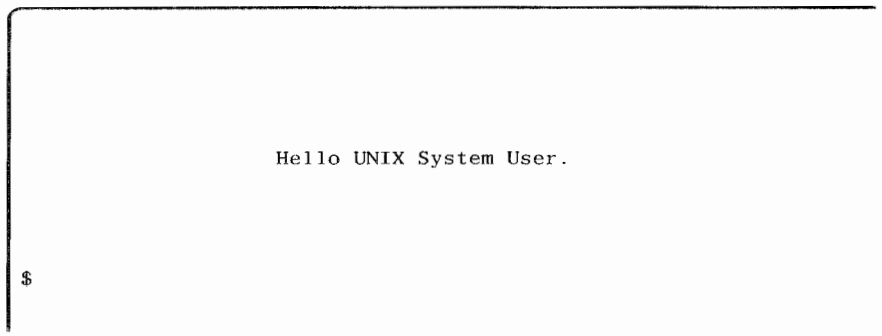
addstr(str)
waddstr(win,str)
mvaddstr(y,x,str)
mvwaddstr(win,y,x,str)

These functions write all the characters of the null terminated character string **str** on the given window. They are identical to a series of calls to **addch**.

The following sample shows the use of the **mvaddstr** function.

```
#include    <curses.h>

main( )
{
    initscr( );
    mvaddstr( LINES / 2, ( COLS / 2 ) - 11,
                "Hello UNIX System User." );
    refresh( );
    endwin( );
}
```

The output for this program would appear as follows:



delch( )
wdelch(win)
mvdelch(y,x)
mvwdelch(win,y,x)

The character under the cursor in the window is deleted. All characters to the right on the same line are moved to the left one position. The rightmost character on the line is set to an unhighlighted blank. This does not necessarily imply use of the hardware delete character feature.

insch(c)
winsch(win, c)
mvinsch(y,x,c)
mvwinsch(win,y,x,c)

The character **c** is inserted before the character under the cursor. All characters to the right are moved one space to the right, possibly losing the rightmost character on the line. This does not necessarily imply use of the hardware insert character feature.

inch( )
winch(win)
mvinch(y,x)
mvwinch(win,y,x)

The character at the current position in the named window is returned. If any attributes are set for that position, their values will be OR-ed in with the value returned. The predefined constants *A_ATTRIBUTES* and *A_CHARTEXT* can be used with the "&" operator to extract the character or attributes alone.

Video Attributes

The function **addch** always draws two things on a window. In addition to the character itself, a set of *attributes* is associated with the character. These attributes cover various forms of highlighting the character. For example, the character can be put in reverse video, bold, or be underlined. You can think of the attributes as the color of the ink used to draw the character.

A window always has a set of *current attributes* associated with it. The current attributes are associated with each character as it is written to the window. Attributes are a property of the character and move with the character through any scrolling and insert/delete line/character operations. To the extent possible on the particular terminal, they will be displayed as the graphic rendition of characters put on the screen. The current attributes can be changed with a call to **attrset(attrs)**. (Think of this as dipping the pen for that window in a particular color ink.)

The names of the attributes are:

A_STANDOUT	A_UNDERLINE
A_REVERSE	A_BOLD
A_BLINK	A_BLANK
A_PROTECT	A_ALTCHARSET
A_DIM	

Not all terminals are capable of displaying all attributes. If a particular terminal cannot display a requested attribute, **curses** will attempt to find a substitute attribute. If none is possible, the attribute is ignored.

One particular attribute is called STANDOUT. This attribute is used to make text attract the attention of the user. The particular hardware attribute used for standout varies from terminal to terminal and is usually the most visually pleasing attribute the terminal can produce. Standout is typically implemented as reverse video or bold. Many programs do not really need a specific attribute, such as bold or inverse video, but instead just need to highlight some text. For such applications, the **A_STANDOUT** attribute is recommended. Two convenient functions, **standout( )** and **standend( )** turn on and off this attribute.

Attributes can be turned on in combination. Thus, to turn on blinking bold text, use **attrset(A_BLINK|A_BOLD)**. Individual attributes can be turned on and off with **attron** and **attroff** without affecting other attributes.

For an example program using attributes, see the **highlight** program in the Appendix. The **highlight** program comes about as close to being a filter as is possible with **curses**. It is not a true filter because **curses** must "take over" the terminal screen.

The following functions set the "current attributes" of the named window. These constants are defined in *curses.h* and can be combined with the C | (or) operator.

attrset(at)
wattrset(win, attrs)
The **attrset(at)** function sets the current attributes of the given window to **at**.

attroff(at)
wattroff(win, attrs)
The **attroff(at)** function turns off the named attributes without affecting any other attributes.

attron(at)
wattron(win, attrs)
The **attron(at)** function turns on the named attributes without affecting any others.

standout( )
standend( )
wstandout(win)
wstandend(win)
The **standout** function is the same as **attron(A_STANDOUT)**. The **standend** function turns off all attributes, the same as **attrset (0)**.

The following sample shows the use of video attributes. Note that this program is expanded from the sample program shown in the "Writing a String" section presented earlier in this chapter.

```
#include    <curses.h>

main( )
{
    initscr( );
    attron( A_STANDOUT );
    mvaddstr( LINES / 2, ( COLS / 2 ) - 11, "Hello " );
    attron( A_BLINK );
    refresh( );
    printw ( "UNIX " );
    refresh( );
    attroff( A_STANDOUT | A_BLINK );
    attron( A_UNDERLINE );
    printw ( "System " );
    refresh( );
    attroff( A_UNDERLINE );
    printw ( "Users" );
    refresh( );
    endwin( );
}
```

If your terminal has all these attribute capabilities, the output will appear as explained below:

- The word "Hello" will be displayed in *inverse-video*.
- The word "UNIX" will be displayed in *inverse-video* and will also blink.
- The word "System" will return to the normal mode of display, but will be underlined.
- The word "Users" will appear as normal text for your terminal.

Input

Curses can do more than just draw on the screen. Functions are also provided for input from the keyboard. The primary function is **getch( )** that waits for the user to type a character on the keyboard and then returns that character. This function is like **getchar** except that it goes through **curses**. Its use is recommended for programs using the **cbreak( )** or **noecho( )** options since several terminal or system dependent options become available that cannot be written portably with **getchar**.

Options available with **getch** include **keypad** that allows extra keys such as arrow keys, function keys, and other special keys that transmit escape sequences to be treated as just another key. The values for these keys are over octal 400; so, they should be stored in a variable larger than a **char**. (See the **curses** manual page in the *AT&T 3B2 Computer Programer Reference Manual* for a list of function keys and their value.) The **nodelay** mode causes the value –1 to be returned if there is no input waiting. (Refer to the "Option Setting" heading in the "TERMINAL SETTINGS" section of this chapter for the description of **nodelay**.) Normally, **getch** will wait until a character is typed. Finally, the routine **getstr(str)** can be called, allowing input of an entire line, up to a newline. This routine handles echoing and the erase and kill characters of the user.

See the program **show** in the Appendix for an example use of **getch**.

getch( )
wgetch(win)
mvgetch(y,x)
mvwgetch(win,y,x)

A character is read from the terminal associated with the window. In **nodelay** mode, if there is no input waiting, the value –1 is returned. In **delay** mode, the program will hang until the system passes text through to the program. Depending on the setting of **cbreak**, this will be after one character or after the first newline.

If **keypad** mode is enabled and a function key is pressed, the code for that function key will be returned instead of the raw characters. Possible function keys are defined with integers beginning with 0401

whose names begin with **KEY_** . (Refer to the **curses** manual page in the *AT&T 3B2 Computer Programer Reference Manual.*) If a character is received that could be the beginning of a function key (such as escape), **curses** will set a 1-second timer. If the remainder of the sequence does not come in within 1 second, the character will be passed through; otherwise, the function key value will be returned. So, on many terminals there will be a one second delay after a user presses the escape key. (Using the escape key for a single character function is discouraged.)

getstr(str)
wgetstr(win,str),
mvgetstr(y,x,str)
mvwgetstr(win,y,x,str)

A series of calls to **getch** is made until a newline is received. The resulting value is placed in the area appointed by the character pointer **str**. The user's erase and kill characters are interpreted.

scanw(fmt, args)
wscanw(win, fmt, args)
mvscanw(y, x, fmt, args)
mvwscanw(win, y, x, fmt, args)

This function corresponds to **scanf**. The **wgetstr** function is called on the window, and the resulting line is used as input for the scan.

Delays

These functions are not considered to be portable, but are often needed by programs, especially real time response programs, that use **curses**. Some of these functions require a particular operating system or a change to the operating system to work. The routine will always compile and return an error status if the requested action is not possible. It is recommended that programmers avoid use of these functions if possible.

draino(0)

The program is suspended until the output queue has drained enough to complete in **0** additional milliseconds. The purpose of this routine is to keep the program (and thus the keyboard) from getting ahead of the screen. If the operating system does not support the ioctls needed to fulfill **draino**, the value ERR is returned; otherwise, OK is returned.

napms(ms)

This function suspends the program for **ms** milliseconds. It is similar to **sleep** except with higher resolution. The resolution actually provided will vary with the facilities available in the operating system, and often a change to the operating system will be necessary to produce good results. The best resolution possible is about .1 seconds. If this resolution is not obtainable, the **napms** routine will round to the next higher second, call **sleep**, and return ERR. Otherwise, the value "OK" is returned.

TERMINAL SETTINGS

Terminal Mode Setting

These functions are used to set modes in the tty driver. The initial mode usually depends on the setting when the program was called:

longname( )
This function returns a pointer to a static area containing a verbose description of the current terminal. It is defined only after a call to **initscr**, **newterm**, or **setupterm**.

The **longname** function does not need any arguments. It returns a pointer to a static area containing the long name of the terminal.

echo( )
noecho( )
These functions control the way characters typed by the user are echoed. Initially, characters typed are echoed by the tty driver. Authors of many interactive programs prefer to do their own echoing in a controlled area of the screen, or they prefer not to echo at all, so they disable echoing.

nl( )
nonl( )
These functions control whether newline is translated into carriage return and linefeed on output, and whether return is translated into newline on input. Initially, the translations do occur. By disabling these translations, **curses** is able to make better use of the linefeed capability, resulting in faster cursor motion. Unless you need to have the RETURN key mapped into NEWLINE, it is recommended that you set **nonl( )**, in addition to **cbreak( )** and **noecho( )**.

scroll(win)
The window is scrolled up one line. This involves moving the lines in the window data structure. As an optimization, if the window is **stdscr** and the scrolling region is the entire window, the physical screen will be scrolled at the same time.

cbreak( )
nocbreak( )

These two functions put the terminal into and out of **cbreak** mode. In this mode, characters typed by the user are immediately available to the program. When out of this mode, the tty driver will buffer characters typed until newline is typed. Interrupt and flow control characters are unaffected by this mode. Initially, the terminal is not in **cbreak** mode. Most interactive programs using **curses** will set this mode.

raw( )
noraw( )

The terminal is placed into or out of raw mode. Raw mode is similar to cbreak mode in that characters typed are immediately passed through to the user program. The differences are that in RAW mode the interrupt and quit characters are passed through uninterpreted instead of generating a signal. RAW mode also causes 8-bit input and output. The behavior of the BREAK key and suspend signal may be different on different systems.

The following sample program combines the **echo** and **noecho** functions with some of the Input/Output functions described earlier in this chapter. The program will not echo (display) what is typed by the user. However, the program will respond with a message according to the character typed. If an "h" is entered, "Hello World" is displayed; a "q" will display "Good Bye" and stop the program. Any other letter entered will display the "Sorry only h ..." message.

```
#include    <curses.h>

main( )
{
    int    g;
    initscr( );
    cbreak( );
    noecho( );
    nonl( );
    refresh( );

    for ( ;; )
    {
        if ( ( g = getch( ) ) != 'q' )
        {
            if( g == 'h' )
                printw( "Hello World.\n" );
            else
                printw( "Sorry only h will be accepted.\n" );
            refresh( );
        }
        else
        {
            printw( "Good Bye.\n" );
            refresh( );
            endwin( );
            exit( 0 );
        }
    }
}
```

Option Setting

General

These functions set options within **curses**.

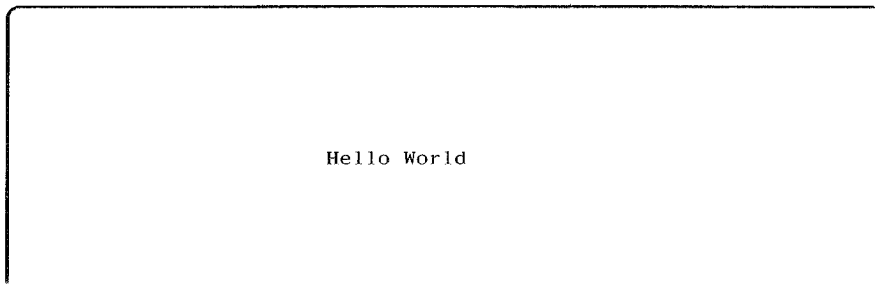
clearok(win,bf)

If set, the next call to **wrefresh** with this window will clear the screen and redraw the entire screen. If **win** is **curscr**, the next call to **wrefresh** with any window will cause the screen to be cleared. This is useful when the contents of the screen are uncertain, or sometimes for a more pleasing visual effect.

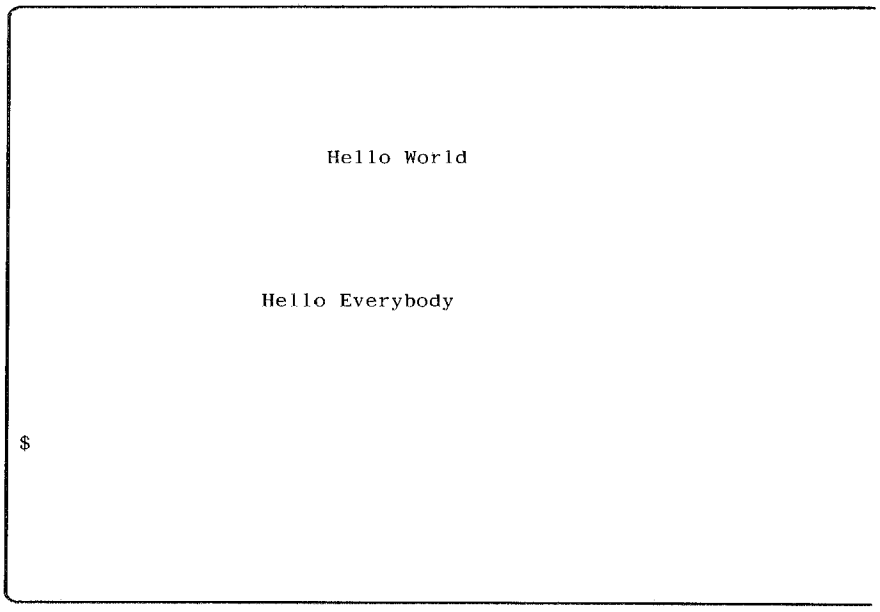
An example of the **clearok** function is shown in the following sample program.

```
#include    <curses.h>

main( )
{
    initscr( );

    move( LINES / 2, ( COLS / 2 ) - 5 );
    printw( "Hello World" );
    refresh( );
    sleep( 1 );
    clearok( stdscr, 1 );
    move( ( LINES / 2 ) + 5, COLS / 3 );
    printw( "Hello Everybody" );
    wrefresh( stdscr );
    endwin( );
}
```

The output from this program will appear in two separate displays. The first display will appear as follows:



The screen will then be cleared and redrawn with the additional message "Hello Everybody." The resulting display will appear as follows:



idlok(win,bf)

If enabled, **curses** will consider using the hardware insert/delete line feature of terminals so equipped. If disabled, **curses** will never use this feature. The insert/delete character feature is always considered. Enable this option only if your application needs the insert/delete line, for example, for a screen editor, or for scrolling. The insert/delete feature is disabled by default because the insert/delete line tends to be visually annoying when used in applications where it isn't really needed. If the insert/delete line cannot be used, **curses** will redraw the changed portions of all lines that do not match the desired line.

keypad(win,bf)

This option enables the keypad of the users terminal. If enabled, the user can press a function key (such as an arrow key), and **getch** will return a single value representing the function key. If disabled, **curses** will not treat function keys specially. If the keypad in the terminal can be turned on (made to transmit) and off (made to work locally), turning on this option will turn on the terminal keypad.

scrollok(win,bf)

This option controls what happens when the cursor of a window is moved off the edge of the window either from a newline on the bottom line or typing the last character of the last line. If disabled, the cursor is left on the bottom line. If enabled, **wrefresh** is called on the window, and then the physical terminal and window are scrolled up one line. Note that to get the physical scrolling effect on the terminal, it is also necessary to call **idlok**.

Advanced

leaveok(win,bf)

Normally, the hardware cursor is left at the location of the window cursor being refreshed. The **leaveok** option allows the cursor to be left wherever the update happens to leave it. The **leaveok** option is useful for applications where the cursor is not used, since it reduces the need for cursor motions. If possible, the cursor is made invisible when this option is enabled.

nodelay(win,bf)
This option causes **getch** to be a nonblocking call. If no input is ready, **getch** will return –1. If disabled, **getch** will hang until a key is pressed.

unctrl(*ch*)
The **unctrl(*ch*)** function is actually a macro that makes a printable representation of the character *ch*. Control characters are displayed in the "^X" notation and printing characters are displayed as is.

Expert

meta(win,bf)
If enabled, characters returned by **getch** are transmitted with all 8 bits instead of stripping the highest bit. The value **OK** is returned if the request succeeded, the value **ERR** is returned if the terminal or system is not capable of 8-bit input.

The meta mode is useful for extending the non-text command set in applications where the terminal has a meta shift key. Curses takes whatever measures are necessary to arrange for 8-bit input. On some other versions of UNIX Systems, the raw mode will be used. On the 3B2 Computer, the character size will be set to 8, parity checking disabled, and stripping of the 8th bit turned off.

Note that 8-bit input is a fragile mode. Many programs and networks only pass 7 bits. If any link in the chain from the terminal to the application program strips the 8th bit, 8-bit input is impossible.

intrflush(win,bf)
If this option is enabled when an interrupt key is pressed on the keyboard (interrupt, quit, suspend), all output in the tty driver queue will be flushed, giving the effect of faster response to the interrupt but causing **curses** to have the wrong idea of what is on the screen. Disabling the option prevents the flush. The default is for the option to be enabled. This option depends on support in the underlying tty driver.

typeahead(fd)

Sets the file descriptor for typeahead check. The **fd** variable should be an integer returned from **open** or **fileno**. Setting typeahead to the default value of –1 will disable typeahead check. By default, file descriptor 0 (stdin) is used. The typeahead is checked independently for each screen, and for multiple interactive terminals it should probably be set to the appropriate input for each screen. A call to **typeahead** always affects only the current screen.

setscrreg(t,b)
wsetscrreg(win,t,b)

These functions allow the user to set a software scrolling region in a window **win** or **stdscr**. The **t** and **b** variables are the line numbers of the top and bottom margin of the scrolling region. (Line 0 is the top line of the window.) If this option and **scrollok** are enabled, an attempt to move off the bottom margin line (**addch** a newline) will cause all lines in the scrolling region to scroll up one line. Note that this has nothing to do with use of a physical scrolling region capability in the terminal. Only the text of the window is scrolled. If **idlok** is enabled and the terminal has either a scrolling region or insert/delete line capability, they will probably be used by the output routines.

The following sample program shows the usage of some of the more complicated "Option Setting" functions. This program is designed to display several lines separated by a scrolling region. The program will then display lines inside the scrolling region and show the scrolling action as lines are added to the bottom of the region.

```
#include    <curses.h>

main( )
{
    int    i, t, b;

    initscr( );
    t = LINES/3;
    b = ( LINES / 3 ) * 2;
    scrollok( stdscr, TRUE );
    idlok( stdscr, TRUE );
    setscrreg( t, b );
    mvaddstr( t-1, 0, "THIS IS LINE 1 OUTSIDE THE SCROLLING REGION" );
    mvaddstr( b+1, 0, "THIS IS LINE 2 OUTSIDE THE SCROLLING REGION" );
    mvaddstr( b+2, 0, "THIS IS LINE 3 OUTSIDE THE SCROLLING REGION" );
    mvaddstr( b+3, 0, "THIS IS LINE 4 OUTSIDE THE SCROLLING REGION" );
    mvaddstr( b+4, 0, "THIS IS LINE 5 OUTSIDE THE SCROLLING REGION" );
    mvaddstr( b+5, 0, "THIS IS LINE 6 OUTSIDE THE SCROLLING REGION" );
    mvaddstr( b+6, 0, "THIS IS LINE 7 OUTSIDE THE SCROLLING REGION" );
    mvaddstr( b+7, 0, "THIS IS LINE 8 OUTSIDE THE SCROLLING REGION" );
    refresh( );
    move( t, 0 );
    for( i = t; i < b+3; i++ )
    {
        printw( "This is line %d of the standard screen.\n", i );
        sleep( 2 );
        refresh( );
    }
    endwin( );
}
```

Chapter 3

WINDOW MANIPULATION

PAGE

INITIALIZATION 3-1
General 3-1
Multiple Windows 3-6
INPUT/OUTPUT FUNCTIONS 3-8
General 3-8
Input 3-11
PAD MANIPULATION 3-12

Chapter 3

WINDOW MANIPULATION

INITIALIZATION

General

These functions are some of the more common window manipulation routines. These functions are used to build, move, and delete **curses** windows.

newwin(num_lines, num_cols, beg_row, beg_col)

Create a new window with the given amount of lines and columns. The upper left corner of the window is at line **beg_row**, column **beg_col**. If either **num_lines** or **num_cols** is zero, they will be defaulted to **LINES-beg_row** and **COLS-beg_col**. A new full screen window is created by calling **newwin(0,0,0,0)**.

delwin(win)

Deletes the named window, freeing up all memory associated with it. If there are overlapping windows, subwindows should be deleted before the main window.

touchwin(win)

Throw away all optimization information about what parts of the window have been touched by pretending the entire window has been drawn on. This is sometimes necessary when using overlapping windows, since a change to one window will affect the other window; but the records of what lines have been changed in the other window will not reflect the change.

mvwin(win, br, bc)

Move the window so that the upper left corner will be at position **(br, bc)**. If the move would cause the window to be off the screen, it is an error and the window is not moved.

subwin(orig, num_lines, num_cols, begy, begx)

Create a new window with the given amount of lines and columns. The window is at position *begy, begx* on the screen. (It is relative to the screen, not **orig**.) The window is made in the middle of the window **orig**, so that changes made to one window will affect both windows. When using this function, often it will be necessary to call **touchwin** before calling **wrefresh**.

overlay(win1, win2)
overwrite(win1, win2)

These functions overlay **win1** on top of **win2**; that is, all text in **win1** is copied into **win2**. The difference is that **overlay** is nondestructive (blanks are not copied). This means that the bottom window will show through the blank spaces of the top window. The **overwrite** is destructive; that is, the top window will completely cover the bottom window.

The following sample program shows how the **overwrite** function can be used.

```
#include    <curses.h>

main( )
{
    int    j;
    WINDOW    *win1, *win2;

    initscr( );
    win1 = newwin( 10, 8, 0, 0 );
    win2 = newwin( 10, 8, 0, 0 );
    for( j=0; j<10; j++ )
        wprintw( win1, "1111111\n" );
    for( j=0; j<10; j++ )
        wprintw( win2, "2 2 2 2\n" );
    wrefresh( win1 );
    wgetch(win1);
    overwrite( win2, win1 );
    wrefresh( win1 );
    endwin( );
}
```

The output from this program will appear with two separate windows. The first window will appear as follows:

```
1111111
1111111
1111111
1111111
1111111
1111111
1111111
1111111
1111111
1111111
```

The second window will appear when a character is entered from the keyboard. This window will appear as follows:

```
2 2 2 2
2 2 2 2
2 2 2 2
2 2 2 2
2 2 2 2
2 2 2 2
2 2 2 2
2 2 2 2
2 2 2 2
2 2 2 2


$
```

If this program had been written to show the **overlay** function, the second window would appear as follows:

```
2121212
2121212
2121212
2121212
2121212
2121212
2121212
2121212
2121212
2121212


$
```

Note: Notice that the "1's" show through the blank spaces between the "2's" in the output from the **overlay** function.

box(win, vert, hor)

A box is drawn around the edge of the window. The **vert** and **hor** variables are the characters the box is to be drawn with. If **vert** or **hor** have the value of zero, **curses** will substitute reasonable characters for the zero.

refresh( )
wrefresh(win)

This function must be called to get any output on the terminal, as other routines merely manipulate data structures. The **wrefresh** function copies the named window to the physical terminal screen, taking into account what is already there to do optimizations. The **refresh** function is the same, using **stdscr** as a default screen. Unless **leaveok** has been enabled, the physical cursor of the terminal is left at the location of the window cursor.

doupdate( )
wnoutrefresh(win)

These two functions allow multiple updates with more efficiency than **wrefresh**. To use them, it is important to understand how **curses** works. In addition to all the window structures, **curses** keeps two data structures representing the terminal screen: a *physical* screen, describing what is actually on the screen, and a *virtual* screen, describing what the programmer *wants* to have on the screen. The **wrefresh** function works by first copying the named window to the virtual screen **(wnoutrefresh)** and then calling the routine to update the screen **(doupdate)**. If the programmer wishes to output several windows at once, a series of calls to **wrefresh** will result in alternating calls to **wnoutrefresh** and **doupdate**, causing several bursts of output to the screen. By calling **wnoutrefresh** for each window, it is then possible to call **doupdate** once, resulting in only one burst of output, with probably fewer total characters transmitted.

Multiple Windows

A window is a data structure representing all or part of the terminal screen. It has room for a two dimensional array of characters, attributes for each character (a total of 16 bits per character: 7 for text and 9 for attributes), a cursor, a set of current attributes, and many flags. Curses provides a full screen window, called **stdscr**, and a set of functions that use **stdscr**. Another window is provided called **curscr**, representing the physical screen.

It is important to understand that a window is only a data structure. Use of more than one window neither implies use of more than one terminal nor involves more than one process. A window is merely an object that can be copied to all or part of the terminal screen.

The programmer can create additional windows with the
newwin(lines, cols, begin_row, begin_col)
function. This function will return a pointer to a newly created window. The window will be **lines** long by **cols** wide, and the upper left corner of the window will be at screen position **(begin_row, begin_col)**. All operations that affect **stdscr** have corresponding functions that affect an arbitrary named window. Generally, these functions have names formed by putting a "w" on the front of the **stdscr** function, and the window name is added as the first parameter. Thus, **waddch(mywin, c)** would write the character **c** to window **mywin**. The **wrefresh(win)** function is used to flush the contents of a window to the screen.

The following sample program shows how to use several of the window manipulation functions. This program will create four different windows on the screen.

```
#include <curses.h>

main( )
{
    WINDOW *win1, *win2, *win3, *win4;

    initscr( );
    win1 = newwin(LINES/3,COLS,0,0);
    win2 = newwin(LINES/3,COLS,LINES/3,0);
    win3 = newwin(LINES/3,COLS,(LINES/3)*2);
    win4 = newwin(LINES/3,COLS/2,(LINES/3)*2);
    box(win1,'1','1');
    box(win2,'2','2');
    box(win3,'3','3');
    box(win4,'4','4');
    wrefresh(win1);
    wrefresh(win2);
    wrefresh(win3);
    wrefresh(win4);
    endwin( );
    exit(0);
}
```

Windows are useful for maintaining several different screen images, and alternating the user among them. Also, it is possible to subdivide the screen into several windows, refreshing each of them as desired. When windows overlap, the contents of the screen will be the more recently refreshed window. The program **window** in the Appendix is another example of the use of multiple windows.

In all cases, the non-w version of the function calls the w version of the function, using **stdscr** as the additional argument. Thus, a call to **addch(c)** results in a call to **waddch(stdscr, c)**.

For convenience, a set of "move" functions are also provided for most of the common functions. These result in a call to **move** before the other function. For example, **mvaddch(row, col, c)** is the same as **move(row, col); addch(c)**. Combinations like **mvwaddch(row, col, win, c)** also exist.

INPUT/OUTPUT FUNCTIONS

General

The Input/Output functions of "windows" is similar to the Input/Output functions of "standard" screens (stdscr). The only real difference is that you must also specify the window to receive the action.

wprintw(win, fmt, args)
mvwprintw(win, y, x, fmt, args)
These functions correspond to **printf**. The characters that would be output by **printf** are instead output using **waddch** on the given window.

wmove(win, y, x)
The cursor associated with the window is moved to the given location. This does not move the physical cursor of the terminal until **refresh** is called. The position specified is relative to the window. Thus, if you have a window that is not in the upper left corner of the screen, you would have to specify the coordinates as the distance from the upper left corner of the window. The upper left corner of the window is position (**0, 0**).

werase(win)
This function copies blanks to every position in the window.

wclear(win)
This function is like **werase** but it also calls **clearok**, arranging that the screen will be cleared on the next call to **refresh** for that window.

wclrtobot(win)
All lines below the cursor in this window are erased. Also, the current line to the right of the cursor is erased.

wclrtoeol(win)
The current line to the right of the cursor is erased.

Inserting and Deleting Text

winsertln(win)
A blank line is inserted above the current line. The bottom line is lost. This does not necessarily imply use of the hardware insert line feature.

wdeleteln(win)
The line under the cursor in the window is deleted. All lines below the current line are moved up one line. The bottom line of the window is cleared. This does not necessarily imply use of the hardware delete line feature.

Writing One Character

waddch(win, ch)
mvwaddch(win, y, x, ch)
The character **ch** is put in the window at the current cursor position of the window. If **ch** is a tab, newline, or backspace, the cursor will be moved appropriately in the window. If **ch** is a different control character, it will be drawn in the ^X notation. The position of the window cursor is advanced. At the right margin, an automatic newline is performed. At the bottom of the scrolling region, if **scrollok** is enabled, the scrolling region will be scrolled up one line.

The **ch** parameter is actually an integer, not a character. Video attributes can be combined with a character by OR-ing them into the parameter. This will result in these attributes also being set. (The intent here is that text, including attributes, can be copied from one place to another with **inch** and **addch**.)

Writing a String

waddstr(win,str)
mvwaddstr(win,y,x,str)

These functions write all the characters of the null terminated character string **str** on the given window. They are identical to a series of calls to **addch**.

wdelch(win)
mvwdelch(win,y,x)

The character under the cursor in the window is deleted. All characters to the right on the same line are moved to the left one position. This does not imply use of the hardware delete character feature.

winsch(win, c)
mvwinsch(win,y,x,c)

The character **c** is inserted before the character under the cursor. All characters to the right are moved one space to the right, possibly losing the rightmost character on the line. This does not imply use of the hardware insert character feature.

winch(win)
mvwinch(win,y,x)

The character at the current position in the named window is returned. If any attributes are set for that position, their values will be OR-ed into the value returned. The predefined constants **A_ATTRIBUTES** and **A_CHARTEXT** can be used with the "&" operator to extract the character or attributes alone.

Video Attributes

The video attributes operate inside windows the same as they operate inside screens. The current attribute of the window applies to all characters written into the window and stay with each character through any movement of the character.

wattrset(win, attrs)
The **wattrset(win, at)** function sets the current attributes of the given window to **at**.

wattroff(win, attrs)
The **wattroff(win, at)** function turns off the named attributes (**at**) without affecting any other attributes.

wattron(win, attrs)
The **wattron(win, at)** function turns on the named attributes without affecting any others.

wstandout(win)
wstandend(win)
The **wstandout** function is the same as **wattron(A_STANDOUT)**. The **wstandend** function is the same as **wattrset(0)**; that is, it turns off all attributes.

Input

wgetch(win)
mvwgetch(win,y,x)
A character is read from the terminal associated with the window. In nodelay mode, if there is no input waiting, the value –1 is returned. In delay mode, the program will hang until the system passes text through to the program. Depending on the setting of **cbreak**, this will be after one character or after the first newline.

Function keys are handled the same in windows as they are in screens. Again, the use of the escape key as a single character function is discouraged.

wgetstr(win,str)
mvwgetstr(win,y,x,str)
A series of calls to **getch** is made until a newline is received. The resulting value is placed in the area pointed at by the character pointer **str**. The user's erase and kill characters are interpreted.

wscanw(win, fmt, args)
mvwscanw(win, y, x, fmt, args)
This function corresponds to **scanf**. The **wgetstr** function is called on the window, and the resulting line is used as input for the scan.

getyx(win,y,x)
The cursor position of the window is placed in the two integer variables **y** and **x**. Since this is a macro, no "&" is necessary.

PAD MANIPULATION

A pad is like a window, except that it is not restricted by the screen size, and a pad is not associated with a particular part of the screen. Pads can be used when a large window is needed and only a part of the window will be on the screen at one time.

newpad(num_lines, num_cols)
Creates a new *pad* data structure. Automatic refreshes of pads (for example, scrolling or echoing of input) do not occur. It is not legal to call **refresh** with a pad as an argument; the routines **prefresh** or **pnoutrefresh** should be called instead.

Note that the following routines require additional parameters to specify the part of the pad to be displayed and the location on the screen to be used for display. The **prefresh** or **pnoutrefresh** function should be called instead.

prefresh(pad,pminrow,pmincol,sminrow,smincol,smaxrow,smaxcol)
pnoutrefresh(pad,pminrow,pmincol,sminrow,smincol,smaxrow,smaxcol)

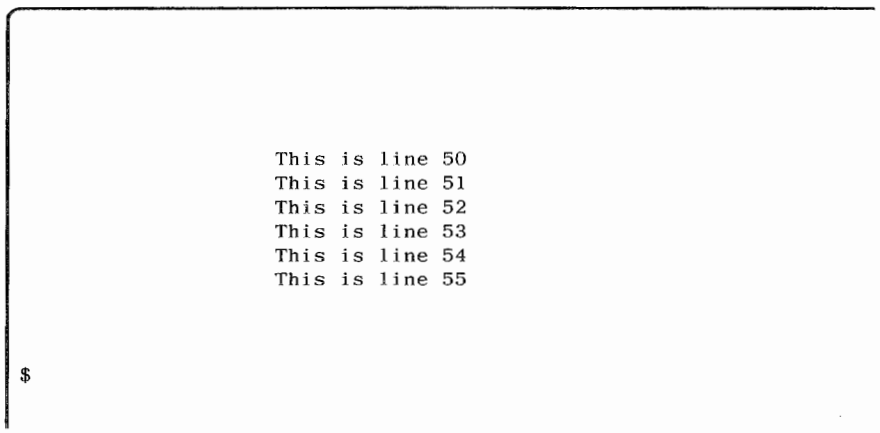
These routines are analogous to **wrefresh** and **wnoutrefresh** except that pads are involved instead of windows. The additional parameters are needed to show what part of the pad and screen are involved. The **pminrow** and **pmincol** variables specify the upper left corner in the pad of the rectangle to be displayed. The **sminrow**, **smincol**, **smaxrow**, and **smaxcol** variables specify the screen coordinates that define the boundaries (edges) of the rectangle to be displayed. The lower right corner in the pad of the rectangle to be displayed is calculated from the screen coordinates, since the rectangles must be the same size. Both rectangles must be entirely contained within their respective structures.

The pad functions are similar to the window functions. The **doupdate** function is used to update the screen like it is used with window structures. The **delwin** function is used to delete a specific pad.

The following sample program shows how to use some of the "pad" functions. This program will output a pad defined by the maximum and minimum cursor positions given on the **prefresh** execution line.

```
#include    <curses.h>

main( )
{
    int    i;

    WINDOW    *pad;

    initscr( );
    pad = newpad( 100, 80 );
    for( i=0; i<99; i++ )
        wprintw( pad, "This is line %d of the pad\n", i );
    prefresh( pad, 50, 0, 10, 20, 15, 35 );
    endwin( );
}
```

The output of this program would appear as follows:



Note: This output may not be shown to scale. The output on your terminal will begin 10 lines down and 20 lines from the left margin.

Chapter 4

MULTIPLE TERMINALS

PAGE

GENERAL PROGRAM FORMAT 4-3
FUNCTIONS 4-4

Chapter 4

MULTIPLE TERMINALS

Curses can produce output on more than one terminal at once. This is useful for single process programs that access a common database such as multi-player games. Output to multiple terminals is a difficult business, and **curses** does not solve all the problems for the programmer. It is the responsibility of the program to determine the file name of each terminal line and what terminal is on each of those lines. The standard method, checking **$TERM** in the environment, does not work since each process can only examine its own environment.

Another problem that must be solved is that of multiple programs reading from one line. This produces a race condition and should be avoided. Nonetheless, a program wishing to take over another terminal cannot just shut off whatever program is currently running on that line. (Usually, security reasons would also make this inappropriate. However, for some applications such as an inter-terminal communication program or a program that takes over unused tty lines, shutting off other programs would be appropriate.) A typical solution requires the user logged in on each line to run a program notifying the master program that the user is interested in joining the master program and telling it the notification program process ID, the name of the tty line, and the type of terminal being used. Then, the program goes to sleep until the master program

finishes. When done, the master program wakes up the notification program, and all programs exit.

Curses handles multiple terminals by always having a "current terminal." All function calls always affect the current terminal. The master program should set up each terminal, saving a reference to the terminals in its own variables. When the master program wishes to affect a terminal, it should set the current terminal as desired, and then call ordinary **curses** routines.

References to terminals are of the type **struct screen ***. A new terminal is initialized by calling **newterm(type, outfd, infd)**. The **newterm** function returns a screen reference to the terminal being set up. The **type** variable represents a character string, naming the type of terminal being used. The **outfd** and **infd** variables are *stdio* file descriptors to be used for input and output to the terminal. (If only output is needed, the file can be opened for output only.) This call replaces the normal call to **initscr**, that calls **newterm(getenv("TERM"), stdout, stdin)**.

To change the current terminal, call "**set_term(sp)**" where **sp** is the screen reference to be made current. The **set_term** function returns a reference to the previous terminal.

GENERAL PROGRAM FORMAT

It is important to realize that each terminal has its own set of windows and options. Each terminal must be initialized separately with **newterm**. Options such as **cbreak** and **noecho** must be set separately for each terminal. The functions **endwin** and **refresh** must be called separately for each terminal. The following sample program is a typical scenario to output a message to each terminal.

```
for (i=0; i<nterm; i++) {
        set_term(terms[i]);
        mvaddstr(0, 0, "Important message");
        refresh();
}
```

See the sample program **two** in the Appendix for a full example. This program pages through a file, showing one page to the first terminal and the next page to the second terminal. Since no standard multiplexor is available in current versions of the UNIX System, it is necessary to either busy wait or call **sleep(1);** between each check for keyboard input. The **two** program sleeps for a second between checks.

The **two** program is just a simple example of two terminal **curses**. It does not handle notification, as described above, instead it requires the name and type of the second terminal on the command line. As written, the command **sleep 100000** must be typed on the second terminal to put it to sleep while the program runs, and the first user must have both read and write permission on the second terminal.

FUNCTIONS

resetty( )
savetty( )

These functions save and restore the state of the tty modes. The **savetty** function saves the current state in a buffer, **resetty** restores the state to what it was at the last call to **savetty**.

newterm(type, fd)

A program that outputs to more than one terminal should use **newterm** instead of **initscr**. The **newterm** function should be called once for each terminal. It returns a variable of type **SCREEN** * that should be saved as a reference to that terminal. The arguments are the type of the terminal (a string) and a stdio file descriptor **(FILE** *) for output to the terminal. The file descriptor should be open for both reading and writing if input from the terminal is desired. The program should also call **endwin** for each terminal being used (see **set_term** below). If an error occurs, the value **NULL** is returned.

set_term(new)

This function is used to switch to a different terminal. The screen reference **new** becomes the new current terminal. The previous terminal is returned by the function. All other calls affect only the current terminal.

Chapter 5

PORTABILITY FUNCTIONS

PAGE

FUNCTIONS 5-1

Chapter 5

PORTABILITY FUNCTIONS

These functions do not directly involve terminal dependent character output but tend to be needed by programs that use **curses**. Unfortunately, their implementation varies from one version of the UNIX System to another.

FUNCTIONS

The following functions have been included here to enhance the portability of programs using **curses**.

erasechar( )
: The erase character chosen by the user is returned to the program. This is the character typed by the user to erase the character just typed.

killchar( )
: The line kill character chosen by the user is returned to the program. This is the character typed by the user to forget the entire line being typed.

flushinp( )

The **flushinp( )** instruction throws away any typeahead that has been typed by the user and has not yet been read by the program.

baudrate( )

The **baudrate( )** instruction returns the output speed of the terminal. The number returned is the integer baud rate, for example, 9600 rather than a table index such as B9600. This function can also be used to check the status of a terminal. If a "0" is returned, the terminal being checked is "off-line." This may suggest a reset (**resetty**) for that terminal if the terminal is supposed to be operating, or it may mean the terminal has been turned OFF.

Chapter 6

LOWER LEVEL FUNCTIONS

PAGE

LOW LEVEL TERMINFO USAGE 6-1
Cursor Motion 6-4
Terminfo Level 6-4

Chapter 6

LOWER LEVEL FUNCTIONS

LOW LEVEL TERMINFO USAGE

Some programs need to use lower level primitives than those offered by **curses**. For such programs, the *terminfo level* interface is offered. This interface does not manage your terminal screen, but rather gives you access to strings and capabilities that you can use yourself to manipulate the terminal.

Programmers are discouraged from using this level. Whenever possible, the higher level **curses** routines should be used. This will make your program more portable to other UNIX Systems and to a wider class of terminals. **Curses** takes care of all the glitches and misfeatures present **in** physical terminals; but at the terminfo level, you must deal with them yourself. Also, you cannot be guaranteed that this part of the interface will not change or be upward compatible with previous releases.

There are two circumstances when it is proper to use terminfo. The first circumstance; you are writing a special purpose tool that sends a special purpose string to the terminal, such as programming a function key, setting tab stops, sending output to a printer port, or dealing with the status line. The second circumstance; you are writing a filter. A typical

filter does one transformation on the input stream without clearing the screen or addressing the cursor. If this transformation is terminal dependent and clearing the screen is inappropriate, use of terminfo is indicated.

The following is a typical format for a program written at the terminfo level.

```
#include <curses.h>
#include <term.h>
...
        setupterm(0, 1, 0);
        ...
        putp(clear_screen);
        ...
        reset_shell_mode( );
        exit(0);
```

Initialization is done by calling **setupterm**. Passing the values 0, 1, and 0 invokes reasonable defaults. If **setupterm** cannot figure out what type of terminal you are on, it will print an error message and exit. The program should call **reset_shell_mode** before it exits.

Global variables with names like **clear_screen** and **cursor_address** are defined by the call to **setupterm**. (See the **terminfo** manual page in the *AT&T 3B2 Computer Programer Reference Manual* for a complete list of capabilities.) They can be output using **putp** or **tputs** that allows the programmer more control. These strings should not be directly output to the terminal using **printf** since they contain padding information. A program that directly outputs strings will fail on terminals that require padding or that use the xon/xoff flow control protocol.

In the terminfo level, the higher level routines described previously are not available. It is up to the programmer to output whatever is needed.

The example program **termhl** in the Appendix shows a simple use of terminfo. It is a version of **highlight** that uses **terminfo** instead of **curses**. This version can be used as a filter. The strings to enter bold and underline mode, and to turn off all attributes, are used.

The **termhl** program is more complex than it need be to illustrate some properties of terminfo. The routine **vidattr** could have been used instead of directly outputting **enter_bold_mode**, **enter_underline_mode**, and **exit_attribute_mode**. The program would be more robust if it used **vidattr** since there are several ways to change video attribute modes. This program was written to illustrate typical use of terminfo.

The function **tputs(cap, affcnt,outc)** applies padding information. Some capabilities contain strings like $<20>, that means to pad for 20 milliseconds. The **tputs** command generates enough pad characters to delay for the appropriate time. The first parameter is the string capability to be output. The second is the number of lines affected by the capability. (Some capabilities may require padding that depends on the number of lines affected. For example, **insert_line** may have to copy all lines below the current line and may require time proportional to the number of lines copied. By convention, **affcnt** is the value 1 rather than 0 if no lines are affected. The value 1 is used for safety reasons, since **affcnt** is multiplied by the amount of time per item, and anything multiplied by 0 is 0.) The third parameter is a routine to be called with each character.

For many simple programs, **affcnt** is always 1 and **outc** always just calls **putchar**. For these programs, the routine **putp(cap)** is a convenient abbreviation. The **termhl** example could be simplified by using **putp**.

Note also the special check for the **underline_char** capability. Some terminals, rather than having a code to start underlining and a code to stop underlining, have a code to underline the current character. The **termhl** program keeps track of the current mode; and if the current character is supposed to be underlined, the program will output

underline_char if necessary. Low level details such as this are precisely why the **curses** level is recommended over the **terminfo** level. Curses takes care of terminals with different methods of underlining and other terminal functions.

Cursor Motion

mvcur(oldrow, oldcol, newrow, newcol)

This routine optimally moves the cursor from (oldrow, oldcol) to (newrow, newcol). The user program is expected to keep track of the current cursor position. Note that unless a full screen image is kept, **curses** will have to make pessimistic assumptions, sometimes resulting in less than optimal cursor motion. For example, moving the cursor a few spaces to the right can be done by transmitting the characters being moved over; but if **curses** does not have access to the screen image, it doesn't know what these characters are.

Terminfo Level

These routines are called by low level programs that need access to specific capabilities of **terminfo**. A program working at this level should include both *curses.h* and *term.h*, in that order. After a call to **setupterm**, the capabilities will be available with macro names defined in *term.h*.

Boolean valued capabilities will have the value 1 if the capability is present — the value 0 if it is not. Numeric capabilities have the value –1 if the capability is missing and have a value of at least 0 if it is present. String capabilities (both those with and without parameters) have the value **NULL** if the capability is missing; otherwise, they have type **char *** and point to a character string containing the capability. The special character codes involving the \ and ^ characters (such as \r for return, or ^A for control A) are translated into the appropriate American Standard Code for Information Interchange (ASCII) characters. Padding information (of the form $<time>) and parameter information (beginning with %) are left uninterpreted at this stage. The routine **tputs** interprets padding information, and **tparm** interprets parameter information.

If the program only needs to handle one terminal, the definition **–DSINGLE** can be passed to the C Language compiler resulting in static references to capabilities instead of dynamic references. This can result in smaller program code, but it prevents use of more than one terminal at a time. Few programs use more than one terminal; so, almost all programs can use this flag.

setupterm(term, filenum, errret)

This routine is called to initialize a terminal. The **term** variable represents the character string representing the name of the terminal being used. The **filenum** variable is the UNIX System file descriptor of the terminal being used for output. The **errret** variable is a pointer to an integer that a success or failure indication is returned. The values returned can be 1 (all is well), 0 (no such terminal), or –1 (some problem locating the **terminfo** database).

The value of **term** can be given as 0, that will cause the value of TERM in the environment to be used. The **errret** pointer can also be given as 0, meaning no error code is wanted. If **errret** is defaulted and something goes wrong, **setupterm** will print an appropriate error message and exit rather than returning. Thus, a simple program can call **setupterm(0, 1, 0)** and not worry about initialization errors.

If the variable TERMINFO is set in the environment to a path name, **setupterm** will check for a compiled **terminfo** description of the terminal under that path before checking */usr/lib/terminfo/*/**. Otherwise, only */usr/lib/terminfo/*/** is checked.

The **setupterm** function will check the tty driver mode bits, using **filenum**, and change any that might prevent the correct operation of other low-level routines. Currently, the mode that expands tabs into spaces is disabled because the tab character is sometimes used for different functions by different terminals. (Some terminals use it to move right one space. Others use it to address the cursor to row or column 9.) If the system is expanding tabs, **setupterm** will remove the definition of the **tab** and **backtab** functions, making the assumption that since the user is not using

hardware tabs, they may not be properly set in the terminal. Other system dependent changes, such as disabling a virtual terminal driver, may be made here.

As a side effect, **setupterm** initializes the global variable **ttytype**, that is an array of characters, to the value of the list of names for the terminal. This list comes from the beginning of the **terminfo** description.

After the call to **setupterm**, the global variable **cur_term** is set to point to the current structure of terminal capabilities. By calling **setupterm** for each terminal and saving and restoring **cur_term**, it is possible for a program to use two or more terminals at once.

The mode that turns newlines into Carriage Return Line Feed (*CRLF*) on output is not disabled. Programs that use **cursor_down** or **scroll_forward** should disable this mode if their value is linefeed. The **setupterm** function calls **reset_prog_mode** after any changes it makes.

reset_prog_mode( )
reset_shell_mode( )
def_prog_mode( )
def_shell_mode( )

These routines can be used to change the tty modes between the two states: *shell* (the mode they were in before the program was started) and *program* (the mode needed by the program). The **def_prog_mode** function saves the current terminal mode as program mode. The **setupterm** function and **initscr** call **def_shell_mode** automatically. The **reset_prog_mode** function puts the terminal into program mode, and **reset_shell_mode** puts the terminal into normal mode. These functions set the tty driver only, they do not transmit anything to the terminal.

A typical calling sequence is for a program to call **initscr** (or **setupterm** if a **terminfo** level program), then to set the desired program mode by calling routines such as **cbreak** and **noecho**, then to call **def_prog_mode** to save the current state. Before a shell escape or control-Z suspension, the program should call **reset_shell_mode** to restore normal mode for the

shell. Then, when the program resumes, it should call **reset_prog_mode**. Also, all programs must call **reset_shell_mode** before they exit. (The higher level routine **endwin** automatically calls **reset_shell_mode**.)

Normal mode is stored in **cur_term->Ottyb**, and program mode is stored in **cur_term->Nttyb**. These structures are both of type **SGTTYB** (that varies depending on the system). Currently, the possible types are **struct sgttyb** (on some other systems) and **struct termio** (on this version of the UNIX System). The **def_prog_mode** function should be called to save the current state in **Nttyb**.

vidputs(newmode, putc)
The **newmode** variable is any combination of attributes, defined in *curses.h*. The **putc** variable is a "putchar-like" function. The proper string to put the terminal in the given video mode is output. The previous mode is remembered by this routine. The result characters are passed through **putc**.

vidattr(newmode)
The proper string to put the terminal in the given video mode is output to **stdout**.

tparm(instring, p1, p2, p3, p4, p5, p6, p7, p8, p9)
The **tparm** function is used to instantiate a parameterized string. The character string returned has the given parameters applied and is suitable for **tputs**. Up to 9 parameters can be passed in addition to the parameterized string.

tputs(cp, affcnt, outc)
A string capability, possibly containing padding information, is processed. Enough padding characters to delay for the specified time replace the padding specification, and the resulting string is passed one character at a time to the routine **outc** that should expect one character as a parameter. (This routine often just calls **putchar**.) The **cp** variable is the capability string. The **affcnt** variable is the number of units affected by the capability, that varies with the particular capability. (For example, the **affcnt** for **insert_line** is the number of lines below the inserted line on the screen, that is, the

number of lines that will have to be moved by the terminal.) The **affcnt** value is used by the padding information of some terminals as a multiplication factor. If the capability does not have a factor, the value 1 should be passed.

putp(str)

This is a convenient function to output a capability with no **affcnt**. The string is output to **putchar** with an **affcnt** value of 1. It can be used in simple applications that do not need to process the output of **tputs**.

delay_output(ms)

A delay is inserted into the output stream for the given number of milliseconds. The current implementation inserts enough pad characters for the delay. This should not be used in place of a high resolution sleep, but rather for delay effects in the output. Because of buffering in the system, it is unlikely that this call will result in the process actually sleeping. Since large numbers of pad characters can be output, it is recommended that **ms** not exceed 500.

Chapter 7

TERMINFO DATABASE

PAGE

PREPARING DESCRIPTIONS 7-1
Naming the Terminal 7-2
Defining Capabilities 7-4
Compiling the New Entry 7-9
Testing an Entry 7-10
TPUT COMMAND 7-11
TERMCAP AND TERMINFO COMPATIBILITY 7-13

Chapter 7

TERMINFO DATABASE

The **terminfo** database describes terminals by giving a set of capabilities and by describing how the terminal performs certain operations. Each terminal description contains the names that the terminal is known by and a group of comma separated fields describing the actions and capabilities of the terminal.

PREPARING DESCRIPTIONS

If there is no terminal description for your terminal, the entry must be built from scratch. You may wish to use partial descriptions and test them as you go along. These tests may expose deficiencies in the ability to describe the terminal.

The general procedure for building this description is as follows:

1. Give the known names of the terminal.
2. List and define the known capabilities.
3. Compile the newly created description entry.
4. Test the entry for correct operation.

Naming the Terminal

The name of the terminal is the first information given in each description. This string of names, assuming there is more than one name, is separated by "pipe" symbols (|). The first name given should be the most common abbreviation for the terminal. The last name given should be a long name fully identifying the terminal. It is usually the manufacturer's formal name for the terminal. All names between the first and last entries should be known synonyms for the terminal name. All names but the formal name should be typed in lowercase letters and contain no blanks. Naturally, the formal name is entered as closely as possible to the manufacturer's name.

Terminal names should follow common naming conventions. These conventions start with a root name; *myterm*, for example. The root name should not contain odd characters, like hyphens, that may not be recognized as a synonym for the terminal name. Possible hardware modes or user preferences should be shown by adding a hyphen and the mode indicator at the end of the name. For example, "wide mode" that is shown by a *-w* would be given as "*myterm-w*."

The following suffixes should be used whenever possible:

Suffix	Meaning	Example
-w	Wide mode (more than 80 columns)	myterm-w
-am	Automatic margins (usually default)	myterm-am
-nam	No automatic margins	myterm-nam
-*n*	Number of lines on the screen (This example defines 30 lines.)	myterm-30
-x*m*	Number of columns	myterm-x132
-*n*x*m*	Both number of lines and columns	myterm-30x80
-na	No arrow keys except in local mode	myterm-na
-*n*p	Number of pages in memory (This example defines 4 pages.)	myterm-4p
-rv	Sets reverse video	myterm-rv
-s	With optional status line enabled	myterm-s

The following example is the name string from the description of the TELETYPE* 5420 Buffered Display Terminal:

```
5420|tty5620|teletype 5420,
```

* Trademark of AT&T

Defining Capabilities

The capabilities for each terminal are described in a string of comma separated fields. This string of fields may continue onto multiple lines as long as white space (that is; tabs, spaces) begins each line except the first line of each description. Comments can be included in the description by entering a number symbol (#) at the beginning of the line.

A complete list of the **terminfo** capabilities is given in the **terminfo** manual page found in the *AT&T 3B2 Computer Programer Reference Manual*. This list contains the name of the capability, the abbreviated name used in the database, the two-letter code that corresponds to the old **termcap** database name, and a short description of the capability. The abbreviated name that you will use in your database descriptions is shown in the column titled **Capname**.

Each terminal description contains abbreviated names of capabilities. Some capabilities also require a terminal-specific instruction that performs the named capability. For example, **bel** is the abbreviated name for the beeping or ringing capability. On most terminals, a "control-g" (^G) is the instruction that produces a beeping sound. Therefore, the beeping capability would be shown in the terminal description as "**bel=^G,**."

The terminal-specific instruction can be a keyed operation (like "^G"), a numeric value, or a parameter string containing the sequence of operations required to achieve the particular capability. There are certain characters that are used after the capability name to shown what type of instruction is required. These characters are explained as follows:

- **#** This shows a numeric value is to follow. This character follows a capability that needs a number as the instruction. For example, the number of columns is defined as "**cols#80,**."

- **=** This shows that the capability instruction is the character string that follows. This string instructs the terminal how to act and may actually be a sequence of commands. There are certain characters used in the instruction strings that have special meanings. These special characters are explained as follows:

^ This shows a control character is to be used. For example, the beeping sound is produced by a "control-G." This would be shown as "**^G**."

\E or \e
These characters followed by another character shows an ESCAPE instruction. An entry of \EC would transmit to the terminal as "ESCAPE-C."

\n These characters provide a "newline" instruction.

\l These characters provide a "linefeed" instruction.

\r These characters provide a "return" instruction.

\t These characters provide a "tab" instruction.

\b These characters provide a "backspace" instruction.

\f These characters provide a "formfeed" instruction.

\s These characters provide a "space" instruction.

$< > These symbols are used to show a delay in milliseconds. The desired amount of delay is enclosed inside the "less than/greater than" symbols (< >). The amount of delay may be a whole number, a numeric value to one decimal place (tenths), or either form followed by an asterisk (*). The "*" shows that the delay will be proportional to the amount of lines affected by the operation. For example, a 20-millisecond delay would appear as "**$<20*>**."

Sometimes, it may be necessary to comment out a capability so that the terminal ignores this particular field. This is done by placing a period (.) in front of the abbreviated name for the capability. For example, if you would like to comment out the beeping capability, the description entry would appear as "**.bel=^G,**."

Basic Capabilities

To build a description from scratch, you would normally start listing the capabilities immediately below the terminal names. The owner's manual for your terminal should provide information on what capabilities are available and what character string makes up the correct instruction to do each capability. It may be beneficial to start with those capabilities that are common to almost all terminals. Some of the common traits of all terminals are bells, columns, lines on the screen, and overstriking of characters if necessary.

For the following example of building a **terminfo** description, we will use a fictitious terminal name. The name string for the terminal is shown as follows:

```
myterm|mytm|mine|fancy|terminal|My FANCY Terminal,
```

Suppose this fictitious terminal has the following capabilities. The **terminfo** manual page of the *AT&T 3B2 Computer Programer Reference Manual* lists these capabilities and gives the abbreviated name to use in the database. The appropriate abbreviated name is shown in parentheses immediately after the capability description.

- An automatic wrap around to the beginning of the next line whenever the cursor reaches the right-hand margin. (**am**)
- The ability to produce a beeping sound. The instruction required to produce the beeping sound is "^G." (**bel**)
- An 80-column wide screen. (**cols**)
- A 30-line long screen. (**lines**)
- An ability to retain the display below the screen. (**db**)

By combining the name string and the capability descriptions that we now have, we can put together a general **terminfo** database entry. Remember that each field must be separated by a comma. The entry would look like this:

```
myterm|mytm|mine|fancy|terminal|My
        FANCY Terminal, am, bel=^G, cols#80, lines#30, db,
```

Keyboard Entered Capabilities

The keyboard entered capabilities are those actions that occur when a key is struck on the keyboard. Although the capabilities may be common to many terminals, the instructions to do the operation could be different. These instructions are related specifically to the terminal that is being described. For example, a carriage return may be shown by a "^M" (control-M) on one terminal. The indication for a carriage return on another terminal may be an "\EG" (ESCAPE-G).

The following characteristics help describe the before mentioned fictitious terminal. Again, the abbreviated command associated with the given capability is shown in parentheses.

- A carriage return is shown by a control-M. (**cr**)
- A cursor up one line motion is shown by a control-K. (**cuu1**)
- A cursor down one line motion is shown by a control-J. (**cud1**)
- Moving the cursor to the left one space is shown by a control-H. (**cub1**)
- Moving the cursor to the right one space is shown by a control-L. (**cuf1**)
- Entering reverse video mode is shown by an ESCAPE-D. (**smso**)

- Exiting reverse video mode is given by an ESCAPE-Z. (**rmso**)
- A clear to the end of a line instruction is shown by an ESCAPE-K and should have a 3-millisecond delay. (**el**)

These capabilities must be added to the general description entry of **myterm**. To do this, simply continue the description in the same way with the new information. The resulting database entry is shown as follows:

```
myterm|mytm|mine|fancy|terminal|My
        FANCY Terminal, am, bel=^G, cols#80, lines#30,
        db, cr=^M, cuu1=^K, cud1=^J, cub1=^H, cuf1=^L,
        smso=\ED, rmso=\EZ, el=\EK$<3>,
```

Note: This is a short, simple **terminfo** database entry. It is shown for illustration purposes only. DO NOT attempt to use this example as a database entry for any terminal.

There are other capabilities that are described using parameter strings. Some parameter string instructions may be the same for different terminals. For example, terminals that conform to the American National Standards Institute (ANSI) standards for computer terminals will all have the same instruction for cursor address (**cup**) and setting attributes (**sgr**). The use of parameter strings is highly complex.

The procedure for building a terminal description and the example shown in this discussion should be enough to show you how a database entry is constructed. Almost all capability descriptions will be defined in one form shown in the example.

Compiling the New Entry

General

The **terminfo** database entries are compiled using the **tic** compiler. This compiler translates **terminfo** database entries from the source format into the compiled format.

The source file for the description must be in a file suffixed with **.ti** . For example, the fictitious database entry being used for illustration purposes would be in a source file named **myterm.ti**. The compiled version is placed in */usr/lib/terminfo/** where * is a directory named with the first letter of each entry. For example, the compiled description of **myterm** (source file **myterm.ti**) would be placed in */usr/lib/terminfo/m* since the first letter in the description entry is "m" (myterm).

Command Format

The general format for the **tic** compiler is as follows:

tic [*-v*] *file file2 ...*

The *-v* option causes the compiler to trace its actions and output information about its progress.

The *file* fields are to show what file is to be compiled. Notice that more than one file can be compiled at one time if the filenames are separated by a space.

Sample Command

The following command line shows how to compile a **terminfo** source file named **myterm.ti**: (The verbose option (*-v*) is included.)

```
$ tic -v myterm.ti<CR>
 (The trace information will
    appear when the compilation
    is complete.)
$
```

Note: The <CR> symbol is used to show a carriage return.

Refer to the **tic** manual page in the *AT&T 3B2 Computer System Administration Reference Manual* for more information on the **terminfo** compiler.

Testing an Entry

You can test a new terminal description entry by setting the environment variable "TERMINFO" to the path name of the directory containing the newly compiled description. If the "TERMINFO" variable is set to a directory before the entry is compiled, the compiled entry will be placed in the "TERMINFO" directory. All programs will look in the new "TERMINFO" directory description file rather than in */usr/lib/terminfo*. If the programs run the same on the new terminal as they did on the older known terminals, then the new description is functional.

A way to test for correct insert line padding is to edit (using *vi*) a large file (over 100 lines) at 9600 baud (if possible), and delete about 15 lines from the middle of the screen. Hit "u" (undo) several times quickly. If the terminal messes up, then more padding is usually required. A similar test can be used for inserting a character.

TPUT COMMAND

General

The **tput** command uses the **terminfo** database to output terminal-specific capabilities and other information to the screen or shell. This command outputs a string or an integer according to the type of capability being described. If the capability is a Boolean expression, then **tput** sets the exit code (0 for TRUE, 1 for FALSE) and produces no output.

Command Format

The general format for the **tput** command is as follows:

tput [*-Ttype*] *capname*

The type of terminal you are requesting information about is identified with the *-Ttype* option. Usually, this option is not necessary because the default terminal name is taken from the environment variable **$TERM**.

The *capname* field is used to show what capability to output from the **terminfo** database.

Sample Command

The following command line shows how to output the clear-screen instruction for the terminal being used:

```
$ tput clear<CR>
  (The instruction for clear-screen
  appears here.)
$
```

The following command line shows how to output the number of columns for the terminal being used:

```
$ tput cols<CR>
  (The number of columns used by
  the terminal will appear here.)
$
```

The **tput** manual page found in the *AT&T 3B2 Computer User Reference Manual* contains more information on the usage and possible messages associated with this command.

TERMCAP AND TERMINFO COMPATIBILITY

The **terminfo** database is designed to take the place of the **termcap** database. Because of the many programs and processes that have been written with and for the **termcap** database, it will be impossible to except a complete cutover at one time. There can be programs written to convert the **termcap** description entries into **terminfo** description entries. However, any conversion from **termcap** to **terminfo** requires much knowledge about both databases. All entrances into the databases should be handled with extreme caution. These files are important to the operation of your computer.

If you have been using cursor optimization programs with the *-ltermcap* option in the "cc" command line, those programs will still be functional. However, the *-ltermcap* option must be replaced with the *-lcurses* option.

Appendix

CURSES EXAMPLES

PAGE

EXAMPLE PROGRAM 'editor' A-2
EXAMPLE PROGRAM 'highlight' A-10
EXAMPLE PROGRAM 'scatter' A-12
EXAMPLE PROGRAM 'show' A-14
EXAMPLE PROGRAM 'termhl' A-16
EXAMPLE PROGRAM 'two' A-19
EXAMPLE PROGRAM 'window' A-22

Appendix

CURSES EXAMPLES

This appendix contains some examples of **curses** programs. Although these examples are functional programs, they are not complete enough to be considered useful. These examples are intended for demonstration purposes only. However, these examples may be used by a skillful programmer as a base to create a useful **curses** program.

The examples contain comments that explain the intended function of a particular step in the program. The comments are for clarity only and are not a functional piece of the program.

EXAMPLE PROGRAM 'editor'

This program is a simple screen editor patterned after the **vi** editor. The program illustrates how to use **curses** to write a screen editor. This editor keeps the buffer in **stdscr** to keep the program simple — obviously, a real screen editor would keep a separate data structure. Many simplifications have been made here — no provision is made for files of any length other than the size of the screen, for lines longer than the width of the screen, or for control characters in the file.

Several points about this program are worth making. The routine to write out the file illustrates the use of the **mvinch** function that returns the character in a window at a given position. The data structure used here does not have a provision for keeping track of the number of characters in a line or the number of lines in the file; so, trailing blanks are eliminated when the file is written out.

The program uses built-in **curses** functions **insch**, **delch**, **insertln**, and **deleteln**. These functions behave like similar functions on intelligent terminals, inserting and deleting a character or line.

The command interpreter accepts not only ASCII characters but also special keys. This is important — a good program will accept both. (Some editors are "modeless," using nonprinting characters for commands. This is largely a matter of taste — the point being made here is that both arrow keys and ordinary ASCII characters should be handled.) It is important to know how to handle special keys. Special keys make it easier for someone else to learn to use your program if they can use the arrow keys instead of having to memorize that "h" means left, "j" means down, "k" means up, and "l" means right. On the other hand, not all terminals have arrow keys; so, your program will be usable on a larger class of terminals if there is an ASCII character that is a synonym for each special key. Also, experienced users dislike having to move their hands from the "home row" position to use special keys, since they can work faster with alphabetic keys.

Note the call to **mvaddstr** in the input routine. The **addstr** function is roughly like the C Language **fputs** function that writes out a string of characters. Like **fputs**, **addstr** does not add a trailing newline. It is the same as a series of calls to **addch** using the characters in the string. (Refer to the **mvaddstr** function description in the "INPUT/OUTPUT" section of Chapter 2.)

The control-L command illustrates a feature that most programs using **curses** should add. Often, some program beyond the control of **curses** has written something to the screen, or some line noise has messed up the screen beyond the tracking capability of **curses**. Here, the user usually types control-L, causing the screen to be cleared and redrawn. This is done with the call to **clearok(curscr)**, that sets a flag causing the next **refresh** to first clear the screen. Then **refresh** is called to force the redraw.

Note also the call to **flash()**, that flashes the screen if possible, and otherwise rings the bell. Flashing the screen is intended as a bell replacement and is particularly useful if the bell bothers someone within earshot of the user. The routine **beep()** can be called when a real beep is desired. (If for some reason the terminal is unable to beep, but able to flash, a call to **beep** will flash the screen.)

Another important point is that the input command is stopped by control-D, not escape. It is tempting to use escape as a command, since escape is a special key that is available on every keyboard. (Return and break are the only others.) However, using escape as a separate key introduces an ambiguity. Most terminals use sequences of characters beginning with escape ("escape sequences") to control the terminal and have special keys that send escape sequences to the computer. If the computer sees an escape coming from the terminal, it cannot tell for sure whether the user pushed the escape key or whether a special key was pressed. Curses handles the ambiguity by waiting for up to one second. If another character is received during this second, and if that character might be the beginning of a special key, more input is read (waiting for up to one second for each character) until either a full special key is read, one second passes, or a character is received that could not have been generated by a special key. While this strategy works most of the time, it is not foolproof.

It is possible for the user to press escape and then to type another key quickly, that causes **curses** to think a special key has been pressed. Also, there is a one second pause until the escape can be passed to the user program, resulting in slower response to the escape key. Many existing programs use escape as a fundamental command that cannot be changed without infuriating a large class of users. Such programs cannot make use of special keys without dealing with this ambiguity, and at best these programs must resort to a timeout solution. The moral is clear: when designing your program, avoid the escape key.

The example program for the simple editor is shown on the following pages.

```
/*
 * editor: A screen-oriented editor.  The user
 * interface is similar to a subset of vi.
 * The buffer is kept in stdscr itself to simplify
 * the program.
 */

#include <curses.h>

#define CTRL(c) ('c' & 037)

main(argc, argv)
char **argv;
{
     int i, n, l;
     int c;
     FILE *fd;

     if (argc != 2) {
          fprintf(stderr, " Usage: edit file\n" );
          exit(1);
     }

     fd = fopen(argv[1], " r" );
     if (fd == NULL) {
          perror(argv[1]);
          exit(2);
     }

     initscr();
     cbreak();
     nonl();
     noecho();
     idlok(stdscr, TRUE);
     keypad(stdscr, TRUE);
```

```
    /* Read in the file */
    while ((c = getc(fd)) != EOF)
        addch(c);
    fclose(fd);

    move(0,0);
    refresh();
    edit();

    /* Write out the file */
    fd = fopen(argv[1], "w" );
    for (l=0; l<23; l++) {
        n = len(l);
        for (i=0; i<n; i++)
            putc(mvinch(l, i), fd);
        putc('\n', fd);
    }
    fclose(fd);

    endwin();
    exit(0);
}

len(lineno)
int lineno;
{
    int linelen = COLS-1;

    while (linelen >=0 && mvinch(lineno, linelen) == ' ')
        linelen--;
    return linelen + 1;
}
```

```
/* Global value of current cursor position */
int row, col;

edit()
{
     int c;

     for (;;) {
          move(row, col);
          refresh();
          c = getch();
          switch (c) {   /* Editor commands */

          /* hjkl and arrow keys: move cursor */
          /* in direction indiated */
          case 'h':
          case KEY_LEFT:
               if (col > 0)
                    col--;
               break;

          case 'j':
          case KEY_DOWN:
               if (row < LINES-1)
                    row++;
               break;

          case 'k':
          case KEY_UP:
               if (row > 0)
                    row--;
               break;

          case 'l':
          case KEY_RIGHT:
               if (col < COLS-1)
                    col++;
               break;
```

```
/* i: enter input mode */
case KEY_IC:
case 'i':
     input();
     break;

/* x: delete current character */
case KEY_DC:
case 'x':
     delch();
     break;

/* o: open a new line and enter input mode */
case KEY_IL:
case 'o':
     move(++row, col=0);
     insertln();
     input();
     break;

/* d: delete current line */
case KEY_DL:
case 'd':
     deleteln();
     break;

/* ^L: redraw screen */
case KEY_CLEAR:
case CTRL(L):
     clearok(curscr);
     refresh();
     break;

/* w: write and quit */
case 'w':
     return;
```

```
            /* q: quit without writing */
            case 'q':
                endwin();
                exit(1);
            default:
                flash();
                break;
            }
        }
}

/*
 * Insert mode: accept characters and insert them.
 *  End with ^D or EIC
 */
input()
{
    int c;

    standout();
    mvaddstr(LINES-1, COLS-20, " INPUT MODE" );
    standend();
    move(row, col);
    refresh();
    for (;;) {
        c = getch();
        if (c == CTRL(D) || c == KEY_EIC)
            break;
        insch(c);
        move(row, ++col);
        refresh();
    }
    move(LINES-1, COLS-20);
    clrtoeol();
    move(row, col);
    refresh();
}
```

EXAMPLE PROGRAM 'highlight'

This program takes a text file as input and allows embedded escape sequences to control attributes. In this example program, **U** turns on underlining, **B** turns on bold, and **N** restores normal text. Note the initial call to **scrollok**. This allows the terminal to scroll if the file is longer than one screen. When an attempt is made to draw past the bottom of the screen, **curses** will automatically scroll the terminal up a line and call **refresh**.

```
/*
 * highlight: a program to turn U, B, and
 * N sequences into highlighted
 * output, allowing words to be
 * displayed underlined or in bold.
 */
#include <curses.h>

main(argc, argv)
char **argv;
{
     FILE *fd;
     int c, c2;

     if (argc != 2) {
          fprintf(stderr, " Usage: highlight file\n" );
          exit(1);
     }

     fd = fopen(argv[1], " r" );
     if (fd == NULL) {
          perror(argv[1]);
          exit(2);
     }

     initscr();
     scrollok(stdscr, TRUE);
```

```
    for (;;) {
        c = getc(fd);
        if (c == EOF)
            break;
        if (c == '\\') {
            c2 = getc(fd);
            switch (c2) {
            case 'B':
                attrset(A_BOLD);
                continue;
            case 'U':
                attrset(A_UNDERLINE);
                continue;
            case 'N':
                attrset(0);
                continue;
            }
            addch(c);
            addch(c2);
        }
        else
            addch(c);
    }
    fclose(fd);
    refresh();
    endwin();
    exit(0);
}
```

EXAMPLE PROGRAM 'scatter'

This program reads a file, and displays the file in a random order on the screen. Some programs assume all screens are 24 lines by 80 columns. It is important to understand that many are not. The variables **LINES** and **COLS** are defined by **initscr** with the current screen size. Programs should use them instead of assuming a 24x80 screen.

```
/*
 *   SCATTER. This program takes the first
 *   23 lines from the standard
 *   input and displays them on the
 *   VDU screen, in a random manner.
 */

#include <curses.h>

#define MAXLINES 120
#define MAXCOLS  160
char s[MAXLINES][MAXCOLS];/* Screen Array */

main()
{
     register int row=0,col=0;
     register char c;
     int char_count=0;
     long t;
     char buf[BUFSIZ];

     initscr();
     for(row=0;row<MAXLINES;row++)
          for(col=0;col<MAXCOLS;col++)
               s[row][col]=' ';
```

```
    row = 0;
    /* Read screen in */
    while( (c=getchar()) != EOF && row < LINES ) {
        if(c != '\n') {
            /* Place char in screen array */
            s[row][col++] = c;
            if(c != ' ')
                char_count++;
        } else {
            col=0;
            row++;
        }
    }

    time(&t); /* Seed the random number generator */
    srand((int)(t&0177777L));

    while(char_count) {
        row=rand() % LINES;
        col=(rand()>>2) % COLS;
        if(s[row][col] != ' ')
        {
            move(row, col);
            addch(s[row][col]);
            s[row][col]=EOF;
            char_count--;
            refresh();
        }
    }
    endwin();
    exit(0);
}
```

EXAMPLE PROGRAM 'show'

The **show** program pages through a file, showing one full screen each time the user presses the space bar. By creating an input file for **show** made up of 24-line pages, each segment varying slightly from the previous page, nearly any exercise for **curses** can be created. Such input files are called "show scripts."

In this program, **cbreak** is called so that the user can press the space bar without having to hit return. The **noecho** function is called to prevent the space from echoing in the middle of a **refresh**, messing up the screen. The **nonl** function is called to enable more screen optimization. The **idlok** function is called to allow insert and delete line, since many show scripts are constructed to duplicate bugs caused by that feature. The **clrtoeol** and **clrtobot** functions clear from the cursor to the end of the line and screen, respectively.

```
#include <curses.h>
#include <signal.h>

main(argc, argv)
int argc;
char *argv[];
{
    FILE *fd;
    char linebuf[BUFSIZ];
    int line;
    void done(), perror(), exit();

    if(argc != 2)
    {
        fprintf(stderr," usage: %s file\n" , argv[0]);
        exit(1);
    }
    if((fd=fopen(argv[1]," r" )) == NULL)
    {
        perror(argv[1]);
        exit(2);
```

```
        }
        signal(SIGINT, done);

        initscr();
        noecho();
        cbreak();
        nonl();
        idlok(stdscr, TRUE);

        while(1)
        {
                move(0,0);
                for(line=0; line<LINES; line++)
                {
                        if(fgets(linebuf, sizeof linebuf, fd) == NULL)
                        {
                                clrtobot();
                                done();
                        }
                        move(line, 0);
                        printw(" %s" , linebuf);
                }
                refresh();
                if(getch() == 'q')
                        done();
        }
}


void
done()
{
        move(LINES-1, 0);
        clrtoeol();
        refresh();
        endwin();
        exit(0);
}
```

EXAMPLE PROGRAM 'termhl'

```
/*
 * A terminfo level version of highlight.
 */
#include <curses.h>
#include <term.h>

int ulmode = 0;          /* Currently underlining */

main(argc, argv)
char **argv;
{
     FILE *fd;
     int c, c2;
     int outch();

     if (argc > 2) {
          fprintf(stderr, " Usage: termhl [file]\n" );
          exit(1);
     }

     if (argc == 2) {
          fd = fopen(argv[1], " r" );
          if (fd == NULL) {
               perror(argv[1]);
               exit(2);
          }
     } else {
          fd = stdin;
     }

     setupterm(0, 1, 0);
```

```
    for (;;) {
        c = getc(fd);
        if (c == EOF)
            break;
        if (c == '\\') {
            c2 = getc(fd);
            switch (c2) {
            case 'B':
                tputs(enter_bold_mode, 1, outch);
                continue;
            case 'U':
                tputs(enter_underline_mode, 1, outch);
                ulmode = 1;
                continue;
            case 'N':
                tputs(exit_attribute_mode, 1, outch);
                ulmode = 0;
                continue;
            }
            putch(c);
            putch(c2);
        }
        else
            putch(c);
    }
    fclose(fd);
    fflush(stdout);
    resetterm();
    exit(0);
}
```

```
/*
 * This function is like putchar, but it checks for underlining.
 */
putch(c)
int c;
{
     outch(c);
     if (ulmode && underline_char) {
          outch('\b');
          tputs(underline_char, 1, outch);
     }
}

/*
 * Outchar is a function version of putchar that can be passed to
 * tputs as a routine to call.
 */
outch(c)
int c;
{
     putchar(c);
}
```

EXAMPLE PROGRAM 'two'

This program pages through a file, showing one page to the first terminal and the next page to the second terminal. It then waits for a space to be typed on either terminal, and shows the next page to the terminal typing the space. Each terminal has to be separately put into nodelay mode. Since no standard multiplexor is available in current versions of the UNIX System, it is necessary to either busy wait or call **sleep(1);** between each check for keyboard input. This program sleeps for a second between checks.

```
#include <curses.h>
#include <signal.h>

struct screen *me, *you;
struct screen *set_term();

FILE *fd, *fdyou;
char linebuf[512];

main(argc, argv)
char **argv;
{
     int done();
     int c;

     if (argc != 4) {
          fprintf(stderr, " Usage: two othertty otherttytype inputfile\n" );
          exit(1);
     }

     fd = fopen(argv[3], " r" );
     fdyou = fopen(argv[1], " w+" );
     signal(SIGINT, done);   /* die gracefully */

     me = newterm(getenv(" TERM" ), stdout);/* initialize my tty */
     you = newterm(argv[2], fdyou);/* Initialize his terminal */
```

```
    set_term(me);           /* Set modes for my terminal */
    noecho();               /* turn off tty echo */
    cbreak();           /* enter cbreak mode */
    nonl();                 /* Allow linefeed */
    nodelay(stdscr,TRUE); /* No hang on input */

    set_term(you);          /* Set modes for other terminal */
    noecho();
    cbreak();
    nonl();
    nodelay(stdscr,TRUE);

    /* Dump first screen full on my terminal */
    dump_page(me);

    /* Dump second screen full on his terminal */
    dump_page(you);

    for (;;) {              /* for each screen full */
        set_term(me);
        c = getch();
        if (c == 'q')       /* wait for user to read it */
            done();
        if (c == ' ')
            dump_page(me);

        set_term(you);
        c = getch();
        if (c == 'q')       /* wait for user to read it */
            done();
        if (c == ' ')
            dump_page(you);
        sleep(1);
    }
}

dump_page(term)
struct screen *term;
```

```
{
    int line;

    set_term(term);
    move(0, 0);
    for (line=0; line<LINES-1; line++) {
        if (fgets(linebuf, sizeof linebuf, fd) == NULL) {
            clrtobot();
            done();
        }
        mvprintw(line, 0, " %s" , linebuf);
    }
    standout();
    mvprintw(LINES-1, 0, " --More--" );
    standend();
    refresh();          /* sync screen */
}

/*
 * Clean up and exit.
 */
done()
{
    /* Clean up first terminal */
    set_term(you);
    move(LINES-1,0); /* to lower left corner */
    clrtoeol();         /* clear bottom line */
    refresh();          /* flush out everything */
    endwin();           /* curses cleanup */

    /* Clean up second terminal */
    set_term(me);
    move(LINES-1,0); /* to lower left corner */
    clrtoeol();         /* clear bottom line */
    refresh();          /* flush out everything */
    endwin();           /* curses cleanup */

    exit(0);
}
```

EXAMPLE PROGRAM 'window'

The main display of this program is kept in **stdscr**. When the user temporarily wants to put something else on the screen, a new window is created covering part of the screen. A call to **wrefresh** on that window causes the window to be written over **stdscr** on the screen. Calling **refresh** on **stdscr** results in the original window being redrawn on the screen. Note the calls to **touchwin** before writing out an overlapping window. These are necessary to defeat an optimization in **curses**. If you have trouble refreshing a new window that overlaps an old window, it may be necessary to call **touchwin** on the new window to get it completely written out.

```
#include <curses.h>

WINDOW *cmdwin;

main()
{
     int i, c;
     char buf[120];

     initscr();
     nonl();
     noecho();
     cbreak();

     cmdwin = newwin(3, COLS, 0, 0);/* top 3 lines */
     for (i=0; i<LINES; i++)
          mvprintw(i, 0, " This is line %d of stdscr" , i);
```

```
    for (;;) {
        refresh();
        c = getch();
        switch (c) {
        case 'c': /* Enter command from keyboard */
            werase(cmdwin);
            wprintw(cmdwin, " Enter command:" );
            wmove(cmdwin, 2, 0);
            for (i=0; i<COLS; i++)
                waddch(cmdwin, '-');
            wmove(cmdwin, 1, 0);
            touchwin(cmdwin);
            wrefresh(cmdwin);
            wgetstr(cmdwin, buf);
            touchwin(stdscr);
            /*
             * The command is now in buf.
             * It should be processed here.
             */
            break;
        case 'q':
            endwin();
            exit(0);
        }
    }
}
```

Replace this

page with the

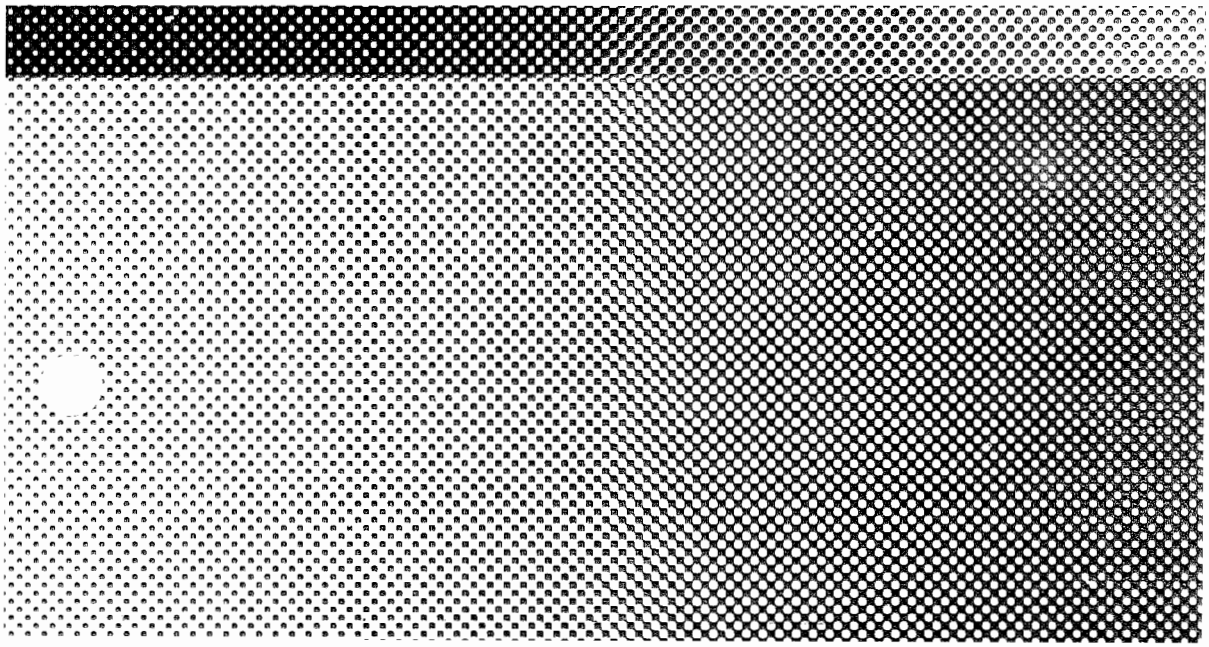
USER ENVIRONMENT

tab separator.

AT&T

AT&T 3B2 Computer

UNIX™ System V Release 2.0 User Environment Utilities Guide



CONTENTS

Chapter 1. INTRODUCTION

Chapter 2. COMMAND DESCRIPTIONS

Chapter 1

INTRODUCTION

PAGE

GENERAL 1-1
GUIDE ORGANIZATION 1-2

Chapter 1

INTRODUCTION

GENERAL

This guide describes command syntax and use of the User Environment Utilities available with your AT&T 3B2 Computer.

The 3B2 Computer user operates in a predefined executing environment. This environment is defined when a user logs in. The login process establishes environment variables, home directory, path, etc. A user may optionally add or alter environment variables in a **.profile** in his home directory.

Many User Environment Utilities commands allow users to control their inherited environment. This allows the user to schedule commands to be executed at a specific time, or access more than one shell from a single terminal.

Four of the commands can be used to do arithmetic calculations. These commands can be executed directly, or used inside a file.

GUIDE ORGANIZATION

The remainder of this guide, Chapter 2 -- "COMMAND DESCRIPTION," describes the command formats (syntax) for each command in the User Environment Utilities. The descriptions include the purpose of the command, a discussion of the command syntax and options, and examples of using each command.

Chapter 2

COMMAND DESCRIPTIONS

PAGE

GENERAL 2-1

HOW COMMANDS ARE DESCRIBED 2-4

COMMAND SUMMARY 2-6

- at — Execute Command at a Specified Time 2-6
- banner — Make Banners 2-9
- batch — Execute Commands at a Later Time 2-11
- bc — Calculator 2-13
- cal — Print Calendar 2-27
- calendar — Reminder Service 2-29
- crontab — Clock Used to Schedule Commands 2-31
- dc — Calculator 2-35
- env — Set Environment for Command Execution 2-45
- factor — Find Prime Factors of a Number 2-47
- logname — Print Login Name 2-49
- nice — Run a Command at Low Priority 2-51
- nohup — Run a Command Immune to Hangups or Quits 2-53
- shl — Layered Shell 2-55
- tabs — Set Tab Stops on a Printer or Terminal 2-59
- tty — Print the Terminal Name 2-61
- units — Find Unit Conversion Factors 2-63
- xargs — Construct Argument List(s) and Execute Command 2-67

Chapter 2

COMMAND DESCRIPTIONS

GENERAL

The User Environment Utilities provide eighteen **UNIX*** System commands. A summary of these commands is provided in Figure 2-1.

The User Environment commands are useful when:

- Performing mathematical calculations
- Writing shell programs
- Checking or changing executing environment of commands
- Scheduling commands to be executed at a later time.

---

* Trademark of AT&T

COMMAND	DESCRIPTION
at	Used to execute commands at a specified time.
banner	Creates a banner with large letters.
batch	Used to execute commands when the system load level permits.
bc	Used to do precise arithmetic calculations.
cal	Prints a calendar for a specified month and/or year.
calendar	Invokes a user's reminder service.
crontab	Used to schedule command execution using **cron**.
dc	Used to do precise arithmetic calculations using a stack to keep a record of the previous calculation.
env	Executes a command with modified environment variables.
factor	Used to calculate the prime factors of any number.

Figure 2-1. User Environment Commands (Sheet 1 of 2)

COMMAND	DESCRIPTION
logname	Displays the environment variable **$LOGNAME** assigned to the user on login.
nice	Runs a command at a lower priority level.
nohup	Makes a command immune to hangups and quits.
shl	Allows a user to interact with several shells from the same terminal.
tabs	Sets tab stops on a printer or terminal.
tty	Prints the path name of the user's terminal.
units	Used to find the conversion factor between two different standard unit values.
xargs	Used to execute a command or a shell program one or more times by combining arguments to this command with arguments read from the standard input.

Figure 2-1. User Environment Commands (Sheet 2 of 2)

HOW COMMANDS ARE DESCRIBED

A common format is used to describe each of the commands. The format is as follows:

- **General:** The purpose of the command is defined. Any uncommon or special information about the command is also provided.
- **Command Format:** The basic command format (syntax) is defined and the various arguments and options discussed.
- **Sample Command Use:** Example command line entries and system responses are provided to show you how to use the command.

In the command format discussions, the following symbology and conventions are used to define the command syntax:

- The basic command is shown in bold type. For example: **command** is in bold type.
- Arguments that you must supply to the command are shown in a special type. For example: **command** *argument*
- Command options and arguments that do not have to be supplied are enclosed in brackets ([]). For example: **command** *[optional arguments]*
- The pipe symbol (|) is used to separate arguments when one of several forms of an argument can be used for a given argument field. The pipe symbol can be thought of as an exclusive OR function in this context. For example:
  command *[argument1 | argument2]*

In the sample command discussions, the lines that you input are ended with a carriage return. This is shown by using <*CR*> at the end of the lines.

Refer to the *AT&T 3B2 Computer User Reference Manual* for UNIX System V manual pages supporting the commands described in this guide.

The following conventions are used to show your terminal input and the system output:

```
This style of type is used to show system generated
responses displayed on your screen.

This style of bold type is used to show inputs
entered from your keyboard that are displayed on your
screen.

These bracket symbols, < > identify inputs from the
keyboard that are not displayed on your screen, such
as: <CR> carriage return, <CTRL d> control d, <ESC g>
escape g, passwords, and tabs.

This style of italic type is used for notes that
provide you with additional information.
```

COMMAND SUMMARY

at — Execute Command at a Specified Time

General

The **at** command is used to execute one or more commands at a specified time and date. Output from the at command is mailed to the user unless it is redirected to a file, printer, etc.

Command Format

The **at** command has the following format:

at *time [date] [+ increment]*
at *-r job...*
at *-l [job...]*

The *time* argument identifies the time of day you want the at-file run. A 24-hour clock is assumed unless the optional letters, **A** (AM), **P** (PM), **N** (noon), or **M** (midnight) follow the four digit number.

The *date* argument can be a month followed by a space and then the day, or just the name of the day of the week. You can also specify **week** that means schedule for seven days from today's date.

The + *increment* argument allows you to schedule commands to be executed according to a given interval. The increment is a number followed by a following interval: minutes, hours, days, weeks, months, or years.

The *-l* option lists all the jobs you have previously scheduled with the **at** or **batch** command. **at** job numbers are ended with a **.a** and **batch** job numbers with a **.b**.

The *-r* option removes jobs previously scheduled by the **at** or **batch** commands.

Sample Commands

The following examples show how to enter an **at** command to execute the file "filename" at a specified time:

```
$ at 11:15 July 19<CR>
nroff -mm memo > memo.f<CR>
<CTRL d>
job 460653300.a at Fri July 19 11:15:00 1985
$
```

```
$ at 1350 < filename > outputfile<CR>
job 460653300.a at Fri July 19 11:15:00 1985
$
```

```
$ at -r 460653300.a<CR>
$
```

banner — Make Banners

General

The **banner** command prints a banner of the arguments specified in the command line. Each of the arguments can be up to 10 characters long. Quotation marks can be used to force the output onto one line.

Command Format

The **banner** command has the following format:

banner *arguments*

Arguments cannot be more than 10 characters. There is no limit to the amount of arguments that can be specified.

Sample Command

The following example shows how to enter the **banner** command and the response that would follow:

```
$ banner "1 2 3" <CR>

  #            #####          #####
 ##           #     #        #     #
# #                 #              #
  #            #####          #####
  #           #                    #
  #           #              #     #
#####         #######         #####

$
```

batch — Execute Commands at a Later Time

General

The **batch** command is used to execute commands at a later time when the system load level permits. It is useful for running text processing commands such as *nroff* or *troff*, or for compiling C programs that require a large amount of processor time. The output from the **batch** command is mailed to the user unless it is redirected to a file.

Command Format

The **batch** command accepts input directly or given in a file. The format of the **batch** command, when data is entered directly, is as follows:

batch
command lines
<CTRL d>

The *command lines* argument represents the commands you wish to execute.

The format of the **batch** command, when data is given in a file is as follows:

batch < *file*

The *file* argument identifies the file containing the commands you want to execute.

To list the jobs you have previously scheduled, use the **at -l** command. To remove jobs, use the **at -r** command.

Sample Command

The following command line entry shows how to enter an *nroff* command using the **batch** command and the response that would follow:

```
$ batch<CR>
nroff file1 > file2<CR>
<CTRL d>
job 460657390.b at Fri July 19 12:35:01 1985
$
```

bc — Calculator

General

The **bc** command is used to do basic arithmetic, number base conversions, trigonometric functions, exponential calculations, natural logarithm calculations, and Bessel functions. It reads input from file arguments given and then reads standard input.

The **bc** command is actually a preprocessor for the **dc** command. The output from the **bc** command is piped to **dc** unless the **-c** (compile only) option is present.

The **bc** command has special operators and functions that must be used when performing any calculations. These operators and functions are described in the following list:

+	adds two numbers and displays the result.
-	subtracts two numbers and displays the result.
/	divides the left argument by the right argument and displays the result.
*	multiplies two numbers and displays the result.
%	displays the remainder from a division process.
^	raises the left argument to a power denoted by the right argument.
a	displays the arctangent of the right argument.
c	displays the cosine of the right argument.
s	displays the sine of the right argument.

e	displays the exponential of the right argument.
l	displays the natural logarithm of the right argument.
ibase	changes the input base.
obase	changes the output base.
scale	changes the amount of digits to the right of the decimal point.
quit	exits the program.

Command Format

Input for the **bc** command can be entered directly or given in a file. The format of the **bc** command, when data is entered directly, is as follows:

bc
calculation

The *calculation* argument represents the mathematical calculation you wish to execute.

The format of the **bc** command, when data is given in a file, is as follows:

bc [*-c*] [*-l*] *file*

The *-c* (compile only) option causes the preprocessed output from **bc** to be written to standard output instead of piping it to **dc**. No calculations are performed.

The *-l* option passes the input through an arbitrary precision math library. This library contains the sine, cosine, exponential, log, arctangent, and Bessel functions. See examples in "Trigonometric and Exponential Calculations."

The *file* argument identifies the file you want the **bc** command to act on. Information on variables, operators, and statements you can use in these files are given in the manual pages.

Sample Calculations

The following examples show how to use the **bc** command for most commonly used calculations. To do these examples, first be sure you have the system prompt (**$**).

Addition, Subtraction, Multiplication, and Division

In this example, you want to add 5 plus 5:

```
$ bc<CR>
5+5<CR>
10
quit<CR>
$
```

This format is also used for subtraction, multiplication, and division. The only difference is you have to use the appropriate operator.

Calculations with Negative Numbers

To do calculations with negative numbers, you must precede the negative number with a minus sign (-). In this example, you want to add a negative 6 to a positive 3 (a positive number will not have a + sign preceding it):

```
$ bc<CR>
-6+3<CR>
-3
quit<CR>
$
```

Find the Remainder of a Division Calculation

The **bc** command rounds off the remainder when performing division calculations. To find the remainder, you need to use the remainder **%** operator. In this example, you want to find the remainder of the problem (17 divided by 5). Notice that only the remainder is displayed:

```
$ bc<CR>
17%5<CR>
2
quit<CR>
$
```

Raise a Number to a Power

In this example, you want to raise 2 to the 6 power:

```
$ bc<CR>
2 6<CR>
64
quit<CR>
$
```

Find the Square Root

In this example, you want to find the square root of 49:

```
$ bc<CR>
sqrt(49)<CR>
7
quit<CR>
$
```

Combining Calculations

When combining several types of calculations into one problem, the **bc** command performs the calculations according to the following standard rules of arithmetic:

1. All calculations enclosed in parenthesis are performed first. If any calculations enclosed in parenthesis are located within another set of parenthesis, the innermost calculations are always performed first.

2. All calculations involving square root and exponentiation are performed next. These calculations are performed from right to left.

3. After performing all calculations enclosed in parenthesis, multiplication and division calculations are performed. These calculations are performed from left to right.

4. Unenclosed addition and subtraction calculations are always performed last. These calculations are performed from left to right.

With **bc**, you can do many calculations with one input. In this example, you want to evaluate 5 times 4 plus 3 times (2 + 2) times 2 to the third power and divide the entire calculation by 2:

```
$ bc<CR>
(5*4+3*(2+2)*2^3)/2<CR>
58
quit<CR>
$
```

Continuous Calculations

You can keep doing calculations with **bc** after each answer without having to quit **bc**. You can also change the type of calculation being performed without quitting **bc**. In this example, you have to do five problems: add 30 to 10, add 6 to 3, multiply 2 times 6, multiply 3 times 8, and add 3 to the sum of 2 times 6. The following example shows how to do all five calculations before quitting **bc**:

```
$ bc<CR>
30+10<CR>
40
6+3<CR>
9
2*6<CR>
12
3*8<CR>
24
3+2*6<CR>
15
quit<CR>
$
```

Changing the Accuracy of Calculations

You can change the accuracy of calculations by increasing the number of digits after the decimal point. In **bc**, this is known as changing the **scale**. The scale in **bc** is initially set to 0. After changing the scale, to get **bc** back to a scale of 0 you must change the scale to 0. In this example, you want to divide 100 by 3 with an accuracy of 5 digits after the decimal point:

```
$ bc<CR>
scale=5<CR>
100/3<CR>
33.33333
scale=0<CR>
quit<CR>
$
```

Changing the Input Base

The **bc** command is normally set for a base of 10 (decimal). You can change the input base from base 10 to base 8 (octal), or base 16 (hexadecimal). To return **bc** to base 10, you must change the input base to 10. You convert the input base to find the base 10 equivalent of a number in base 8, or base 16. In the next example, you want to find the base 10 equivalent of 1000 in base 16 and then convert back to base 10 using the command "ibase=A":

```
$ bc<CR>
ibase=16<CR>
1000<CR>
4096
ibase=A<CR>
1000<CR>
1000
quit<CR>
$
```

Changing the Output Base

The **bc** command is normally set for base 10 (decimal). You can change the output base from base 10 to base 8 (octal), or base 16 (hexadecimal). To return **bc** to base 10, you must change the output base to 10. You convert the output base to find the base 8, or base 16 equivalent of a base 10 number.

In this example, you want to find the base 16 equivalent of 1000 in base 10 and then change back to base 10 using the command "obase=A":

```
$ bc<CR>
obase=16<CR>
1000<CR>
3E8
obase=A<CR>
1000<CR>
1000
quit<CR>
$
```

Using Registers in Calculations

You can do calculations using one or more of 26 registers named **a** through **z**. Registers can be used in any of the following types of calculations:

- Addition
- Subtraction
- Multiplication
- Division (positive and whole number results are required when you use more than one register)

- Raise a number (register) to a power
- Multiple calculations using a parenthetical expression.

You can do more than one calculation at a time using register(s). Once you have done a calculation using a register, the result of the calculation will be the content of the register.

You must be in the **bc** command to use these registers. You enter information into a register by typing the register name (**a** through **z**), an equal sign (=), the information (or calculation), and a carriage return. To see the information or result of a calculation, you enter the register name followed by a carriage return. The system will display the content of the register.

You can also do calculations with a register using the **bc -l** command. However, you can only use one register.

In the following sample calculations you want to enter 4 into the **x** register and 2 into the **y** register. Then check the contents of the register before performing the sample calculations.

Inputting 4 in the **x** register and 2 in the **y** register:

```
$ bc<CR>
x=4<CR>
y=2<CR>
x<CR>
4
y<CR>
2
z=x+y<CR>
z<CR>
6
z=y-x<CR>
z<CR>
-2
z=x*y<CR>
z<CR>
8
z=x/y<CR>
z<CR>
2
z=x^y<CR>
z<CR>
16
z=(x+x)*y<CR>
z<CR>
32
quit<CR>
$
```

Trigonometric and Exponential Calculations

The **bc** command has an option (**-l**) that accesses the arbitrary precision math library. This option can be used with the **bc** command to:

- Find the sine of an angle
- Find the cosine of an angle
- Find the arctangent of a number
- Find the natural logarithm (ln) of a number
- Raise a number to the **e** (2.718) power.

The **bc -l** command performs all calculations involving angles in radians. Therefore, if the angle you want to work with is in degrees, you must change it to radians (360 degrees = 2 pi radians). To change degrees to radians, multiply the amount of degrees by 0.01745.

Find the Sine of 45 Degrees:

```
$ bc -l<CR>
45*.01745<CR>
0.78525
s(.78525)<CR>
.707002
quit<CR>
$
```

Find the Cosine of 1.04700 Radians (60 Degrees):

```
$ bc -l<CR>
c(1.04700)<CR>
.500171
quit<CR>
$
```

Find the Natural Logarithm of the Number 20:

```
$ bc -l<CR>
l(20)<CR>
2.995732
quit<CR>
$
```

Find the Exponential of the Number 2:

```
$ bc -l<CR>
e(2)<CR>
7.389056
quit<CR>
$
```

Find the Sine of 45 Degrees using a register:

```
$ bc -l<CR>
x=45*0.01745<CR>
s(x)<CR>
.707002
quit<CR>
$
```

cal — Print Calendar

General

The **cal** command prints a calendar for the year specified. If an argument for a month is also specified, a calendar for only that month will be printed.

Command Format

The **cal** command has the following format:

cal *[[month] year]*

month must be a number ranging from 1 through 12

year must be a number ranging from 1 through 9999.

If no arguments are given, a calendar for the current month will be printed.

Sample Command

The following example shows how to enter the **cal** command and the response that would follow:

```
$ cal 7 1985<CR>
       July 1985
 S   M Tu  W Th  F  S
     1  2  3  4  5  6
 7   8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31

$
```

calendar — Reminder Service

General

The **calendar** command is used to access a file in the current directory named *calendar* and output the lines that contain today's or tomorrow's date. The **calendar** command can be automatically executed on a daily basis as a function of the **cron** command. When automatically executed, the **calendar** command accesses every *calendar* file on the system and reports any results to the appropriate user via **mail**. To receive reminders automatically via **mail**, the *calendar* file should be created in your login directory.

The lines in the *calendar* file must contain dates in some reasonable form. The following is an example of a *calendar* file located in your login directory:

```
$ cat calendar<CR>
Confirm travel reservations on 07/23/85.
Bill's 10th anniversary on July 27, 1985.
Appointment with Dr. Miller on 7/29/85 at 9:00 a.m.
Nancy's birthday on Oct 29.
$
```

Each of the lines containing a date will be output via mail on that day and the preceding day.

Command Format

The **calendar** command has the following format:

calendar [-]

The - argument is an optional flag that causes the command to execute for every user having a *calendar* file. This is the form of the command that is executed by the **cron** command to check all *calendar* files on a daily basis. Without the argument, the **calendar** command will only check your **calendar** file and report its contents.

Sample Command

The following example shows how to enter the **calendar** command and the output that would follow if the current date was July 19, 1985:

```
$ calendar<CR>
Bill's 10th anniversary on July 27, 1985.
Appointment with Dr. Miller on 7/29/85 at 9:00 a.m.
$
```

In order for this example to be correct, the *calendar* file must be located in the current working directory.

To provide reminder service automatically, an entry must be made in the crontab file for **calendar**. The following is a sample crontab entry for **calendar**:

0 9 * * * /usr/lib/calendar -

With this entry, the **calendar** command would execute every morning at 9:00 a.m. (provided the system is up) and search all login directories for *calendar* files. For a detailed description of the crontab file and how its entries are structured, refer to the *AT&T 3B2 Computer System Administration Utilities Guide.*

crontab — Clock Used to Schedule Commands

General

The **crontab** command allows specified users to submit their jobs for execution. Users are permitted to use **crontab** if their name is in the file **/usr/lib/cron/cron.allow**. Users are denied permission if their name is in the file **/usr/lib/cron/cron.deny**. If neither file exists, only root is allowed to use the **crontab** command.

The **crontab** command accepts a command file or standard input. The input file normally contains lines that are broken into six fields. Each field is separated by a space. The first five fields are dedicated to:

minute (0-59)
hour (0-23)
day of the month (1-31)
month of the year (1-12)
day of the week (0-6 with Sunday=0)

The last field is the command you want to execute. If no file is specified, the data is read from the standard input.

Commands in a crontab file are executed in a user's home directory. The output from the command is mailed to the user unless redirected. Therefore, you would normally want to redirect the output to a file.

The **crontab** command is especially useful for scheduling programs to run periodically, such as accounting programs, weekly reports, or system usage programs.

Command Format

The **crontab** command uses the following formats:

crontab *[-l] [-r] [file]*

[file] contains the command lines a user wishes to have executed by cron. If the file is not specified, command lines are read from standard input.

The *-l* option lists the crontab files for the invoking user.

The *-r* option removes the users files from the crontab directory.

None of the options for the **crontab** command can be used together.

Sample Commands

The following is an example of the **crontab** command using a command file as the input.

Create the command file (whoison) containing the following line:

0 8 * * 1-5 who ;ps >> outputfile

This line instructs cron to execute a "who" command and a "ps" command every Monday through Friday at 8 AM and place the output in a file named "outputfile" in your home directory. To have cron execute the command file you would enter the command:

```
$ crontab whoison<CR>
$
```

To have cron execute the same command using standard input, you would enter the following commands:

```
$ crontab<CR>
0 8 * * 1-5 who ;ps >> outputfile<CR>
<CTRL d>
$
```

To list all the commands in your crontab file, you would use the **l** option. For example:

```
$ crontab -l<CR>
0 8 * * 1-5 who ;ps >> outputfile
$
```

dc — Calculator

General

The **dc** command does various arithmetic calculations on one or two numbers that have been placed on a pushdown stack. The stack is like that of a calculator using reverse Polish notation. The **dc** command takes one or two numbers from the top of the stack, performs the desired arithmetic operation, and returns the number(s) to the stack.

dc Operators and Functions

+	adds the top two numbers on the stack and stores the result in their place.
-	subtracts the top two numbers on the stack and stores the result in their place.
*	multiplies the top two numbers on the stack and stores the result in their place.
/	divides the top two numbers on the stack and stores the result in their place.
%	stores the remainder of a division operation in the top of the stack.
^	raises the second number from the top of the stack to the number in the top of the stack and stores the result in their place.

> ***Note:*** When using +, -, *, /, **%**, or ^, an exponent must not have any digits after the decimal point.

v restores the top number of the stack with the square root of that number.

s*x* stores the number on the top of the stack in a register named *x* (where *x* may be any character). If **s** is uppercase, *x* is treated as a stack.

l*x* stores the number in register *x* on the stack. If the **l** is uppercase, register *x* is treated as a stack and its top value is placed on the main stack.

> ***Note:*** All registers start with an empty value that is treated as a zero by the command **l** and as an error by the command **L**.

c clears all the values in the stack.

d duplicates the top value on the stack.

p prints the top value on the stack.

f prints all values on the stack and in the registers.

x removes the top element from the stack. Treats it as a character string and executes it as a string of dc commands.

[...] puts the bracketed character string on the top of the stack.

q exits the program.

Command Format

The **dc** command accepts data entered directly or from file arguments. The format of the **dc** command when data is entered directly is as follows:

dc
data on stack
operation symbol
p

The *data* argument represents any numbers you have input or numbers existing in the stack.

The *operation symbol* argument represents the mathematical calculation you wish to execute.

The **p** prints the character on top of the stack.

The format of the **dc** command when data is given from a file is as follows:

dc *file*

The *file* argument identifies the file that you want input to the **dc** command.

Sample Calculations

The following examples show how to use the **dc** command when the data is entered directly.

Addition, Subtraction, Multiplication, and Division

In this example, you want to add 10 and 20:

```
$ dc<CR>
10<CR>
20<CR>
+<CR>
p<CR>
30
q<CR>
$
```

This format is also used for subtraction, multiplication, and division. The only difference is you have to use the appropriate operation symbol.

Calculations with Negative Numbers

To do calculations with negative numbers, you must precede the negative number with a minus (-) sign. In this example, you are adding a negative 6 to a positive 3 (a positive number does not need a + preceding it):

```
$ dc<CR>
-6<CR>
3<CR>
+<CR>
p<CR>
-3
q<CR>
$
```

Find the Remainder of a Division Calculation

When performing division calculations, the **dc** command rounds off the remainder. To find the remainder of a division calculation, you need to use the remainder (**%**) calculation symbol instead of the division symbol. In this example, you want to find the remainder of the problem (25 divided by 3):

```
$ dc<CR>
25<CR>
3<CR>
%<CR>
p<CR>
1
q<CR>
$
```

Raise a Number to a Power

In this example, you are raising the number 2 to the 6th power:

```
$ dc<CR>
2<CR>
6<CR>
 <CR>
p<CR>
64
q<CR>
$
```

Find the Square Root of a Number

In this example, you want to find the square root of 49:

```
$ dc<CR>
49<CR>
v<CR>
p<CR>
7
q<CR>
$
```

Note: Square root answers are rounded off to the nearest whole number.

Combining Calculations

In this example, you are executing a string of different types of calculations without quitting between each one. Also, notice that the answer is not displayed until it is requested by entering **p**:

```
$ dc<CR>
12<CR>
6<CR>
*<CR>
3<CR>
-<CR>
25<CR>
5<CR>
/<CR>
+<CR>
p<CR>
74
q<CR>
$
```

Continuous Calculations

The **dc** command allows you to do separate calculations continuously without having to return to the UNIX System. You can also change the type of calculation being performed. In this example, you are executing two completely separate calculations without having to quit **dc**:

```
$ dc<CR>
30<CR>
10<CR>
+<CR>
p<CR>
40
c<CR>
5<CR>
6<CR>
*<CR>
p<CR>
30
q<CR>
$
```

Converting Number Base

Normally, **dc** is set for base 10 (decimal). You can set the input and output bases in **dc** to base 10 (decimal), base 8 (octal), or base 16 (hexadecimal). By changing the input base, you can do calculations in the different bases. By changing the output base you can do calculations in one base and print the result in the base you want. To change either base back to base 10, just enter **q** followed by a carriage return.

Changing the Input Base The following example shows how to change the input base to base 16 and then add 13 to 16:

```
$ dc<CR>
16<CR>
i<CR>
13<CR>
16<CR>
+<CR>
p<CR>
65
q<CR>
$
```

Changing the Output Base The following example shows how to change the output base to base 8 and add 12 to 12:

```
$ dc<CR>
8<CR>
o<CR>
12<CR>
12<CR>
+<CR>
p<CR>
30
q<CR>
$
```

Changing the Accuracy of Calculations

You can increase the accuracy of calculations by increasing the amount of digits after the decimal point. To do this in **dc**, you change the scale (scale is shown as **k**). The scale in **dc** is normally set to 0. After you change the scale from 0, to get **dc** back to a scale of 0, enter **q** followed by a carriage return.

In the following example, you want to change the scale to 5, then divide 100 by 3, and change the scale back to 0:

```
$ dc<CR>
5<CR>
k<CR>
100<CR>
3<CR>
/<CR>
p<CR>
33.33333
q<CR>
$
```

env — Set Environment for Command Execution

General

The **env** command momentarily changes the user's environment variables for execution of a command. A user's environment is automatically initialized on login. The environment variables of the shell are unchanged. For information on the variables and how they affect the executing environment, refer to the section in the *UNIX System User Guide* where shell programming variables are described.

Command Format

The **env** command has the following format:

env *[-] [name=value] ... [command [arguments]]*

The - flag eliminates the current environment so that the environment consists of the specified variables only.

Arguments of the form *name=value* are environment variable(s) that are to be inherited into the current environment before executing *command arguments.* More than one argument of the form *name=value* can be listed.

If there is no *command* specified, the **env** command will print the resulting environment. This is useful for checking the environment before executing a command.

Sample Command

In a shell program, you may have a need to temporarily change the PATH variable in the executing environment. You may have several versions of a program, and each program and the files required for its execution are located in its own directory. If the current PATH variable does not include the path needed, you could use the **env** command to temporarily change the path as follows:

```
$ env - PATH=/usr/rsc/dbase/labels1.4 getlabels<CR>
$
```

where **getlabels** is a shell program to be executed using the files located in the **labels1.4** directory. Therefore, all files required for execution must reside in **labels1.4**.

factor — Find Prime Factors of a Number

General

The **factor** command enables you to calculate the prime factors of any whole number.

Command Format

There are two formats for the **factor** command. These formats are:

factor *number*

factor
number

The *number* argument is the number that you want to factor. The results are the same regardless of what format you use.

Sample Calculations

The following examples show the two different ways to execute the **factor** command. 12 is the number to be factored. 2,2,3 are the prime factors of 12, with the prime factor of 1 not shown.

Factor One Number

```
$ factor 12<CR>
12
      2
      2
      3
$
```

Factor More Than One Number

When factoring more than one number, enter **factor** followed by a carriage return. Then, enter the number you want to factor followed by a carriage return. The answer will then be displayed. To factor another number, simply enter the number followed by a carriage return. The answer will then be displayed. You may continue to factor numbers as long as you like. To quit the **factor** command and return to the UNIX System, enter **q** followed by a carriage return:

```
$ factor<CR>
12<CR>
      2
      2
      3
4<CR>
      2
      2
q<CR>
$
```

logname — Print Login Name

General

The **logname** command displays the contents of the environment variable **$LOGNAME**. The **$LOGNAME** variable is set for a user on login.

Command Format

The **logname** command has no arguments or options.

Sample Commands

The following example shows how to enter the **logname** command and the output that would follow if you were logged in as **root** (#):

```
# logname<CR>
root
#
```

This example shows the output of the **logname** command if you were a normal user logged in as **user5**:

```
$ logname<CR>
user5
$
```

nice — Run a Command at Low Priority

General

The **nice** command executes a command at a lower central processing unit (CPU) priority level.

Command Format

The **nice** command has the following format:

nice *[-increment] command [arguments]*

The *increment* argument specifies the priority level to be lowered for the execution of *command*. The *increment* specified must be between 1 through 19, where the larger the value the lower CPU priority. If no argument is specified for *increment*, an increment of 10 is assumed.

Sample Command

Compiling C programs can consume a large amount of CPU time and reduce the overall system response time. To prevent lose of response time, compilation commands can be given a lower priority level. To use the **nice** command with the **cc** command enter the following:

```
$ nice -19 cc source.c&<CR>
3745
$
```

Note the (**&**) before the (<*CR*>). This causes the compilation to be done in the background. After the <*CR*>, the process identification number (3745) is echoed. You are then returned to the shell while the make command runs as a background process with a low priority.

nohup — Run a Command Immune to Hangups or Quits

General

The **nohup** command allows you to execute a command that is immune to hangups and quits. The use of the ampersand (&) with **nohup** puts the command in the background and allows you to log off the system without terminating the process running under **nohup**. To interrupt a command operating under **nohup**, depress the BREAK key. If the command operating under **nohup** is being executed in the background, the UNIX System command **kill** must be executed. The **kill** command is described in the *AT&T 3B2 Computer User Reference Manual.*

Command Format

The **nohup** command has the following format:

nohup *command [arguments]*

If the output of the command is not redirected in the command line, it is sent to a file called **nohup.out**. If the **nohup.out** file is not writable in the current directory, the output is directed to **$HOME/nohup.out**.

Sample Commands

In the following example, a **diff** command is run under **nohup** and an attempt is made to log off during the processing of the command:

```
$ nohup diff source source2<CR>
Sending output to nohup.out
exit<CR>
$
login:
```

Note that the user was not logged off until the **diff** command was completed. Since the output was not redirected to another file, the output was sent to **nohup.out**.

In this example, the output of the **diff** command is sent to a file called **differ**. The command is run in the background so that the user can log off while the processing of the **diff** command continues:

```
$ nohup diff source source2 > differ&<CR>
4025
$ exit<CR>
login:
```

Immediately after the **nohup** command is entered, the background process number is displayed and the user is returned to the shell. The user can then log off the system without terminating the background process or enter another command.

shl — Layered Shell

General

The **shl** command allows a user to interact with more than one shell from a single terminal. This is done by alternating control of the terminal between several shells that are known as layers. The layer that can accept input from the keyboard is considered the "current layer."

The maximum amount of layers that you can invoke with **shl** is eight. The first layer, however, is used by **shl** to manipulate the other layers. Its prompt is >>>. The other layers have a prompt corresponding to their name. Layer names can have up to eight significant characters, but cannot be the default names **1** through **7** or **(1)** through **(7)**. Every layer has a virtual tty device associated with it and the default name corresponds with the number of this device.

Some examples of what you might want to do in layers would be:

Layer 1 Edit a file.

Layer 2 Use as interactive shell.

Layer 3 Execute a text processing command such as "nroff."

Layer 4 Edit another file.

There are several commands that may be issued from the shl prompt level. These commands allow you to create layers, delete layers, switch between layers, and manipulate the output from each layer. The commands can be abbreviated by any prefix of the command (for example: **c** instead of **create**).

The following list describes the use of each command:

create *[name]*
Create a layer called *[name]* and make it the current layer. If no name is given, a default name is used.

block *name [name...]*
For each *[name]*, block the output of the corresponding layer when it is not the current layer.

delete *name [name...]*
For each *[name]*, delete the corresponding layer.

help or **?**
Print the syntax of the shl commands.

layers *[-l] [name...]*
For each *[name]* list the layer name and its process group ID. If no name is given, information is given for all processes. The **-l** option gives a more detailed listing.

resume *[name]*
Make the referenced layer the current layer.

toggle
Resume the layer that was previously current.

unblock *name [name...]*
For each *[name]*, do not block the output of the corresponding layer when it is not the current layer.

quit Exit shl.

name Make the layer referenced by *name* the current layer. If the name is the same as any prefix of a shl command, the "**resume** *name*" command must be used.

When you are in a layer and wish to return to the **shl**, enter a <*CTRL z*>.

Command Format

The **shl** command has no arguments or options.

Sample Command

The following example shows how to enter the **shl** command, create two more layers, execute some commands in the layers, and then return to the UNIX System:

```
$ shl<CR>
>>> create<CR>
(1) pwd<CR>
/usr/abc
(1) ls<CR>
file.c
memo1
(1) nroff -cm memo1 > formatmemo<CR>
<CTRL z>  >>> c Layer2<CR>
Layer2 pwd<CR>
/usr/abc
Layer2 ed file.c<CR>
36
1,$p<CR>
main()
{
          printf(''hello wold/n'');
}
<CTRL z>  >>> layers -l<CR>
(1) (3650) executing or awaiting input
     UID   PID  PPID  C    STIME TTY      TIME COMMAND
     abc  3669  3650  0 14:24:52 sxt001  0:01 nroff -cm memo1
     abc  3650  3649  0 14:23:50 sxt001  0:00 /bin/sh -i
Layer2 (3700) executing or awaiting input
     UID   PID  PPID  C    STIME TTY      TIME COMMAND
     abc  3705  3690  0 14:25:56 sxt002  0:01 ed file.c
     abc  3690  3649  0 14:25:02 sxt002  0:00 /bin/sh -i
>>> r<CR>
resuming Layer2
3s/wold/world/p<CR>
          printf(''hello world0');
w<CR>
37
q<CR>
Layer2 <CTRL z>  >>> layers -l 1<CR>
(1) (3650) executing or awaiting input
     UID   PID  PPID  C    STIME TTY      TIME COMMAND
     abc  3650  3649  0 14:23:50 sxt001  0:00 /bin/sh -i

>>> q<CR>
$
```

Note: User knows nroff has completed because it is not listed.

tabs — Set Tab Stops on a Printer or Terminal

General

The **tabs** command is used to set tabs on a printer or terminal. Note that the printer must be enabled to set tabs. Also note that not all terminals are compatible with the escape sequences output by the tabs command to clear existing tabs and set new tabs. Executing the command without arguments sets tabs every 8 spaces. Tab stops set every 8 spaces is the standard default setting used by most programs (commands) for high-speed output. Arguments can be provided to the tabs command that specify particular tab stops. Tabs specifications can also be saved in a file.

Command Format

The **tabs** command has the following format:

tabs *[tab specification] [+***m***n] [-***T***type]*

The [*tab specification*] argument specifies the placement of tabs.

The +**m***n* argument specifies the position of the left margin, where *n* is the amount of spaces to be indented. Omitting the *n* from the argument causes a default indent of 10 spaces.

The -**T***type* argument specifies the terminal type. Both the margin and terminal type arguments can be omitted from a command line for most applications.

Canned Tab Specifications

A variety of ready-made tab specifications can be called by the **tabs** command by specifying the appropriate option code. Refer to the **tabs** manual pages in the *AT&T 3B2 Computer User Reference Manual.*

Sample Commands

The following examples show different types of **tabs** command specifications. The following command line sets tab stops for use in writing FORTRAN programs:

```
$ tabs -f<CR>
$
```

A repetitive tab specification sets tab stops every "n" spaces. The following command sets tab stops every 6 spaces:

```
$ tabs -6<CR>
$
```

Tabs can be set at selected points. The following command line sets tabs at 5, 20, 23, and 40 spaces on a line:

```
$ tabs 5,20,23,40<CR>
$
```

A specification line that sets tabs at 20, 25, and 32 is as follows:

<:t20,25,32:>

tty — Print the Terminal Name

General

The **tty** command prints the path name of the user's terminal.

Command Format

The **tty** command has the following format:

tty [*-l*] [*-s*]

The *-l* option prints the synchronous line number of the user's terminal (if connected to a synchronous line).

The *-s* option inhibits printing of the terminal path name, allowing the user to test the exit code.

The exit codes for the **tty** command are different from those of other commands. The exit codes for the **tty** command are as follows:

0	Standard input is a terminal
1	Standard input is from source other than terminal
2	Invalid option(s) were specified.

Sample Commands

The following examples show how to enter the **tty** command and the response that would follow:

```
$ tty<CR>
/dev/tty21
$
```

The *-s* option is used to check the exit code as follows:

```
$ tty -s<CR>
$ echo $?<CR>
0
$
```

The question mark (**?**) is the special shell variable, representing the exit code.

units — Find Unit Conversion Factors

General

The **units** command enables you to find the conversion factors (divisor and multiplier) for converting from one unit of measurement to another related unit of measurement. The types of unit measurements that can be converted include: length, weight, mass, electrical, money, liquid, etc. The file **/usr/lib/unittab** contains a list of units you can convert using the **units** command.

Command Format

The **units** command has the following format:

units
you have: *argument*
you want: *argument*

The **you have:** *argument* identifies the unit of measure that you are using as a reference.

The **you want:** *argument* identifies the unit of measure you want to convert to.

The response will be two different numbers. The first number will be a multiplier that you use as a conversion factor. The second number will be a divisor that can also be used as a conversion factor.

Sample Calculations

You must have a system prompt (**$**) to begin the use of the **units** command. To execute a **units** command, simply enter **units** followed by a carriage return. The system will respond with the display "**you have:**". Enter the unit of measurement you have, followed by a carriage return. The system will then display "you want:". Now, enter the unit of

measurement that you want, followed by a carriage return. The system will display two conversion factors: one will be a multiplier (shown with a * symbol), the other will be a divisor (shown with a / symbol). The system will also display "you have:", so you can continue using the **units** command. To quit the **units** command, press the <*DEL*> or <*BREAK*> key. The system prompt (**$**) will then be displayed to show that you have returned to the UNIX System.

In this example, inch is the unit of measurement you have, and mile is the unit of measurement you want. The answer *1.578283e-05 means to multiply inch by 1.578283 times 10 raised to the -5 power (0.00001578283) to get miles. The answer /6.336000e+04 means to divide inch by 6.336000 times 10 raised to the +4 power (6336.00) to get miles:

```
$ units<CR>
you have: inch<CR>
you want: mile<CR>
        * 1.578283e-05
        / 6.336000e+04
you have: <BREAK>
$
```

If you enter an invalid unit after the system displays "you want:", the system will respond with "cannot recognize". For example:

```
$ units<CR>
you have: inch<CR>
you want: frt<CR>
cannot recognize frt
you want: <BREAK>
$
```

To correct this error, reenter the correct unit after "**you want:**" is displayed again.

If you try to find the conversion factors for unrelated unit measurements, the system will respond with the message "**conformability**" and conversion factors to convert the entered units to related units. This message means the unit conversions you want cannot be done. For example:

```
$ units<CR>
you have: inch<CR>
you want: lbs<CR>
conformability
2.540000e-02m
4.535924e-01kg
you have: <BREAK>
$
```

The response 2.540000e-02m means multiplying an inch by 0.0254 will give you meters (m). The response 4.535924e-01kg means multiplying lbs by 0.4535924 will give you kilograms (kg).

xargs — Construct Argument List(s) and Execute Command

General

The **xargs** command is used to execute a command or a shell program one or more times by combining arguments to the **xargs** command with arguments read from the standard input. Every time a command or a shell program is invoked through the **xargs** command, the amount of arguments read and the way arguments are combined are determined by the flags specified.

Command Format

The **xargs** command has the following format:

xargs *[flags] [command [initial-arguments]]*

The *flag* values are as follows:

-l*[number]*
: *Command* is executed after every *[number]* of lines is read in. Fewer lines of arguments will be used when the last invocation of *command* occurs and there are fewer than *[number]* before the end of the file. A line ends with the first new-line character, unless the last character of the line is a blank or a tab. Here, a line continues onto the next line. If *number* is not specified, "1" is assumed. Option **-x** is automatically used when you use this option.

-i*[replstr]*
: This is the insert mode. *Command* is executed for each line. Each line is considered a single argument and substituted into each occurrence of *[replstr]* in the initial arguments. No more than five initial arguments may contain *[replstr]*. The *[replstr]* can occur more than five times if it occurs more than once in a single argument. Blanks and tabs at the beginning of each line are thrown away. Constructed arguments may not grow larger than

255 characters. Option **-x** is automatically used when you use this option. {} is assumed for *[replstr]* if it is not specified.

-n*[number]*
Command is executed once for every *[number]* of arguments read in. Fewer arguments will be used for the invocation of a command if their total size is greater than *size* characters (see **-s***size* option). Fewer arguments will be used for the last command invocation if there are fewer than *number* arguments before the end of the file. If option **-x** is also used, each *number* arguments must not exceed the *size* limitation or **xargs** will stop execution.

-t This is the trace mode. The *command* and each constructed argument list are echoed to file descriptor 2 just before their execution.

-p This is the prompt mode. You are asked if you want to execute *command* before each invocation. The trace mode (**-t**) is automatically turned on to print *command* to be invoked. A **?...** prompt will follow *command*. A reply of **y** (optionally followed by anything) will execute *command*. Any other response, including a carriage return, will skip that particular invocation of *command*.

-x Causes the **xargs** command to end if any argument list would be greater than *size* characters. This option is used automatically if options **-i** or **-l** are used. The total length of all arguments must be within the *size* limit if options **-i, -l**, or **-n** are not specified.

-s*[size]*
The maximum size of each argument list is set to *size* characters. *Size* must be a positive integer less than or equal to 470. If this option is not specified, 470 is taken as the default. The character count for *size* includes one extra character for each argument and the count of characters in the command name.

-e*[eofstr]*

The *eofstr* is taken as the logical end-of-file string. Underbar (_) is assumed for the logical end-of-file string if this option is not specified. If you use this option with no *eofstr*, the logical end-of-file string capability is turned off (_ is taken literally). The **xargs** command will read the standard input until the end of the file is reached or the logical end-of-file string is encountered.

Command, that may be a shell program, is searched using your **variable PATH**. If *command* is omitted, **/bin/echo** is used.

Arguments read from the standard input are defined to be continuous strings of characters. These strings of characters are delimited by one or more blanks, tabs, or new-lines. Empty lines are always discarded. Blanks and tabs may be embedded as part of an argument if escaped or quoted. To escape the next character, precede that character with a backslash (\). Characters enclosed in quotes (single or double) are taken as written [including backslash (\)] and the delimiting quotes are removed.

Each argument list is constructed of *initial-arguments* followed by some amount of arguments read from the standard input, except when the **-i** flag is used. When the **-i**, **-l**, and **-n** flags are not specified, the *initial-arguments* are followed by arguments read continuously from the standard input. These arguments will continue to be read until an internal buffer is full. After the buffer is full, *command* is executed with the accumulated arguments. When there are conflicts between flags, such as using **-l** and **-n** together, the last flag specified has precedence.

The **xargs** command will end if it receives a return code of **-1**. It will also end if it cannot execute *command*. When *command* is a shell program, **it** should *exit* explicitly with an appropriate value to avoid accidentally returning with a return code of **-1**.

Sample Commands

This example shows how the **xargs** command works when using the **-l** option and designating 5 lines. Since no command argument is given, the output is echoed onto the display:

```
$ xargs -15<CR>
11111<CR>
222<CR>
33<CR>
4444<CR>
5<CR>
11111 222 33 4444 5
AAAA<CR>
BB<CR>
<CTRL d>
AAAA BB
$
```

This example shows how the **xargs** command works when using the **-i** option. The maximum amount of arguments (5) is used:

```
$ xargs -i echo {}a{} b{} c{}{} d{} {}e<CR>
x<CR>
xax bx cxx dx xe
z z<CR>
z zaz z bz z cz zz z dz z z ze
<CTRL d>
$
```

This example shows how to use the **xargs** command to move files to different directories. For each file in dir1, you are asked if you want to move that file to dir2. If you respond with a **y**, the file is moved to dir2. If you give any other response, the file is left in dir1:

```
$ ls dir1 | xargs -i -p mv dir1/{} dir2/{}<CR>
mv dir1/file1 dir2/file1 ?...y<CR>
mv dir1/file2 dir2/file2 ?...y<CR>
mv dir1/file3 dir2/file3 ?...n<CR>
mv dir1/file4 dir2/file4 ?...y<CR>
$ ls dir1<CR>
file3
$ ls dir2<CR>
file1
file2
file4
$
```

The **filetest** program illustrates a simple use of the **xargs** command. The first argument (**$1**) is checked to see if it is a file or a directory. If **$1** is a file, the file contents are printed on your terminal screen. If **$1** is a directory, you are asked if you want to move the files in that directory, one at a time, to the specified directory (**$2**). If **$1** is neither a file nor a directory, a message is printed indicating that **$1** is neither a file nor a directory:

```
$ cat filetest<CR>
# usage: filetest testfile directory
if test -f "$1"          # is $1 a file?
then
cat $1
elif test -d "$1"        # else, is $1 a directory?
then
ls $1 | xargs -i -p mv $1/{} $2/{}
                         # move files from $1 to $2
                         # if response is y
else
echo $1 is neither a file nor a directory
fi
$ ls<CR>
file1
file2
test1
test2
$ filetest test1 test2<CR>
mv test1/cec1 test2/cec1 ?...y<CR>
mv test1/cec2 test2/cec2 ?...n<CR>
mv test1/cec3 test2/cec3 ?...n<CR>
mv test1/cec4 test2/cec4 ?...y<CR>
$ ls test1<CR>
cec2
cec3
$ ls test2<CR>
cec1
cec4
$
```

Replace this

page with the

INDEX

tab separator.

Index

12-pitch ..... TF 2-5
TF 2-7
TF 2-11
TF 2-12
300 filter ..... TF 2-5
300s filter ..... TF 2-5
4014 filter ..... TF 2-8
450 filter ..... TF 2-11

A

abs ..... GR 4-9
accept command ..... LP 3-3
ACCESSING POINTS BY NAME ..... GR 5-9
Accessing the Graphics Editor ..... GR 5-2
ACCESSING THE GRAPHICS UTILITIES ..... GR 2-5
active machine ..... BN 1-3
BN 2-3
ACU; problems ..... BN 6-2
add a new printer ..... LP 3-9
add a printer to a class ..... LP 4-8
LP 4-11
add LP printer, manually ..... LP 4-1
Adding a Term ..... HP 7-6
ADDING AN LP PRINTER ..... LP 4-1
adding change comments ..... SC 3-7

Adding Command Information ..... HP 7-11
adding terminal ..... TI 4-2
TI 4-4
ADDITIONAL INFORMATION ABOUT **get** ..... SC 3-9
ADJUSTING THE SCREEN in vi ..... ED 4-46
admin command ..... SC 2-4
SC 4-3
ADMINISTRATION ..... BN 3-1
LP 4-1
ADMINISTRATION UTILITIES ..... HP 7-1
ADMINISTRATIVE COMMANDS ..... LP 3-1
administrative commands; summary ..... BN 7-3
Administrative Programs ..... BN 2-7
ADMINISTRATIVE TASKS ..... BN 3-33
administrator, SCCS ..... SC 2-5
af ..... GR 4-11
allow print requests ..... LP 3-3
allow printer ..... LP 4-7
American spelling list ..... SP 3-3
Appending Text in edit and ex ..... ED 2-13
Appending Text in vi ..... ED 4-20
ASCII characters ..... TI 6-4
Assign LP System Default Destination ..... LP 4-13
associated terminals ..... TF 1-3
ASSUMPTIONS ..... BN 4-8
at command ..... UE 2-6
AT&T Automatic Dial Modem ..... BN 2-3
attributes ..... TI 2-11
TI 3-10
attributes, low level ..... TI 6-7
attributes, OFF ..... TI 2-12
TI 3-11
attributes, ON ..... TI 2-12
TI 3-11
audible signal ..... TI 2-8
Auditing ..... SC 2-8
automatic call unit (ACU) ..... BN 2-2

B

-b option, using the ..... SP 2-7
-b, **spell** option ..... SP 2-3
background process ..... UE 2-51
backspaces ..... TF 1-2
banner command ..... UE 2-9
bar ..... GR 4-15
Basic Capabilities ..... TI 7-6
BASIC CONCEPTS ..... GR 2-6
Basic Movement Commands ..... ED 2-8
BASIC NETWORKING SOFTWARE ..... BN 2-4
BN 5-7
Basic Networking Utilities; operation ..... BN 2-9
BASIC NETWORKING; WHAT IS ..... BN 2-1
Basic Program ..... SC 2-11
batch command ..... UE 2-11
baudrate ..... TI 5-2
bc command ..... UE 2-13
begin pad ..... TI 3-12
begin window ..... TI 3-1
beginning ..... TI 2-3
bel ..... GR 4-19
bell ..... TI 2-8
blocking message operation ..... IP 2-2
blocking semaphore operation ..... IP 2-5
bolding ..... TF 1-2
box, window ..... TI 3-5
branch number ..... SC 2-16
British spelling ..... SP 2-3
SP 3-3
bucket ..... GR 4-21
Building networks ..... GR 3-8
Bypassing the Help Menu ..... HP 2-6

C

C. (work) file; contents ..... BN 3-3
C. (work) file; description ..... BN 3-3
cal command ..... UE 2-27
calculator, bc ..... UE 2-13
calculator, dc ..... UE 2-35
calendar - print a ..... UE 2-27
calendar file ..... UE 2-29
calendar command ..... UE 2-29
CALLBACK option; **Permissions** file ..... BN 3-23
cancel job request ..... LP 2-5
cancel command ..... LP 2-5
Cartesian plane ..... GR 2-11
cdc command ..... SC 4-47
ceil ..... GR 4-23
ceiling flag ..... SC 2-4
change a tty line ..... LP 4-3
change device ..... LP 4-9
Change Existing Destination ..... LP 4-8
change interface program ..... LP 4-8
change printer device ..... LP 4-8
Change */etc/inittab* file ..... LP 4-2
changing a file ..... SC 3-6
Changing a Term ..... HP 7-6
Changing Command Information ..... HP 7-12
Changing Files in vi ..... ED 4-38
Changing Glossary Information ..... HP 7-6
Changing Starter Information ..... HP 7-3
Changing Text in edit and ex ..... ED 2-14
Changing Text in vi ..... ED 4-23
CHANGING THE HELP DATABASE ..... HP 7-3
Changing the Location of an Object ..... GR 5-16
Changing the Orientation of an Object ..... GR 5-23
Changing the Shape of an Object ..... GR 5-16
Changing the Size of an Object ..... GR 5-18
Changing the Style and Width of Lines ..... GR 5-24
CHARACTER FUNCTIONS SUMMARY in vi ..... ED 4-58
character representation ..... TI 2-24

character, deleting ..... TI 2-10
TI 3-10
character, getting ..... TI 2-15
TI 3-11
character, inserting ..... TI 2-9
TI 2-11
TI 3-9
check status of LP scheduler ..... LP 4-5
class, definition ..... LP 1-4
class directory ..... LP 4-21
CLEANING OUT LOG FILES ..... LP 4-24
cleanup of cron log file ..... BN 3-36
Cleanup of Public Area ..... BN 3-34
cleanup of spool directory ..... BN 2-7
cleanup of sulog file ..... BN 3-36
Cleanup of Undeliverable Jobs ..... BN 3-34
clear partial screen ..... TI 2-6
clear partial window ..... TI 3-8
clear screen ..... TI 2-5
TI 2-21
clear, window ..... TI 3-8
cmpress command ..... CT 2-5
comb command ..... SC 4-53
combining **spell** options ..... SP 2-17
COMMAND ..... GR 5-36
COMMAND DESCRIPTIONS ..... BN 7-1
CT 2-1
GR 4-1
IP 7-1
LP 2-5
LP 3-3
PM 3-7
SA 2-5
SP 2-1
command format ..... GR 2-5
LP 2-3
COMMAND FORMAT ..... GR 5-3
SP 2-3
command line ..... TI 1-2

Command Substitution ..... GR 3-9
COMMAND SUMMARY ..... CT 2-1
LP 2-2
LP 3-2
PM 3-1
SA 2-1
UE 2-6
command syntax ..... TF 2-3
commands ..... TF 2-1
COMMANDS option; **Permissions** file ..... BN 3-24
commands; categories ..... BN 7-1
COMMENT LINES in ex ..... ED 3-14
Comments ..... TI 1-4
Common Functionality ..... BN 4-3
COMMON PROBLEMS ..... BN 6-1
communication link ..... BN 2-2
compacting log files ..... BN 3-35
compatibility, termcap and terminfo ..... TI 7-13
compiling terminal descriptions ..... TI 7-9
Compiling the New Entry ..... TI 7-9
compress program ..... SP 3-2
Concurrent Edits of Different SIDs ..... SC 4-23
Concurrent Edits of Same SID ..... SC 4-27
configure printers ..... LP 3-7
Construct Commands ..... GR 5-36
CONSTRUCTING GRAPHICAL OBJECTS ..... GR 5-5
Continuous Text Input in vi ..... ED 4-21
control character printing ..... TI 2-24
CONTROLLING MESSAGE QUEUES ..... IP 3-21
CONTROLLING SEMAPHORES ..... IP 4-17
CONTROLLING SHARED MEMORY ..... IP 5-21
conventions ..... TI 1-3
copy in (hpio) ..... TF 2-17
copy out (hpio) ..... TF 2-17
copying another file into the buffer in ex ..... ED 3-8
copying another file into the buffer in vi ..... ED 4-36
Copying Objects in vi ..... ED 4-26
Copying Text in edit and ex ..... ED 2-19
COPYING TEXT in vi ..... ED 4-25

copying UNIX System commands into the buffer in vi ..... ED 4-37
cor ..... GR 4-25
corruption repair ..... SC 2-8
Creating a New File using edit ..... ED 2-3
Creating a New File using ex ..... ED 3-3
Creating a New File using vi ..... ED 4-5
CREATING SCCS FILES ..... SC 3-3
Creating SCCS Files ..... SC 4-4
cron log file; cleanup of ..... BN 3-36
crontab entry for **calendar** ..... UE 2-30
crontab entry; uudemon.admin ..... BN 3-37
crontab entry; **uudemon.cleanu** ..... BN 3-37
crontab entry; **uudemon.hour** ..... BN 3-38
crontab entry; **uudemon.poll** ..... BN 3-38
crontab command ..... UE 2-31
cron; how used by UUCP ..... BN 3-36
crypt — encode/decode files ..... SA 2-5
CTC RECOVERY PROCEDURE ..... CT -6
ctccpio command ..... CT 2-9
ctcfmt command ..... CT 2-15
ctcinfo command ..... CT 2-17
ct command; description ..... BN 7-7
ct command; format ..... BN 7-7
ct command; options ..... BN 7-7
ct command; sample ..... BN 7-8
ct program; definition of ..... BN 2-6
ct program; operation of ..... BN 2-10
CURRENT LINE DEFINITION for edit ..... ED 2-2
CURRENT LINE DEFINITION for ex ..... ED 3-2
current screen ..... TI 2-1
current terminal ..... TI 4-2
current terminal, change ..... TI 4-2
TI 4-4
curscr ..... TI 2-1
curses description ..... TI 1-1
CURSES EXAMPLES ..... TI A-1
curses structure ..... TI 2-1
Cursor Movements ..... ED 4-11
cursor optimization ..... TI 2-7

cursor position ..... TI 2-4
TI 2-23
TI 3-8
cursor position, low level ..... TI 6-4
cursor position, window ..... TI 3-12
cur_term ..... TI 6-6
TI 6-7
cusum ..... GR 4-27
cu command; description ..... BN 7-11
cu command; format ..... BN 7-11
cu command; options ..... BN 7-12
cu command; sample ..... BN 7-15
cu command; tilda (~) string interpretations ..... BN 7-12
cu program; definition of ..... BN 2-5
cu program; operation of ..... BN 2-10
cvrtopt ..... GR 4-29

D

D. (data) file; description ..... BN 3-3
Daemons ..... BN 2-7
data (D.) file ..... BN 2-11
data (D.) file; description ..... BN 3-3
data file; how used by **uux** ..... BN 2-6
data files ..... BN 2-6
LP 4-22
dc command ..... UE 2-35
deamons; definition of UUCP ..... BN 2-7
debugging; **Uutry** ..... BN 2-7
debugging; **Uutry** command ..... BN 6-2
default destination ..... LP 4-13
default destination, user defined ..... LP 4-13
default terminal ..... TF 1-2
default file ..... LP 4-19
define system default destination ..... LP 3-7
Defining Capabilities ..... TI 7-4
Defining Your Terminal ..... ED 4-4
DEFINITION OF TERMS ..... BN 1-3

LP 1-4
delay, low level ..... TI 6-8
Delays ..... TI 2-17
delete line ..... TI 2-8
TI 3-9
Deleting a Term ..... HP 7-7
Deleting Command Information ..... HP 7-12
Deleting Text in edit and ex ..... ED 2-15
deleting text in vi ..... ED 4-22
delta ..... SC 3-2
delta command ..... SC 4-33
description characters ..... TI 7-4
description of editors ..... ED 1-1
description symbols ..... TI 7-4
description syntax ..... TI 7-4
destination ..... LP 1-4
destination, change ..... LP 4-8
destination, remove ..... LP 4-14
destination, system default ..... LP 4-13
device, definition ..... LP 1-4
devicemgmt subcommand; simple administration ..... BN 4-5
Devices file entry for direct links ..... BN 5-8
Devices file; definition of ..... BN 2-8
Devices file; description ..... BN 3-5
Devices file; field descriptions ..... BN 3-6
Devices file; format ..... BN 3-6
Devices file; simple administration ..... BN 4-5
device ..... LP 4-6
diagnostics ..... SC 2-3
Dialcodes file; definition of ..... BN 2-9
Dialcodes file; description ..... BN 3-17
Dialcodes file; format ..... BN 3-17
Dialers file; definition of ..... BN 2-9
Dialers file; description ..... BN 3-10
direct link ..... BN 2-2
Direct Link, How to Connect 3B2 to 3B2 ..... BN 5-5
Direct Link, How to Connect 3B2 to 3B20 ..... BN 5-7
Direct Link, How to Connect 3B2 to 3B5 ..... BN 5-7
direct link; **Systems** file entry for ..... BN 5-11

DIRECT LINKS ..... BN 5-1
direct links; benefits ..... BN 5-1
direct links; parts needed ..... BN 5-2
direct links; requirements ..... BN 5-2
direct links; **Devices** file entry ..... BN 5-8
direct links; **inittab** entry for ..... BN 5-9
directories ..... BN 2-4
Directory Organization ..... PM 4-6
disable a tty line ..... LP 4-3
disable printer ..... LP 2-7
disable command ..... LP 2-7
display enhancements (hp) ..... TF 2-15
displaying all characters in vi ..... ED 4-47
DISPLAYING LINES IN THE FILE ..... ED 2-7
ED 3-7
displaying lines in vi ..... ED 4-47
dot (.) command in vi ..... ED 4-34
DRAWING CURVES ..... GR 5-13
DRAWING LINES ..... GR 5-8
Drawings Built From Boxes ..... GR 2-16
dtoc ..... GR 4-31

E

echo ..... TI 2-18
ed, edit, ex, vi — editors in which files can be encrypted and decrypted ..... SA 2-11
Edit Commands ..... GR 5-36
EDIT EDITOR ..... ED 2-1
Editing an Existing File using edit ..... ED 2-6
Editing an Existing File using ex ..... ED 3-6
Editing an Existing File using vi ..... ED 4-8
Editing More Than One Files in ex ..... ED 3-10
Editing Multiple Files and Using Named Buffers in ex ..... ED 3-10
Editing Multiple Files in vi ..... ED 4-39
Editing Objects ..... GR 5-14
editor description (general) ..... ED 1-2
effective user ..... SC 2-2

enable printer ..... LP 4-7
enable command ..... LP 2-9
Entering Glossary Screen 1 (Terms) ..... HP 4-2
Entering Glossary Screen 2 (Definitions) ..... HP 4-4
Entering Locate Screen 1 ..... HP 5-2
Entering Locate Screen 2 ..... HP 5-3
Entering Starter Command Screen ..... HP 3-3
Entering Starter Documents Screen ..... HP 3-5
Entering Starter Education Screen ..... HP 3-6
Entering Starter Local Screen ..... HP 3-7
Entering Starter Screen 1 ..... HP 3-2
Entering Starter Teach Screen ..... HP 3-8
Entering Text ..... ED 2-3
Entering Text in vi ..... ED 4-6
Entering Text with ex ..... ED 3-4
Entering the Help Menu ..... HP 2-4
Entering the 'helpadm' Menu ..... HP 7-1
Entering Usage Description Screen ..... HP 6-4
Entering Usage Example Screen ..... HP 6-5
Entering Usage List Screen ..... HP 6-3
Entering Usage Options Screen ..... HP 6-7
Entering Usage Screen 1 ..... HP 6-2
env command ..... UE 2-45
erase ..... GR 4-33
TI 2-5
erase character ..... TI 5-1
erase, window ..... TI 3-8
Erasing Inserted Text in vi ..... ED 4-20
ERR ..... TI 2-1
ERROR MESSAGES ..... BN 6-3
CT A-1
HP 2-6
LP A-1
error messages; ASSERT ..... BN 6-3
error messages; Status ..... BN 6-7
error, fatal ..... SC 2-3
Errors and Interrupts in ex ..... ED 3-14
errors history ..... SP 3-2
escape characters; **Devices** file ..... BN 3-10

escape characters; **Dialers** file ..... BN 3-11
escape characters; **Systems** file ..... BN 3-16
etc/inittab file, change ..... LP 4-2
Ex Command Line Options ..... ED 3-15
EX EDITOR ..... ED 3-1
Example for Changing Glossary Terms ..... HP 7-7
Example for Modifying Command Data ..... HP 7-13
Example of Changing Starter Data ..... HP 7-4
EXAMPLE OF CREATING MULTIDRAWINGS IN THE SAME UNIVERSE ..... GR 5-32
EXAMPLE OF EDITING A GPS IN THE GRAPHICS EDITOR ..... GR 5-29
EXAMPLE PROGRAM 'editor' ..... TI A-2
EXAMPLE PROGRAM 'highlight' ..... TI A-10
EXAMPLE PROGRAM 'scatter' ..... TI A-12
EXAMPLE PROGRAM 'show' ..... TI A-14
EXAMPLE PROGRAM 'termhl' ..... TI A-16
EXAMPLE PROGRAM 'two' ..... TI A-19
EXAMPLE PROGRAM 'window' ..... TI A-22
Example Program, msgctl ..... IP 3-23
Example Program, msgget ..... IP 3-16
Example Program, msgop ..... IP 3-35
Example Program, semctl ..... IP 4-19
Example Program, semget ..... IP 4-12
Example Program, semop ..... IP 4-32
Example Program, shmctl ..... IP 5-22
Example Program, shmget ..... IP 5-16
Example Program, shmop ..... IP 5-32
EXAMPLES ..... GR 3-20
execute (X.) file ..... BN 2-6
BN 2-8
BN 2-13
execute (X.) file; contents ..... BN 3-4
execute (X.) files; description ..... BN 3-4
executing environment ..... UE 2-45
executing UNIX System commands while in edit and ex ..... ED 2-23
exit codes ..... UE 2-61
exit codes, LP ..... LP 4-17
exiting ..... TI 2-3
Exiting the Editor ..... HP 7-3

exp ..... GR 4-35

F

Facilities ..... IP 1-1
facilities, types of ..... IP 1-1
factor command ..... UE 2-47
FEATURE DESCRIPTION ..... HP 1-2
TI 1-1
feature highlights ..... HP 1-2
FIFO file ..... LP 4-19
Figure 5-3
Accessing the Previous Point Set ..... GR 5-11
Figure 7-2
Administrative Commands ..... BN 7-3
Figure 3-4
Bar Chart Showing Execution Profile ..... GR 3-25
Figure 3-1
Bucket A | hist | td ..... GR 3-18
Figure 5-2
Building a Triangle ..... GR 5-9
Figure 2-3
Cartesian plane ..... GR 2-11
Figure 2-1
Command Summary—Cartridge Tape Utilities ..... CT 2-3
Figure 4-1
Command Summary—Graphics Utilities ..... GR 4-2
Figure 2-1
Command Summary—Line Printer Spooling Utilities ..... LP 2-2
Figure 3-1
Command Summary—LP Spooling Utilities ..... LP 3-2
Figure 3-1
Command Summary—Performance Measurement Utilities ..... PM 3-2
Figure 2-1
Command Summary—Security Administration Utilities ..... SA 2-2
Figure 2-1
Command Summary—Terminal Filter Utilities ..... TF 2-2

Figure 7-1
Command Summary—User Commands ..... BN 7-3
Figure 2-1
Command Summary—User Environment..... UE 2-3
Figure 4-3
Control Commands (Semget Flags) ..... IP 4-10
Figure 5-5
Control Commands (Shmget Flags) ..... IP 5-14
Figure 3-3
Control Commands, msgget ..... IP 3-12
Figure 5-11
Creating a Multidrawing in the Same Screen ..... GR 5-35
Figure 4-1
Determination of New SID ..... SC 4-24
Figure 4-1
Dumb Line Printer Interface Program ..... LP 4-19
Figure A-1
Error Codes—Msgget ..... IP A-3
Figure A-2
Error Codes—Msgctl ..... IP A-5
Figure A-3
Error Codes—Msgsnd ..... IP A-6
Figure A-4
Error Codes—Msgrcv ..... IP A-8
Figure A-5
Error Codes—Semget ..... IP A-10
Figure A-6
Error Codes—Semctl ..... IP A-12
Figure A-7
Error Codes—Semop ..... IP A-13
Figure A-8
Error Codes—Shmget ..... IP A-18
Figure A-9
Error Codes—Shmctl ..... IP A-18
Figure A-10
Error Codes—Shmat ..... IP A-21
Figure A-11
Error Codes—Shmdt ..... IP A-21
Figure 2-2
Evolution of an SCCS File ..... SC 2-15

Figure 5-8
Example of Edit -h1000 ..... GR 5-23
Figure 5-10
Example of Editing a GPS in the Graphics Editor ..... GR 5-32
Figure 5-7
Example of Moving a Circle Using the Move p+ ..... GR 5-22
Figure 3-2
Example of sag Output ..... PM 3-23
Figure 4-1
Example of sag Output ..... PM 4-24
Figure 5-2
Examples of Direct Links ..... BN 5-7
Figure 2-4
Extending the Branching Concept ..... SC 2-17
Figure 5-1
Generating Text Objects ..... GR 5-8
Figure 5-5
Growing a Box ..... GR 5-18
Figure 2-5
Histogram of 100 Random Numbers ..... GR 2-16
Figure 5-13
Making Notes on a Plot ..... GR 5-43
Figure 3-1
Message IPC Organization ..... IP 3-4
Figure 3-2
Operation Permissions Codes ..... IP 3-12
Figure 4-2
Operation Permissions Codes ..... IP 4-9
Figure 5-4
Operation Permissions Codes ..... IP 5-13
Figure 2-6
Output of dtoc Command ..... GR 2-18
Figure 2-7
Output of vtoc Command ..... GR 2-20
Figure 5-14
Page Layout with Drawings and Text ..... GR 5-44
Figure 5-1
Part Numbers for Hardware Used in Directs Links ..... BN 5-3

Figure 4-2
Plot of bar C ¦ td ..... GR 4-17
Figure 4-7
Plot of gas ¦ af ~x^2~ ¦ plot -F-—dg B ¦ td ..... GR 4-90
Figure 4-3
Plot of graph A ¦ tplot ..... GR 4-50
Figure 4-5
Plot of label -Flab,h,r90,y Randplot ¦ td ..... GR 4-68
Figure 4-6
Plot of pie -p piedata ¦ td ..... GR 4-86
Figure 4-8
Plot of ptog B ¦ td ..... GR 4-102
Figure 4-4
Plot of qsort F ¦ bucket ¦ hist ¦ td ..... GR 4-61
Figure 4-9
Plot of siline -n10,s2,i1 ¦ plot ¦td ..... GR 4-119
Figure 4-10
Plot of spline <Z ¦ graph ¦ ptog ¦ td ..... GR 4-126
Figure 4-11
Plot of title -l~lower title~,u~upper title~
Randplot ¦ td ..... GR 4-136
Figure 4-12
Plot of ttoc txt ¦ vtoc ¦ td ..... GR 4-144
Figure 5-4
Referencing Points from Previous Point Set ..... GR 5-12
Figure 3-5
Relationship Between Execution Time and
Number of Processes ..... GR 3-28
Figure 5-9
Rotating Text ..... GR 5-24
Figure 2-1
SCCS Interface Program ..... SC 2-11
Figure 5-6
Scaling Text ..... GR 5-19
Figure 3-2
Scatter Plot ..... GR 3-19
Figure 4-1
Semaphore IPC Organization ..... IP 4-3
Figure 5-1
Shared Memory IPC Organization ..... IP 5-4

Figure 5-2
Shared Memory Segment Descriptor ..... IP 5-7
Figure 5-3
Shared Memory State Information ..... IP 5-7
Figure 2-4
Some Roots of the First Ten Integers ..... GR 2-15
Figure 5-12
Text Centered Within a Circle ..... GR 5-41
Figure 3-3
Transformed Scatter Plot ..... GR 3-20
Figure 2-3
Tree Structure with Branch Deltas ..... SC 2-16
Figure 7-3
cu Command Strings ..... BN 7-14
Figure 7-4
uupick Options ..... BN 7-24
file arguments ..... SC 2-1
FILE MANIPULATION in edit ..... ED 2-21
FILE MANIPULATION in ex ..... ED 3-8
FILE MANIPULATION in vi ..... ED 4-35
file parameters, Initialization and Modification ..... SC 4-7
File System Organization ..... PM 4-5
filename, SCCS ..... SC 2-3
FILES AND DIRECTORIES ..... LP 4-19
filetest program ..... UE 2-72
finc command ..... CT 2-21
Find Command in vi ..... ED 4-18
flags ..... SC 2-2
flash ..... TI 2-8
floor ..... GR 4-37
floppy diskette ..... BN 2-4
flush ..... TI 5-2
flush tty driver ..... TI 2-24
Format of SCCS files ..... SC 2-6
Forward and Backward Search Commands ..... ED 2-9
frec command ..... CT 2-25
function prefixes ..... TI 1-4
FUNCTIONALITY ..... BN 4-2
FUNCTIONS ..... TI 4-4

G

gamma ..... GR 4-39
gas ..... GR 4-41
gd ..... GR 4-43
ged ..... GR 4-45
general options, ipcs ..... IP 7-6
GENERAL PROGRAM FORMAT ..... TI 4-3
General Rules for All Types of Help Screens ..... HP 7-19
GENERATING TEXT ..... GR 5-6
Generator Node ..... GR 3-10
get command ..... SC 3-4
SC 4-11
get, additional information ..... SC 3-9
GETTING MESSAGE QUEUES ..... IP 3-10
GETTING SEMAPHORES ..... IP 4-7
GETTING SHARED MEMORY SEGMENTS ..... IP 5-11
GETTING STARTED ..... ED 2-2
GETTING STARTED with ex ..... ED 3-3
GETTING STARTED with vi ..... ED 4-4
getty process, turn off ..... LP 4-3
Global Searches in edit and ex ..... ED 2-10
Global Searches in vi ..... ED 4-32
Global Substitutes in vi ..... ED 4-33
Global Substitutions in edit and ex ..... ED 2-17
GLOSSARY MODULE ..... HP 4-1
Glossary Screen 1 Options ..... HP 4-2
Glossary Screen 2 Options ..... HP 4-4
GLOSSARY SCREENS ..... HP 4-1
Go to Command in vi ..... ED 4-18
graph ..... GR 4-47
graphics ..... GR 4-51
Graphics Editor ..... GR 5-1
GRAPHICS EDITOR COMMAND DESCRIPTION ..... GR 5-3
Graphics Editors Options ..... GR 5-38
greek filter ..... TF 2-13
gtop ..... GR 4-53
GUIDE BASELINE ..... IN 1-4

GUIDE ORGANIZATION ..... BN 1-2
CT 1-3
DF 1-2
DF 1-2
ED 1-6
GR 1-2
HP 1-4
IP 1-4
LP 1-2
PM 1-4
SA 1-1
SC 1-2
SP 1-2
TF 1-4
TI 1-2
UE 1-2
Guidelines for Description Screens ..... HP 7-20
Guidelines for Examples Screens ..... HP 7-22
Guidelines for Glossary Screens ..... HP 7-19
Guidelines for Options Screens ..... HP 7-21

H

half-line spacing ..... TF 2-6
half-line spacing ..... TF 2-7
hardcopy ..... GR 4-55
HARDWARE ..... BN 2-2
hardwired printer ..... LP 4-9
hashcheck program ..... SP 3-2
hashmake program ..... SP 3-2
help command ..... SC 4-43
help for error codes ..... SC 3-12
HELP MENU ..... HP 2-1
HP 2-4
Help Menu, Options ..... HP 2-5
help menu, screen 1 ..... HP 2-4
Help Screen 1 ..... HP 2-1
HELP UTILITIES TREE ..... HP 2-2
help, SCCS ..... SC 2-3

helpadm Menu ..... HP 7-2
highlighting ..... TI 2-11
TI 3-10
hilo ..... GR 4-57
hist ..... GR 4-59
histogram ..... GR 2-15
hlista file ..... SP 3-3
hlistb file ..... SP 3-3
HOW COMMANDS ARE DESCRIBED ..... BN 7-4
CT 2-3
GR 2-3
HP 1-2
LP 2-3
PM 3-4
SA 2-3
SP 2-1
TF 2-2
UE 2-4
HOW THE DIRECT LINK IS CONNECTED ..... BN 5-5
HOW TO INTERPRET COMMANDS ..... ED 1-5
How to Operate the System Profiler ..... PM 2-5
How to Produce Records and Important Activities ..... PM 2-2
How to Restart Activity Counters From Zero ..... PM 2-2
hp filter ..... TF 2-15
hpd ..... GR 4-63
hpio filter ..... TF 2-17
hstop file ..... SP 3-3

I

-i option, using the ..... SP 2-8
-i, **spell** option ..... SP 2-3
ID Keywords ..... SC 4-13
illegal characters ..... HP 2-6
IMPROVING DISK USEAGE ..... PM 4-3
INITIALIZATION ..... TI 2-3
TI 3-1
Initialization and Modification of SCCS File Parameters ..... SC 4-7

initialization, terminal ..... TI 4-2
TI 4-4
initialization, terminfo level ..... TI 6-2
TI 6-5
inittab entry for direct links; ..... BN 5-9
inittab file ..... BN 3-39
inittab file; simple administration ..... BN 4-7
init program ..... BN 2-7
Input ..... TI 2-15
TI 3-11
INPUT/OUTPUT FUNCTIONS ..... TI 2-4
TI 3-8
input/output functions, window ..... TI 3-8
insert line ..... TI 2-8
TI 3-9
Inserting Commentary, initial delta ..... SC 4-6
Inserting Text in edit and ex ..... ED 2-14
Inserting Text in vi ..... ED 4-20
INTER-PROCESS COMMUNICATION REMOVE ..... IP 7-13
INTER-PROCESS COMMUNICATION STATUS ..... IP 7-2
Interacting with a Data Base ..... GR 3-13
Interacting with Files ..... GR 5-27
Interface Program ..... SC 2-10
interface program ..... SC 2-5
interface program, change ..... LP 4-9
interface program, printer ..... LP 4-15
interface programs ..... LP 4-6
interface directory ..... LP 4-21
INTERFACING THE 5620 DMD TO THE 3B2 COMPUTER ..... GR 2-2
Internal Activity ..... PM 4-7
Internal Programs ..... BN 2-7
introduce a new printer ..... LP 4-6
INTRODUCTION ..... CT 1-1
DF 1-1
GR 1-1
HP 1-1
IN 1-1
PM 1-1
SA 1-1
SP 1-1

IPC ERROR CODES ..... IP A-1
ipcrm ..... IP 7-1
ipcs ..... IP 7-1
Ipcs With Options ..... IP 7-5
Ipcs Without Options ..... IP 7-2
ISSUING UNIX SYSTEM COMMANDS ..... ED 2-23
ED 3-11
ED 4-42

J

Joining Lines in vi ..... ED 4-21

K

KERNEL PROFILING ..... PM 1-3
Keyboard Entered Capabilities ..... TI 7-7
keyletter arguments ..... SC 2-1
keyletter value ..... SC 2-1
Keyletters that Affect Output, get ..... SC 4-28
keypad ..... TI 2-23
kill character ..... TI 5-1

L

-l option, using the ..... SP 2-10
-l, **spell** option ..... SP 2-3
label ..... GR 4-65
LCK. (lock) file; description ..... BN 3-2
LEAVING THE GRAPHICS EDITOR ..... GR 5-35
Leaving the Input Mode in edit ..... ED 2-4
Leaving the Input Mode in ex ..... ED 3-4
leaving the input mode in vi ..... ED 4-6
Leaving the Text Insertion Mode of vi ..... ED 4-6
level number ..... SC 2-15

limited distance modems ..... BN 2-2
Line Numbers in vi ..... ED 4-47
LINE REPRESENTATION IN THE DISPLAY in vi ..... ED 4-47
link (name) ..... SC 2-4
Linking and Use ..... SC 2-13
list ..... GR 4-69
List of Options for vi ..... ED 4-51
Listing All Characters in vi ..... ED 4-47
LOAD MANIPULATION AND HOUSEKEEPING ..... PM 4-30
Local Area Network (LAN) ..... BN 2-3
local machine ..... BN 1-3
local_file option ..... SP 2-3
+local_file option, using the ..... SP 2-14
LOCATE EXAMPLE ..... HP 5-4
LOCATE MODULE ..... HP 5-1
Locate Screen 1 Options ..... HP 5-2
Locate Screen 2 Options ..... HP 5-3
LOCATE SCREENS ..... HP 5-1
lock (LCK.) file; description ..... BN 3-2
lock files ..... LP 4-23
lock-file ..... SC 2-3
log ..... GR 4-71
log files ..... BN 2-7
log files; compacting ..... BN 3-35
log files; description ..... BN 3-5
login; nuucp ..... BN 3-40
login; **uucp** ..... BN 3-40
logname command ..... UE 2-49
log file ..... LP 4-19
longname ..... TI 2-18
low level usage ..... TI 6-1
LOWER LEVEL FUNCTIONS ..... TI 6-1
LOWER LEVEL MODULES ..... HP 2-7
LP exit codes ..... LP 4-17
LP scheduler status ..... LP 3-14
LP Spooling, description ..... LP 1-1
LP status ..... LP 2-15
lpadmin command ..... LP 3-7
LPDEST, environment variable ..... LP 4-13

lpmove command ..... LP 3-11
lpsched command ..... LP 3-13
lpshut command ..... LP 3-15
lpstat command ..... LP 2-15
lp command ..... LP 2-11
lreg ..... GR 4-73

M

MACRO in vi ..... ED 4-48
macros, .so and .nx ..... SP 2-8
make output request ..... LP 2-11
makekey — generate encryption key ..... SA 2-9
MAKING CORRECTIONS TO THE FILE ..... ED 2-13
ED 3-7
MAKING SIMPLE CHANGES in vi ..... ED 4-20
MAN definition ..... TF 1-1
manually add an LP printer ..... LP 4-1
MARKING LINES in vi ..... ED 4-45
Maxuuscheds file; description ..... BN 3-32
Maxuuxqts file; description ..... BN 3-32
mean ..... GR 4-75
member directory ..... LP 4-21
message data structure ..... IP 3-1
MESSAGE ERROR CODES ..... IP A-2
message queue ..... IP 3-1
message queue identifier ..... IP 3-1
MESSAGES ..... IP 2-2
IP 3-1
IP 6-2
mod ..... GR 4-77
Mode Setting ..... TI 2-18
model directory ..... LP 4-15
model interface programs ..... LP 4-6
Model Interface Programs ..... LP 4-15
model directory ..... LP 4-22
modems; limited distance ..... BN 2-2
modems; problems ..... BN 6-2

modification, non-SCCS commands . . . . . SC 2-4
Modifying Command Information . . . . . HP 7-10
Module Contents . . . . . HP 2-7
Module Menus . . . . . HP 2-7
Monitoring the Use of Help . . . . . HP 7-17
move . . . . . TI 2-4
move request to another printer . . . . . LP 3-11
move, in window . . . . . TI 3-8
movement commands in edit and ex . . . . . ED 2-8
MOVING AROUND IN THE FILE . . . . . ED 2-8
ED 3-7
ED 4-10
moving by sentences, paragraphs, and sections in vi . . . . . ED 4-14
Moving Text in edit and ex . . . . . ED 2-20
MOVING TEXT in vi . . . . . ED 4-30
Moving Through a File in vi . . . . . ED 4-14
Moving to Different Lines in vi . . . . . ED 4-13
moving windows . . . . . TI 3-2
Moving Within a Line in vi . . . . . ED 4-11
Msgctl, using . . . . . IP 3-21
Msgget, using . . . . . IP 3-10
MSGMAP . . . . . IP 6-2
MSGMAX . . . . . IP 6-2
MSGMNB . . . . . IP 6-3
MSGMNI . . . . . IP 6-3
Msgop, using . . . . . IP 3-31
MSGSEG . . . . . IP 6-4
MSGSSZ . . . . . IP 6-3
MSGTQL . . . . . IP 6-3
multi-line plot . . . . . GR 2-13
MULTIPLE COMMANDS PER LINE in ex . . . . . ED 3-14
MULTIPLE TERMINALS . . . . . TI 4-1
multiple terminals, format . . . . . TI 4-3
Multiple Windows . . . . . TI 3-6

N

naming a printer ..... LP 4-6
Naming the Terminal ..... TI 7-2
network ..... BN 1-3
newline ..... TI 2-18
nice command ..... UE 2-51
node ..... BN 1-3
BN 2-3
NODE DESCRIPTIONS ..... GR 3-2
nodes command-line format ..... GR 3-2
nohup command ..... UE 2-53
nonblocking message operation ..... IP 2-2
nonblocking semaphore operation ..... IP 2-5
NOREAD option; **Permissions** file ..... BN 3-23
NOWRITE option; **Permissions** file ..... BN 3-23
NROFF definition ..... TF 1-1
null-modem cable ..... BN 5-4
Numerical Manipulation and Plotting ..... GR 2-12
nuucp login ..... BN 3-40

O

Obtaining Information About the Buffer in edit ..... ED 2-22
Obtaining Information About the Buffer in ex ..... ED 3-9
Obtaining Information about the Buffer in vi ..... ED 4-41
OK ..... TI 2-1
oldlog file ..... LP 4-20
Open Text to Insert New Line in vi ..... ED 4-20
OPERATIONS FOR MESSAGES ..... IP 3-31
OPERATIONS FOR SHARED MEMORY ..... IP 5-30
OPERATIONS ON SEMAPHORES ..... IP 4-30
OPTION DESCRIPTION for ex ..... ED 3-15
Option Setting ..... TI 2-21
Organization of File System Free List ..... PM 4-5
Output ..... TI 2-6
outputq file ..... LP 4-20
outputting capabilities, low level ..... TI 6-8

overlay ..... TI 3-2
overstriking ..... TF 1-2
OVERVIEW OF BASIC NETWORKING ..... BN 2-1
OVERVIEW OF GRAPHICS ..... GR 2-1
OVERVIEW OF IPC FACILITIES ..... IP 2-1
OVERVIEW OF SCCS ..... SC 2-1
overwrite ..... TI 3-2

P

p-file ..... SC 3-6
pad ..... TI 3-12
pad initialization ..... TI 3-12
PAD MANIPULATION ..... TI 3-12
pad, begin ..... TI 3-12
padding ..... TI 6-3
TI 6-4
TI 6-7
Paging Through the File in vi ..... ED 4-11
pair ..... GR 4-79
parameter information, low level ..... TI 6-4
Parameters ..... GR 3-6
parameters, system tunable ..... IP 6-1
passive machine ..... BN 1-3
BN 2-3
passwords; assigning ..... BN 3-40
pd ..... GR 4-81
PERFORMANCE TOOLS ..... PM 4-7
permission data structure ..... IP 3-6
Permissions file entries; structure ..... BN 3-18
Permissions file entries; types ..... BN 3-18
Permissions file; CALLBACK option ..... BN 3-23
Permissions file; checked by **uuxqt** ..... BN 2-8
Permissions file; checking ..... BN 2-7
Permissions file; COMMANDS option ..... BN 3-24
Permissions file; Considerations ..... BN 3-19
Permissions file; definition of ..... BN 2-9
Permissions file; description ..... BN 3-18

Permissions file; NOREAD option ..... BN 3-23
Permissions file; NOWRITE option ..... BN 3-23
Permissions file; READ option ..... BN 3-21
Permissions file; REQUEST option ..... BN 3-20
Permissions file; Sample ..... BN 3-29
Permissions file; SENDFILES option ..... BN 3-20
Permissions file; VALIDATE option ..... BN 3-25
Permissions file; WRITE option ..... BN 3-21
pie ..... GR 4-83
pipe ..... GR 3-8
pipe symbol (|) ..... TF 2-3
pipe symbol ..... TI 7-2
pitch switch ..... TF 2-5
plot ..... GR 4-87
plot switch ..... TF 2-5
TF 2-11
point ..... GR 4-91
poll a passive machine ..... BN 2-12
polling ..... BN 2-3
pollmgmt subcommand; simple administration ..... BN 4-6
Poll file; contents of ..... BN 2-12
Poll file; description ..... BN 3-32
Poll file; simple administration ..... BN 4-6
portability functions ..... TI 5-1
portmgmt subcommand; simple administration ..... BN 4-7
power ..... GR 4-93
PREPARING DESCRIPTIONS ..... TI 7-1
preparing terminal descriptions ..... TI 7-6
preprocessor ..... SC 2-11
PREREQUISITES ..... SP 1-2
prevent LP requests ..... LP 3-5
Previous Context Commands in vi ..... ED 4-18
prfdc — Profiler Data Collector ..... PM 3-7
prfld — Profiler Loader ..... PM 3-9
prfpr — Profiler Formatter ..... PM 3-11
prfsnap — Profiler Snapshot Data Collector ..... PM 3-13
prfstat — Profiler Status ..... PM 3-15
prime ..... GR 4-95
print ..... TI 2-4

print job status ..... LP 2-16
print two copies of a file ..... LP 2-14
print, window ..... TI 3-8
PRINTER INTERFACE PROGRAM ..... LP 4-15
printer names ..... LP 4-6
printer, add manually ..... LP 4-1
printer, definition ..... LP 1-4
priority level ..... UE 2-51
PROBLEMS, COMMON ..... BN 6-1
process ID ..... TI 4-1
prod ..... GR 4-97
program structure ..... TI 2-1
protection, flags ..... SC 2-4
protection, user list ..... SC 2-4
prs command ..... SC 4-39
ps ..... PM 4-30
pstatus file ..... LP 4-20
ptog ..... GR 4-99

Q

qsort ..... GR 4-103
qstatus file ..... LP 4-21
queued transfers; controlling ..... BN 2-6
quit ..... GR 4-105
Quitting the edit Editor ..... ED 2-5
Quitting the ex Editor ..... ED 3-5
Quitting the vi Editor ..... ED 4-7

R

rand ..... GR 4-107
rank ..... GR 4-109
READ option; **Permissions** file ..... BN 3-21
Read-Only Mode in ex ..... ED 3-9
Read-Only Mode in vi ..... ED 4-40
Reading an Existing File in vi ..... ED 4-9

Reading Another File Into the Buffer in edit ..... ED 2-21
Reading Another File Into the Buffer in ex ..... ED 3-8
Reading Another File into the Buffer in vi ..... ED 4-36
Reading UNIX System Commands into the Buffer in vi ..... ED 4-37
real user ..... SC 2-2
RECOMMENDATIONS FOR FORMATTING DATA ..... HP 7-19
RECORDING CHANGES ..... SC 3-7
Recovering from Hang-Ups and Crashes in ex ..... ED 3-13
Recovering Lost Files in edit ..... ED 2-24
Recovering Lost Files in ex ..... ED 3-12
Recovering Lost Files in vi ..... ED 4-44
Recovering Lost Lines in vi ..... ED 4-43
RECOVERING LOST TEXT in edit ..... ED 2-24
RECOVERING LOST TEXT in ex ..... ED 3-12
RECOVERING LOST TEXT in vi ..... ED 4-43
redirecting output ..... SP 2-6
redraw pad ..... TI 3-13
redraw screen ..... TI 2-4
redraw window ..... TI 3-5
refreshing the screen in vi ..... ED 4-46
region ..... GR 5-26
reject command ..... LP 3-5
Relations Between vi and ex Editors ..... ED 4-3
release floor flag ..... SC 2-4
release lock flag ..... SC 2-4
release number ..... SC 2-15
remcom ..... GR 4-111
reminder service ..... UE 2-29
remote machine ..... BN 1-3
remote terminal; calling a ..... BN 2-6
remote.unknown; description ..... BN 3-33
removal by ID, ipcrm ..... IP 7-13
removal by key, ipcrm ..... IP 7-14
remove destination ..... LP 3-7
Remove Destination ..... LP 4-14
remove job request from queue ..... LP 2-5
remove printer from a class ..... LP 4-8
remove printers from a class ..... LP 4-11
removing facilities, ipcrm ..... IP 7-13

Removing Text in edit and ex ..... ED 2-15
Removing Text in vi ..... ED 4-22
removing windows ..... TI 3-1
REPEATING COMMANDS in vi ..... ED 4-34
Repeating Searches in edit and ex ..... ED 2-10
Repeating Searches in vi ..... ED 4-16
replacing text in vi ..... ED 4-23
request files ..... LP 4-22
request ID ..... LP 2-11
REQUEST option; **Permissions** file ..... BN 3-20
requests, enable printing ..... LP 2-9
request directory ..... LP 4-22
REQUIREMENTS ..... PM 2-1
restore tty modes ..... TI 4-4
Restoring Good File System Organization ..... PM 4-6
RESTRICTIONS ..... ED 1-3
Retrieval of Different Versions ..... SC 4-15
Retrieval with Intent to Make a Delta ..... SC 4-19
retrieve file from public area ..... BN 2-12
RETRIEVING A FILE ..... SC 3-5
Retrofit-graphics terminals ..... TF 1-3
rmdel command ..... SC 4-45
root ..... GR 4-113
root crontab file ..... BN 3-36
round ..... GR 4-115

S

sa1 — System Activity Report Package ..... PM 3-35
sa2 — System Activity Report Package ..... PM 3-37
sact command ..... SC 4-57
sadc — System Activity Data Collector ..... PM 3-17
sadp ..... PM 4-27
sadp — Disk Access Profiler ..... PM 3-19
sag ..... PM 4-24
Sag Requirements ..... PM 2-7
sag — System Activity Graph ..... PM 3-21
SAMPLE COMMAND LINES ..... SP 2-4

Sample **Permissions** Files ..... BN 3-29
sar ..... PM 4-8
sar -a ..... PM 4-9
sar -A ..... PM 4-23
sar -b ..... PM 4-10
sar -c ..... PM 4-12
sar -d ..... PM 4-14
sar -m ..... PM 4-16
sar -q ..... PM 4-17
sar -u ..... PM 4-18
sar -v ..... PM 4-19
sar -w ..... PM 4-21
sar -y ..... PM 4-22
sar — System Activity Reporter ..... PM 3-25
save tty modes ..... TI 4-4
saving changes to the buffer in vi ..... ED 4-6
scanning ..... TI 2-16
TI 3-12
Scatter plot ..... GR 3-15
SCCS COMMAND DESCRIPTIONS ..... SC 4-1
SCCS FILES ..... SC 2-4
SCCS FOR BEGINNERS ..... SC 3-1
sccsdiff command ..... SC 4-51
SCHEDLOCK file ..... LP 3-13
LP 4-23
scheduler status ..... LP 2-16
screen displays, conventions ..... LP 2-4
SCREEN MANIPULATION ..... TI 2-1
scroll ..... TI 2-18
SCROLL variable ..... HP 2-3
Scrolling Through the File in vi ..... ED 4-11
searching for a pattern of characters in vi ..... ED 4-15
searching for text in edit and ex ..... ED 2-9
Selecting Tunable Parameters ..... PM 4-3
SEMAEM ..... IP 6-7
semaphore array operations ..... IP 2-5
semaphore data structure ..... IP 4-1
SEMAPHORE ERROR CODES ..... IP A-8
semaphore identifier ..... IP 4-1

semaphore set (array) ..... IP 4-1
semaphore undo structures ..... IP 2-6
SEMAPHORES ..... IP 2-4
IP 4-1
IP 6-5
semaphores, decrementing ..... IP 2-4
semaphores, incrementing ..... IP 2-4
semaphores, testing for zero ..... IP 2-5
Semctl, using ..... IP 4-17
Semget, using ..... IP 4-7
SEMMAP ..... IP 6-5
SEMMNI ..... IP 6-6
SEMMNS ..... IP 6-6
SEMMNU ..... IP 6-6
SEMMSL ..... IP 6-6
Semop, using ..... IP 4-30
SEMOPM ..... IP 6-6
SEMUME ..... IP 6-7
SEMVMX ..... IP 6-7
SENDFILES option; **Permissions** file ..... BN 3-20
seqfile file ..... LP 4-21
sequence number ..... SC 2-16
sequence operator (semicoln) ..... GR 3-9
SETTING KERNEL CONFIGURATION PARAMETERS ..... PM 4-2
setting options ..... TI 2-21
Setting Options for vi and ex editors ..... ED 4-50
Setting Terminal ..... TI 2-18
Setting Text-bit (Sticky-bits) ..... PM 4-4
SETTING UP YOUR TERMINAL ..... HP 2-3
Setting Up Your Terminal Configuration ..... ED 4-4
SHARED MEMORY ..... IP 2-7
IP 5-1
IP 6-9
shared memory attach ..... IP 2-7
shared memory data structure ..... IP 5-1
shared memory detach ..... IP 2-7
SHARED MEMORY ERROR CODES ..... IP A-16
shared memory identifier ..... IP 5-1
shared memory operations ..... IP 2-7

shared memory segment ..... IP 5-1
shl command ..... UE 2-55
SHMALL ..... IP 6-10
Shmctl, using ..... IP 5-21
Shmget, using ..... IP 5-11
SHMMAX ..... IP 6-9
SHMMIN ..... IP 6-9
SHMMNI ..... IP 6-9
Shmop, using ..... IP 5-30
SHMSEG ..... IP 6-10
SID, identification string ..... SC 3-2
siline ..... GR 4-117
Simple Administration ..... BN 2-4
simple administration; assumption ..... BN 4-8
simple administration; functionality ..... BN 4-2
simple administration; **devicemgmt** subcommand ..... BN 4-5
simple administration; **pollmgmt** subcommand ..... BN 4-6
simple administration; **portmgmt** subcommand ..... BN 4-7
simple administration; **systemmgmt** subcommand ..... BN 4-4
sin ..... GR 4-121
Software Programs and Their Purpose ..... BN 2-5
spacing switch ..... TF 2-11
special characters ..... TF 1-2
TI 6-4
SPECIAL NOTATIONS ..... TI 1-3
SPECIAL PURPOSE KEYS ..... ED 1-4
Special Search Characters in edit and ex ..... ED 2-11
Special Search Characters in vi ..... ED 4-17
Special Substitute Characters in edit and ex ..... ED 2-17
Speeding up Things ..... GR 5-35
spell filename ..... SP 2-3
SPELL UTILITIES ADMINISTRATION ..... SP 3-1
spellhist file ..... SP 3-2
spellin program ..... SP 3-3
spell options ..... SP 2-3
spell, by itself ..... SP 2-4
spell, on text files ..... SP 2-5
spline ..... GR 4-123
spool a print request ..... LP 2-11

spool directory; reorganization ..... BN 3-34
spool file system; out of space ..... BN 6-1
spooling ..... LP 1-1
standard screen ..... TI 2-1
start LP scheduler ..... LP 3-13
start LP scheduler ..... LP 4-7
STARTER MODULE ..... HP 3-1
STARTER SCREENS ..... HP 3-2
Stat ..... GR 2-12
GR 3-1
status of LP system ..... LP 2-15
status report ..... LP 2-15
status, ipcs ..... IP 7-2
stdscr ..... TI 2-1
stop currently printing request ..... LP 2-6
stop LP scheduler ..... LP 3-15
LP 4-5
stop printing requests ..... LP 2-7
stop request from printing ..... LP 2-5
string, getting ..... TI 2-16
TI 3-12
string, inserting ..... TI 2-9
TI 3-10
STRUCTURE ..... TI 2-1
SUBCOMMANDS ..... BN 4-4
subscripts ..... TF 2-6
subset ..... GR 4-127
Substituting Text in edit and ex ..... ED 2-16
substituting text in vi ..... ED 4-23
successor deltas ..... SC 2-15
sulog file; cleanup of ..... BN 3-36
Summarizers Node ..... GR 3-6
summary of vi character functions ..... ED 4-58
supersubscripts ..... TF 2-6
supporting data base ..... BN 2-4
Supporting Data Base ..... BN 2-8
supporting data base; location ..... BN 3-5
SUPPORTING DOCUMENTATION ..... LP 1-3
syntax ..... UE 2-4

sysadm command ..... LP 4-2
SYSTEM ACTIVITY ..... PM 1-2
System Administration menu ..... LP 4-2
system calls ..... IP 1-2
system calls, categories of ..... IP 1-2
system calls, naming of ..... IP 1-2
system default destination ..... LP 4-13
LP 4-19
system prompt ..... BN 7-5
SYSTEM TUNABLE PARAMETERS ..... IP 6-1
systemmgmt subcommand; simple administration ..... BN 4-4
Systems file entry for direct link ..... BN 5-11
Systems file; definition of ..... BN 2-9
Systems file; description ..... BN 3-13
Systems file; field descriptions ..... BN 3-13
Systems file; format ..... BN 3-13
Systems file; simple administration ..... BN 4-4
Systems file; updating ..... BN 6-2

T

tabs command ..... UE 2-59
tar command ..... CT 2-27
td ..... GR 4-129
tee ..... GR 3-8
tekset ..... GR 4-131
telephone network ..... BN 2-2
temporary data files (TM.); description ..... BN 3-2
TERM ..... TF 2-13
TI 4-1
TI 4-2
TI 6-5
TERM variable ..... HP 2-3
termcap ..... TI 7-4
TI 7-13
TERMCAP AND TERMINFO COMPATIBILITY ..... TI 7-13
terminal ..... TI 7-10
terminal configuration ..... ED 4-4

terminal definition ..... ED 4-4
terminal descriptions, capabilities ..... TI 7-4
terminal descriptions, compiling ..... TI 7-9
terminal descriptions, preparing ..... TI 7-6
terminal filter definition ..... TF 1-1
Terminal Mode Setting ..... TI 2-18
terminal name ..... TI 2-18
terminal naming ..... TI 7-2
terminal speed ..... TI 5-2
terminal, initialization ..... TI 4-2
TI 4-4
terminfo database ..... TI 7-13
TERMINFO DATABASE, description ..... TI 7-1
terminfo description ..... TI 1-1
terminfo level ..... TI 6-1
Terminfo Level ..... TI 6-4
terminfo level, begin ..... TI 6-2
TI 6-5
TERMINOLOGY ..... SC 3-2
test call processing ..... BN 2-7
Testing an Entry ..... TI 7-10
testing terminal descriptions ..... TI 7-10
The Concept of Yank and Put in vi ..... ED 4-25
THE **help** COMMAND ..... SC 3-12
tic — terminfo compiler ..... TI 7-9
timex ..... PM 4-25
Timex Requirements ..... PM 2-3
timex — Time a Command; Report Process Data
and System Activity ..... PM 3-39
title ..... GR 4-133
TM. file; temporary data files ..... BN 3-2
total ..... GR 4-137
tparm ..... TI 6-4
TI 6-7
tplot ..... GR 4-139
TPUT COMMAND ..... TI 7-11
tputs ..... TI 6-3
TI 6-7
Transformer Node ..... GR 3-3

Translators Node ..... GR 3-14
ttoc ..... GR 4-141
tty management menu ..... LP 4-3
tty modes ..... TI 4-4
TI 6-6
tty command ..... UE 2-61
TUNING A 3B2 COMPUTER SYSTEM ..... PM 4-1
turn off getty process ..... LP 4-3
turn off LP scheduler ..... LP 4-5
turn off tty line ..... LP 4-3
type options, ipcs ..... IP 7-5
TYPICAL ADMINISTRATIVE TASKS ..... LP 4-8
Typing Ahead ..... GR 5-35

U

underline (hp) ..... TF 2-15
underlines ..... TF 1-2
Undoing the Last Command in edit ..... ED 2-24
Undoing the Last Command in ex ..... ED 3-12
Undoing the Last Command in vi ..... ED 4-43
unget command ..... SC 4-31
units command ..... UE 2-63
universe ..... GR 2-11
update pad ..... TI 3-13
update screen ..... TI 2-4
update window ..... TI 3-5
updates, simultaneous ..... SC 2-3
Usage Description Screen Example ..... HP 6-11
Usage Description Screen Options ..... HP 6-5
Usage Example Screen Options ..... HP 6-6
Usage Examples Screen Example ..... HP 6-12
Usage List Screen Example ..... HP 6-10
Usage List Screen Options ..... HP 6-4
USAGE MODULE ..... HP 6-1
USAGE MODULE EXAMPLES ..... HP 6-9
USAGE MODULE SCREENS ..... HP 6-2
Usage Options Screen Example ..... HP 6-13

Usage Options Screen Options ..... HP 6-8
Usage Screen 1 Example ..... HP 6-9
Usage Screen 1 Options ..... HP 6-3
User $PATH Variables ..... PM 4-31
USER COMMANDS ..... BN 7-7
LP 2-1
user commands; summary ..... BN 7-1
user defined default destination, **LPDEST** ..... LP 4-13
user list ..... SC 2-5
SC 2-11
User Programs ..... BN 2-5
Using Named Buffers in vi ..... ED 4-39
Utilities ..... IP 1-3
IP 7-1
utilities, naming of ..... IP 1-3
uucheck command; description ..... BN 7-41
uucheck command; format ..... BN 7-41
uucheck command; options ..... BN 7-41
uucheck command; sample ..... BN 7-42
uucheck program; definition of ..... BN 2-7
uucico daemon; definition of ..... BN 2-7
uucico daemon; used by **uucp** ..... BN 2-11
uucleanup command; description ..... BN 7-37
uucleanup command; format ..... BN 7-37
uucleanup command; options ..... BN 7-37
uucleanup command; sample ..... BN 7-38
uucleanup program; definition of ..... BN 2-7
UUCP ..... BN 1-3
uucppublic directory; used by **uuto** ..... BN 2-6
uucp command; description ..... BN 7-17
uucp command; format ..... BN 7-17
uucp command; options ..... BN 7-18
uucp command; sample ..... BN 7-19
uucp login ..... BN 3-40
uucp program; definition of ..... BN 2-6
uucp program; operation of ..... BN 2-11
uudemon.admin ..... BN 3-34
uudemon.admin; crontab entry ..... BN 3-37
uudemon.admin; description ..... BN 3-36

uudemon.cleanu ..... BN 3-34
uudemon.cleanu; crontab entry ..... BN 3-37
uudemon.cleanu; description ..... BN 3-37
uudemon.hour ..... BN 2-12
uudemon.hour; crontab entry ..... BN 3-38
uudemon.hour; description ..... BN 3-37
uudemon.poll; crontab entry ..... BN 3-38
uudemon.poll; description ..... BN 3-38
uugetty program; definition of ..... BN 2-7
uulog command; description ..... BN 7-31
uulog command; format ..... BN 7-31
uulog command; options ..... BN 7-31
uulog command; sample ..... BN 7-32
uulog program; definition of ..... BN 2-7
uuname command; description ..... BN 7-35
uuname command; format ..... BN 7-35
uuname command; sample ..... BN 7-35
uupick command; description ..... BN 7-23
uupick command; format ..... BN 7-23
uupick command; options ..... BN 7-23
uupick command; sample ..... BN 7-24
uupick program; definition of ..... BN 2-6
uusched daemon ..... BN 2-12
uusched daemon; definition of ..... BN 2-8
uustat command; description ..... BN 7-33
uustat command; format ..... BN 7-33
uustat command; options ..... BN 7-33
uustat command; sample ..... BN 7-34
uustat program; definition of ..... BN 2-6
uuto command; description ..... BN 7-21
uuto command; format ..... BN 7-21
uuto command; options ..... BN 7-22
uuto command; sample ..... BN 7-22
uuto program; called by **uucp** ..... BN 2-12
uuto program; definition of ..... BN 2-6
uuto program; operation of ..... BN 2-12
Uutry command; description ..... BN 7-39
Uutry command; format ..... BN 7-39
Uutry command; options ..... BN 7-39

Uutry command; sample ..... BN 7-40
Uutry program; definition of ..... BN 2-7
uuxqt daemon; definition of ..... BN 2-8
uux command; description ..... BN 7-27
uux command; format ..... BN 7-27
uux command; options ..... BN 7-27
uux command; sample ..... BN 7-29
uux program; definition of ..... BN 2-6
uux program; operation of ..... BN 2-13

V

-v option, using the ..... SP 2-12
-v, **spell** option ..... SP 2-3
val command ..... SC 4-55
VALIDATE option; **Permissions** file ..... BN 3-25
var ..... GR 4-145
variables ..... TI 1-3
vc command ..... SC 4-59
Vector(s) ..... GR 3-3
vectors ..... GR 2-12
GR 3-10
video attributes ..... TI 2-11
TI 3-10
VIEW COMMANDS ..... GR 5-25
View Commands ..... GR 5-37
VISUAL EDITOR (vi) ..... ED 4-1
vtoc ..... GR 4-147

W

what command ..... SC 4-49
whatis ..... GR 4-149
window initialization ..... TI 3-1
window, begin ..... TI 3-1
window, delete ..... TI 3-1
window, moving ..... TI 3-2

WINDOWING ..... GR 5-26
windows ..... TI 3-1
WORD ABBREVIATIONS in vi ..... ED 4-46
work (C.) file ..... BN 2-11
work (C.) file; contents ..... BN 3-3
work (C.) file; description ..... BN 3-3
work files ..... BN 2-6
WRITE option; **Permissions** file ..... BN 3-21
Writing Interface Programs ..... LP 4-15
Writing the Buffer into the File in edit ..... ED 2-4
Writing the Buffer into the File in ex ..... ED 3-5
Writing the Buffer into the File in vi ..... ED 4-6
Writing the Buffer to Another File in edit ..... ED 2-21
Writing the Buffer to Another File in ex ..... ED 3-8
Writing the Buffer to Another File in vi ..... ED 4-35

X

-x option, using the ..... SP 2-13
-x, **spell** option ..... SP 2-3
x-file ..... SC 2-3
X. (execute) file ; description ..... BN 3-4
X. (execute) file; contents ..... BN 3-4
xargs command ..... UE 2-67

Y

yoo ..... GR 4-151

Z

z-file ..... SC 2-3