IBM RT PC Advanced Interactive Executive Operating System Version 2.1

Managing the AIX Operating System

Programming Family

IBM

Personal
Computer
Software

SC23-0793-0

IBM RT PC Advanced Interactive Executive Operating System Version 2.1

Managing the AIX Operating System

Programming Family

Personal
Computer
Software

First Edition (January 1987)

Portions of the code and documentation described in this book were developed at the Electrical Engineering and Computer Sciences Department at the Berkeley Campus of the University of California under the auspices of the Regents of the University of California.

This edition applies to Version 2.1 of IBM RT PC AIX Operating System Licensed Program, and to all subsequent releases until otherwise indicated in new editions or technical newsletters. Changes are made periodically to the information herein; these changes will be reported in technical newsletters or in new editions of this publication.

About This Book

A computer system has two main parts:

Hardware The physical components of the system.

Software The programs that control how the hardware works.

On the IBM RT Personal Computer[1] system, there are two types of software:

The *operating system*
: A set of programs that controls how the system works. Some of the tasks an operating system usually performs are allocating system resources, scheduling operations, and controlling the flow of data through the system.

Application programs
: Software that performs a particular task such as word processing, project planning, or inventory control.

The AIX[1] Operating System consists of a ***kernel*** (the programs that control how the system works) and a ***shell*** or (***command interpreter***). The shell provides a set of ***commands*** (programs) that cause the system to perform specific operations. The subject of this book is the AIX Operating System—a part of the RT PC system that exists between the hardware and the application programs. To determine whether you want to work with the AIX Operating System as this book describes, please read the remainder of this section.

1 RT, RT PC, RT Personal Computer, and AIX are trademarks of International Business Machine Corporation.

The AIX Operating System is based on UNIX[2] System V, which consists of a kernel and a standard UNIX shell (the Bourne shell). Your RT PC system may have additional command interpreters installed, for example:

- **csh** (described under **csh** in *AIX Operating System Commands Reference*
- **DOS Services** (described in *Using AIX Operating System DOS Services*)
- **Usability Services** (described in *Usability Services Guide*).

This book explains how to manage the AIX system. ***System management*** is a broad term for all of the tasks required to adapt the AIX Operating System to your needs and to keep the system in good working order. Among the topics included are:

- Setting up user accounts of different types
- Creating, using, and maintaining file systems
- Making backup copies of information stored on the system
- Using system accounting facilities.

Who Should Read This Book

This book is written for those who are responsible for managing an AIX system. If more than one person uses your system, system management responsibilities may be given to one person or shared among several. If you are the only person using your system, you still must perform certain system management tasks.

2 UNIX was developed and licensed by AT&T. It is a registered trademark of AT&T in the United States of America and other countries.

Before You Begin

Before you can begin to work with this book, your RT PC system must be set up and the AIX Operating System must be installed. In addition, you should have a **user name** and possibly a password. If your system is not installed, see *Installing and Customizing the AIX Operating System*. If you need a user name, see "Creating, Changing, and Removing Accounts–The users Command" on page 2-16.

Note: Many of the tasks in this book require you to use one of the RT PC text editing programs. The following editing programs are available on the RT PC system:

- **ed** (see *Using the AIX Operating System*)
- **INed**[3] (see *INed*)
- **vi** (see *AIX Operating System Commands Reference*)

How to Use This Book

This book is divided into chapters, with each chapter devoted to a major AIX Operating System concept or feature:

- Chapter 1, "Introduction to System Management" on page 1-1 includes background information necessary for effective system management. It explains the structure of the system in general, and the file system in particular, from the viewpoint of someone responsible for managing and maintaining the AIX Operating System.
- Certain system management tasks must be performed routinely. Chapter 2, "Routine System Management" on page 2-1 explains such common system management tasks as starting the system, establishing user accounts, creating and using file systems, and making backup copies of data stored in file systems.

[3] INed is a registered trademark of INTERACTIVE Systems Corporation

- In addition to the routine system management tasks, there are other tasks that you may need to perform periodically, as well as other concepts that you should understand in order to make your system as efficient and dependable as possible. Chapter 3, "Maintaining the AIX Operating System" on page 3-1 explains several of these system maintenance tasks, including the very important task of maintaining the file system. This chapter also contains explanatory information about the AIX Operating System input/output, queueing, and error-reporting systems.

- Chapter 4, "Additional System Management Topics" on page 4-1 is an assortment of other useful system management information. Among the topics in this chapter are ways to communicate with system users, how to run commands at pre-determined times, and some considerations about system security. You may find it helpful to simply browse through this chapter, looking for features or capabilities that you can use to make your system better suit your needs.

- The AIX Operating System includes an accounting system that allows you to collect data about how your system is being used. The accounting system, described in Chapter 5, "Running System Accounting" on page 5-1, is most useful on systems that have more than one user, especially those with more than one display station attached.

- Chapter 6, "Using the System Activity Package" on page 6-1 describes the AIX Operating System facilities for measuring and reporting input and output, processing unit utilization, file system access, and other system activity. This chapter explains how the system activity package works, what its capabilities are, and how to use it to generate daily system activity reports.

In addition, this book also includes supplemental information in the appendixes, and a glossary and an index to make it easier for you to find information.

A Reader's Comment Form and Book Evaluation Form are provided at the back of this book. Use the Reader's Comment Form at any time to give IBM information that may improve the book. After you become familiar with the book, use the Book Evaluation Form to give IBM specific feedback about the book.

You can use this book in one of two ways:

- As a training manual. Read and work through it from the first chapter to the last one. This should give you a general understanding of the AIX Operating System.

- As a reference manual. Use the "Contents" and "Index" to locate particular topics. This is a good way to refresh your memory or learn more details about the AIX Operating System.

Special Features

This book uses type style to distinguish among kinds of information. General information is printed in the standard type style (the type style used for this sentence). The following type styles indicate other types of information:

New terms
Each time a new term is introduced, its first occurrence is printed in this type style (for example, "the AIX Operating System ***file system***").

System parts
The names for keys, commands, files, and other parts of the system are printed in this type style (for example, "the **cp** command").

Variable information
The names for information that you must provide are printed in this type style (for example, "type *yourname*").

`Special characters`
Any characters that have a special meaning are printed in this type style (for example, "the `&` and `&&` operators have different uses"). This type is also used for the names of files that you create as you work through this book (for example, "create a file named `afile`").

`Information you are to type`
Many examples in this book are designed for you to try them on your own system; the information that you

should type is printed in this type style (for example, "type `ls text` and press **Enter**").

Where appropriate, the chapters and major sections of this book begin with a box containing quick reference material, for example:

To Use a Quick Reference Box

1. Skip the quick reference boxes the first time you read a section.
2. Use the quick reference boxes as a fast path through the book.
3. Refer to the quick reference boxes to refresh your memory.

You should use the boxes for reference after you are generally familiar with the contents of a section or chapter. You can skip the box the first time you read a section. The boxes make a convenient ***fast path*** through the book, but they are not comprehensive and they are not intended to take the place of the explanatory material in each section.

After the quick reference boxes, each chapter in this book takes the same general approach to the topics it covers—a series of explanations and examples. The examples build upon each other; in many instances, an example uses a file created in a previous example. Therefore, if you intend to follow the examples on your system, it is important for you to work through each chapter from start to finish.

In the examples, the characters you should type are printed in blue, as this example shows:

```
$ ls
  afile
  bfile
  cfile
$ _
```

After you type the characters on a line, press the **Enter** key.

In the text, whenever you are told to ***enter*** a command or other information, you should type the information and then press the **Enter** key.

Related Books

- *IBM RT PC Installing and Customizing the AIX Operating System* provides step-by-step instructions for installing and customizing the AIX Operating System, including how to add or delete devices from the system and how to define device characteristics. This book also explains how to create, delete, or change AIX and non-AIX minidisks.

- *IBM RT PC Using the AIX Operating System* describes using the AIX Operating System commands, working with file systems, and developing shell procedures.

- *IBM RT PC AIX Operating System Commands Reference* lists and describes the AIX Operating System commands.

- *IBM RT PC Guide to Operations* describes the IBM 6151 and IBM 6150 system units, the displays, keyboard, and other devices that can be attached. This guide also includes procedures for operating the hardware and moving the IBM 6151 and IBM 6150 system units.

- *IBM RT PC Problem Determination Guide* provides instructions for running diagnostic routines to locate and identify hardware problems. A problem determination guide for software and three high-capacity (1.2MB) diskettes containing the IBM RT PC diagnostic routines are included.

- *IBM RT PC Usability Services Guide* shows how to create and print text files, work with directories, start application programs, and do other basic tasks with Usability Services. (Packaged with *Usability Services Reference*)

- *IBM RT PC Usability Services Reference* supplements *IBM RT PC Usability Services Guide* by including information on using all of the Usability Services commands. (Packaged with *Usability Services Guide*)

- *IBM RT PC Exploring Usability Services* is an online tutorial for first-time users of the Usability Services. This tutorial simulates the user interface and shows how to use the keyboard and the optional mouse, how to manipulate windows, and how to use files and directories.

- *IBM RT PC Messages Reference* lists messages displayed by the IBM RT PC and explains how to respond to the messages.

- *IBM RT PC Using AIX Operating System DOS Services* provides step-by-step information for using AIX Operating System shell. (Available optionally; packaged with *IBM RT PC AIX Operating System DOS Services Reference*)

- *IBM RT PC AIX Operating System Technical Reference* describes the system calls and subroutines that a C programmer uses to write programs for the AIX Operating System. This book also includes information about the AIX file system, special files, file formats, GSL subroutines, and writing device drivers. (Available optionally)

- *IBM RT PC AIX Operating System Communications Guide* provides information and customizing details on communicating with other users in a multiple work station environment and in two system-to-system configurations. One configuration links a single interactive work station with a remote host, and the other links two AIX-based systems for batch file transfer.

- *IBM RT PC INed* provides guide and reference information for using the INed program to create and revise files.

- *IBM RT PC AIX Operating System Programming Tools and Interfaces* describes the programming environment of the AIX Operating System and includes information about using the operating system tools to develop, compile, and debug programs. In addition, this book describes the operating system services and how to take advantage of them in a program. This book also includes a diskette that includes programming examples, written in C language, to illustrate using system calls and subroutines in short, working programs. (Available optionally)

- *IBM RT PC SNA Services Guide and Reference* describes the capabilities and functions provided by the IBM RT PC Systems Network Architecture (SNA) Services when it is installed on an IBM RT PC. It includes information on the structure and operation of SNA (Systems Network Architecture) on the RT PC, configuring a network, controlling SNA on the local system, finding problems associated with SNA Services, programming using the SNA Services application programming interface, and using the Usability Services extensions that are active when SNA Services is installed. (Available optionally)

- *IBM RT PC Interface Program for use with TCP/IP* describes the Interface Program commands for transferring data among host computers, logging into remote computers, and managing networks. This book also describes the programming interfaces to the Interface Program.

Ordering Additional Copies of This Book

To order additional copies of this publication (without program diskettes), use either of the following sources:

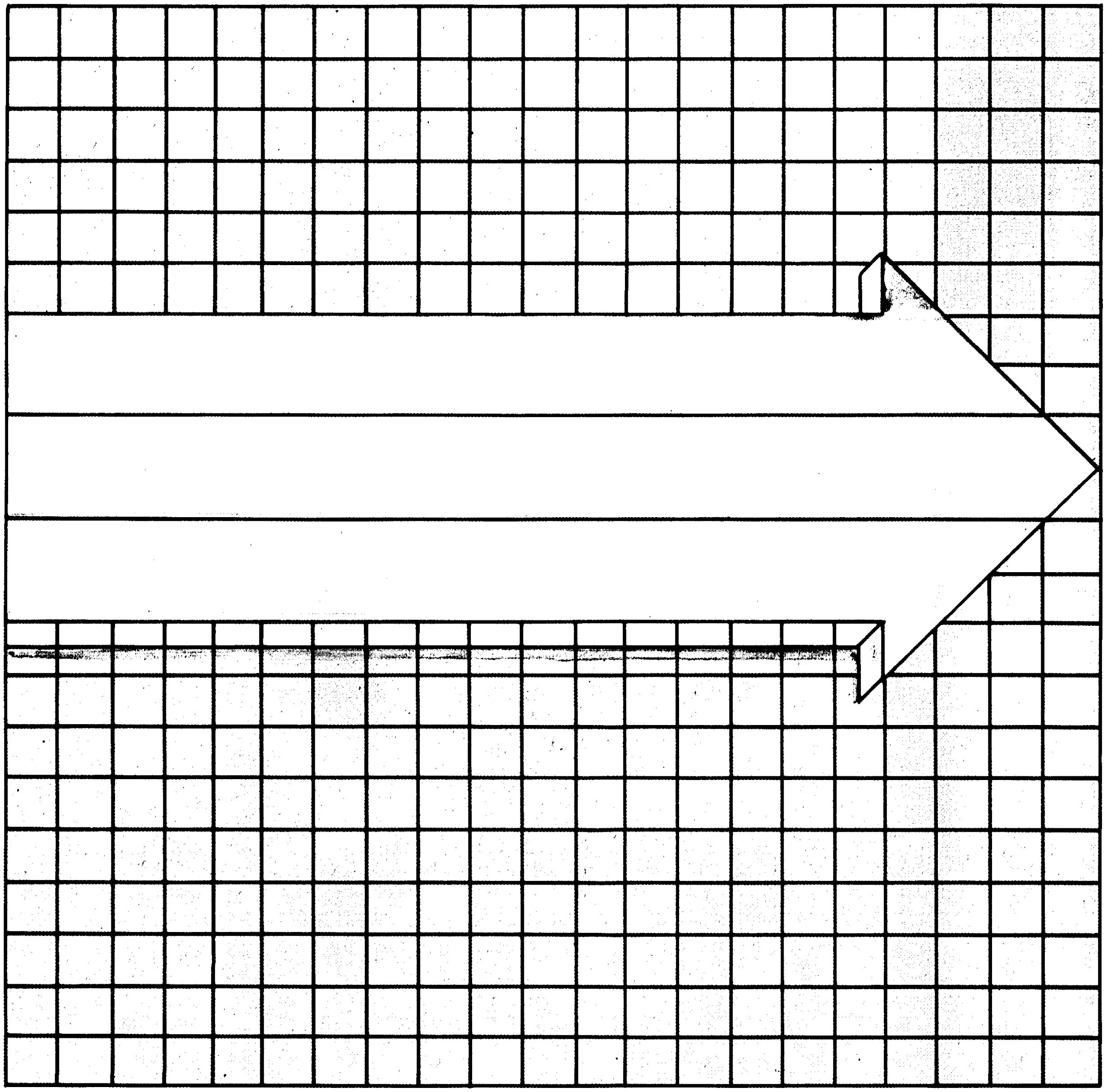
- To order from your IBM representative, use Order Number SBOF-0168.

- To order from your IBM dealer, use Part Number 92X1268.

A binder, and the *Managing the AIX Operating System* manual are included with the order. For information on ordering the binder and manual separately, contact your IBM representative or your IBM dealer.

Contents

Chapter 1. Introduction to System Management



CONTENTS

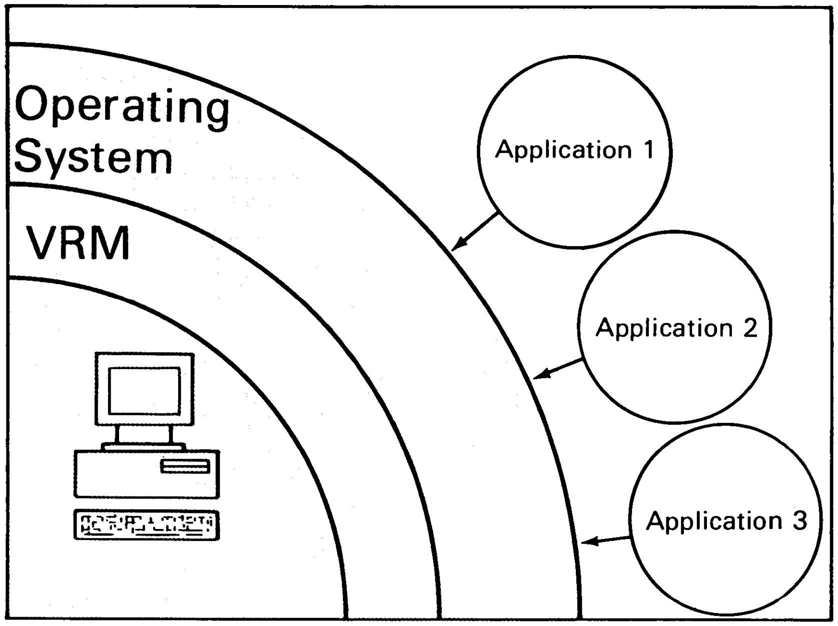
About this Chapter

The AIX Operating System operating system is a powerful and versatile tool. Generally, you use the operating system to do your work—for example, to process data, edit text files, run spreadsheet programs, and communicate with other users. There are times, however, when you must work on the operating system itself—for example, to add new users to the system or to create, back up, and repair file systems. The work you do on the operating system is called ***system management***.

This chapter includes the background information necessary for effective system management. While this chapter and the remainder of this book provide much specific information and careful guidance, you should understand that the job of system management is not the same on any two systems, and that system management requirements may change considerably on a particular system over time. Thus, in addition to becoming familiar with the information in the rest of this book, the best way to manage your system effectively is to continue to learn about the AIX system and the requirements placed on your system by its users.

General System Structure

You can think of the AIX system as having four levels. At the first level is the computer ***hardware***, especially the system unit. At the second level is the Virtual Resource Manager (***VRM***), a set of programs that manages the resources of the computer (main storage, disk storage, display stations, and printers). At the third level is the AIX Operating System kernel, a group of programs that send instructions to the VRM. At the fourth level are the ***application programs*** such as the shell, text editors, electronic spreadsheets, calendars, and communications programs. Together, the hardware, VRM, AIX Operating System kernel, and application programs comprise the RT PC system ***software***.



AUS105149

Figure 1-1. Relationships Between RT PC Hardware and Software

The Virtual Resource Manager (VRM)

The VRM is a group of programs that controls the system's physical components (the hardware) and makes the functions of the hardware available to an operating system (more than one operating system can run on the same VRM). An operating system interacts with the VRM as if the VRM were the hardware itself.

Through its management of hardware, the VRM provides ***virtual*** resources to the operating system. A virtual resource, such as virtual memory or a virtual terminal, is a simulation of a real resource. Virtual resources are based on the real resources of the system. The use of virtual resources makes the RT PC system more flexible than it would be if the operating system had to use real resources directly.

The Kernel

The kernel, using the VRM, controls the physical components of the RT PC system. Among the tasks done by the kernel are:

- Scheduling of processes (programs or commands) so that each process gets its share of processing unit time
- Handling data transfer (input and output) among system devices (for example, keyboards, displays, printers, and storage media)
- Managing the file system so that data can be stored and retrieved in an orderly way.

The shell and other programs use ***system calls*** to access the kernel. You do not need to understand system calls in order to manage your RT PC system. For those who do need to understand them (programmers, for example), the system calls are described in *AIX Operating System Technical Reference.*

The Shell

The shell, often called an ***interface*** or a ***command interpreter***, is the part of the operating system that makes it possible for you to use the kernel. The shell is actually an ordinary program that is said to run *on top of* the kernel.

You can use different interfaces, or different versions of the standard shell. Following is a list of the interfaces available on the AIX system:

- AIX shell
- Usability Services, described in *Usability Services Guide*
- DOS Services, described in *Using AIX Operating System DOS Services*
- C Shell, described under **csh** in *AIX Operating System Commands Reference.*

The information in this book is about the standard AIX shell.

The shell makes it possible for other programs to run by starting kernel processes. The kernel manages how its services are shared among programs.

The shell also makes it convenient for you to use certain basic kernel services, such as:

- Device-independent **input** and **output** (used by the shell to accomplish I/O redirection; for more information about redirecting the input and output, see *Using the AIX Operating System.*)
- **Pipes** (For information about pipes, see *Using the AIX Operating System.*)

Finally, the shell is a ***command programming language***. That is, with the shell, you can develop your own commands or shell procedures without having to use a conventional programming language. (For more information about the shell, see *Using the AIX Operating System* and **sh** in *AIX Operating System Commands Reference.*)

The File System–Background for System Management

Note: Before you begin this section, you should be familiar with how the file system appears to an ordinary system user. For information on file systems, see *Using the AIX Operating System.*

A file system is a complete directory structure, including a root directory and any subdirectories and files beneath it. File systems are confined either to a single ***minidisk*** (partition of a fixed disk) or to a diskette (one file system per diskette). Some of the most important system management jobs have to do with file systems, specifically:

- Allocating space for file systems on minidisks
- Creating file systems
- Making file system space available to system users
- Monitoring file system space usage
- Backing up file systems to guard against data loss in the event of system or disk failures
- Maintaining file systems in a consistent state.

There are certain system commands designed specifically for system management. Of those commands, the ones you probably will use regularly for working with file systems are:

backup	Performs a full or incremental backup of a file system.
clri	Clears i-nodes (used when file system inconsistencies cannot be corrected by **fsck**).
dd	Copies data directly from one device to another (for making file system back ups).
df	Reports the amount of space used and free on a file system.

fsck	Checks file systems and repairs inconsistencies.
mkfs	Makes a file system of a specified size on a specified minidisk.
mount	Attaches a file system to the system-wide naming structure so that files and directories in that file system can be accessed.
restore	Restores files from a backup.
umount	Removes a file system from the system-wide naming structure, making the files and directories in the file system inaccessible.

Regardless of the device they reside on (diskette or minidisk), all file systems have the same structure. Thus, you can move a file system from one device to another, provided the destination device is large enough. Disk space is allocated for file systems in ***blocks*** that can contain 512 ***bytes*** of data.

A file system has four major parts:

- Bootstrap block
- Superblock
- I-node blocks
- Data blocks.

Figure 1-2 on page 1-10 represents the major parts of a file system.

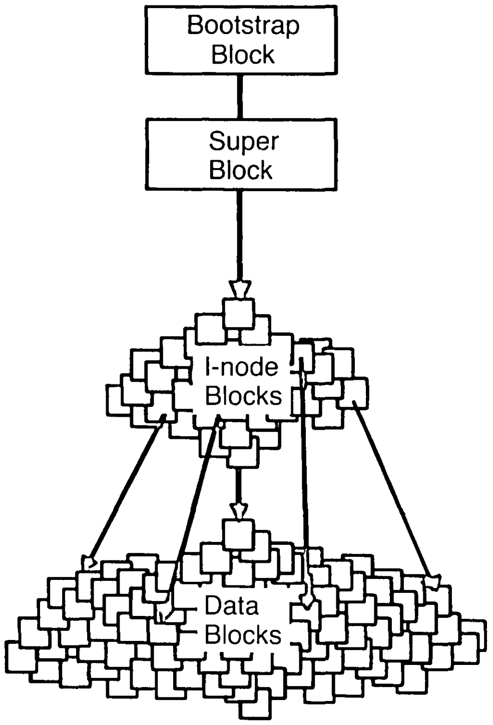


Figure 1-2. Major Parts of an AIX File System

Bootstrap Block

The first block of every file system (block 0) is reserved for a ***bootstrap***, or initialization, program. The bootstrap block is not actually part of the file system structure. File system data begin on block 1 of the minidisk.

The Superblock

Block 1 of every file system is called the ***superblock***. Among the most important information the superblock contains are:

- Total size of the file system (in blocks).
- Number of blocks reserved for i-nodes (explained below).
- Name of the file system.
- Volume ID of the minidisk or diskette.
- Date of the last superblock update.
- The head of the ***free-block list***, a chain that contains all of the free blocks in the file system (the blocks available for allocation). When new blocks are allocated to a file, they are allocated from this list. When a file is deleted, its blocks are returned to this list.
- A list of free i-nodes. This is a partial list of i-nodes available to be allocated for newly created files.

I-nodes

After the superblock there is a group of blocks that contain ***i-nodes*** (descriptions of the individual files in the file system). There is one i-node for each possible file in the file system. Each i-node is associated with an i-node number, or i-number. For any file system, there is a maximum number of i-nodes and thus, a maximum number of files that the file system can contain. This maximum number varies with the size of the file system. The first i-node (i-node 1) on every file system is unnamed and unused. When associated with files, the remaining i-nodes contain the following information:

- **File type**. Possible file types are ordinary file, directory, block device, character device, and first-in-first-out (also sometimes called *FIFO* or *named pipe*).

- **File owner**. The i-node contains the user and group IDs associated with the file.

- **Protection information**. This is a specification of read, write, and execute access for the owner, members of the group associated with the file, and others. It also includes other mode information specified by the **chmod** command. (For a discussion of protections, see *Using the AIX Operating System.*)

- **Link count**. A directory entry (link) consists of a name and the number of the i-node that represents the file (its ***i-number***). The link count specifies the number of directory entries that refer to the file. A file is deleted when the link count becomes zero. That is, its i-node is returned to the list of free i-nodes and its associated data blocks are returned to the free-block list. (For a discussion of links, see *Using the AIX Operating System.*)

- **Size of the file (in bytes)**.

- **Date the file was last accessed**.

- **Date the file was last modified**.

- **Date the i-node was last modified**.

- **Pointers to data blocks**. These pointers indicate the location of the data blocks on the physical disk.

I-node 2 must correspond to the root directory for the file system. All other files in the file system are below the root directory in the file system. Beyond i-node 2, any i-node can be assigned to any file. Similarly, any data block can be assigned to any file. Neither i-nodes nor blocks have to be allocated in any particular order.

Data Blocks

Beyond the i-nodes, the file system consists of ***data blocks***. Each data block can contain 512 bytes of information. The i-node points directly to the first 10 data blocks of the file.

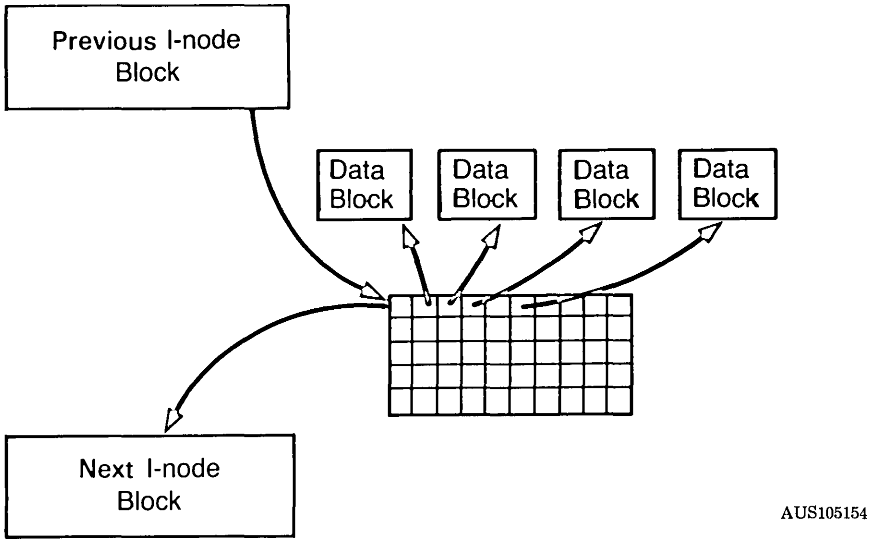
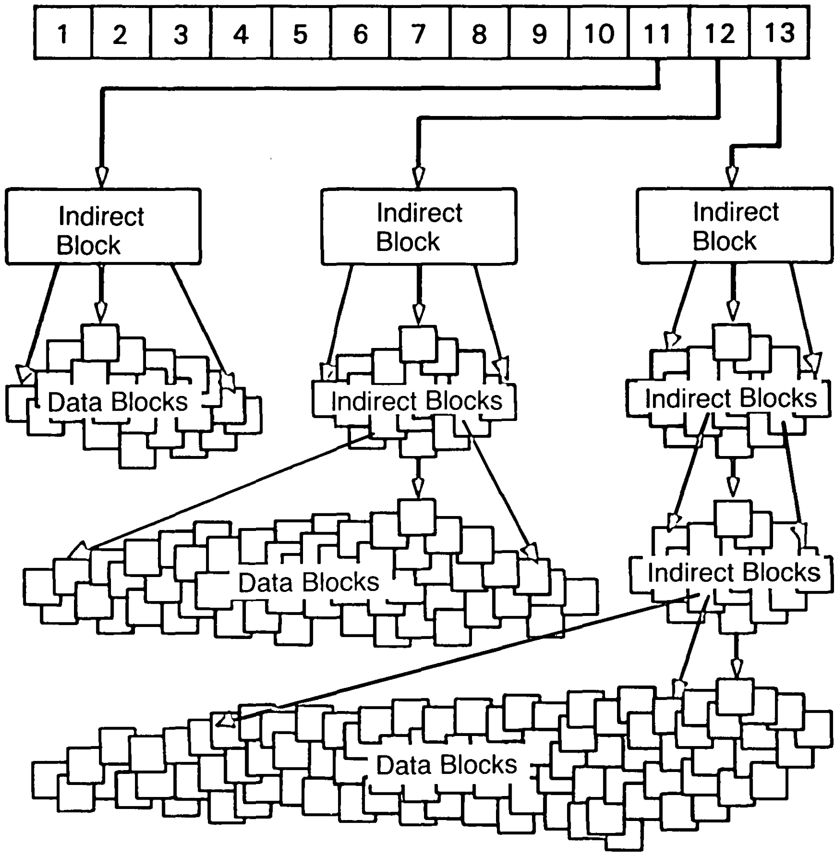


Figure 1-3. Direct Pointers from an I-node to Data Blocks

When a file contains more than 10 blocks of data, block number 11 is an ***indirect block***. The indirect block contains ***pointers*** to 128 additional data blocks. A file that has one indirect block is called ***single-indirect***, and has a maximum size of 139 blocks (the first 10 data blocks, plus the indirect block, plus the 128 data blocks.

If a file is larger than 138 blocks, block number 12 is a ***double indirect*** block. A double-indirect block points to an indirect block that contains 128 pointers to other indirect blocks. Each of those indirect blocks in turn points to 128 data blocks. Thus, a double-indirect file has a maximum size of 138 + (128 * 128), or 16,522 blocks.

The largest files possible are ***triple-indirect***. Block number 13 of a triple-indirect file points to the first of three levels of indirect blocks, allowing a maximum file size of 16,522 + (128 * 128 *128), or 2,113,674 blocks. Figure 1-4 shows the relationship of data blocks and indirect blocks associated with an i-node.



AUS105155

Figure 1-4. The Relationship of Data Blocks and Indirect Blocks

The Base AIX File System

The AIX operating system is made up of four separate file systems:

/ (called ***root***)

/usr

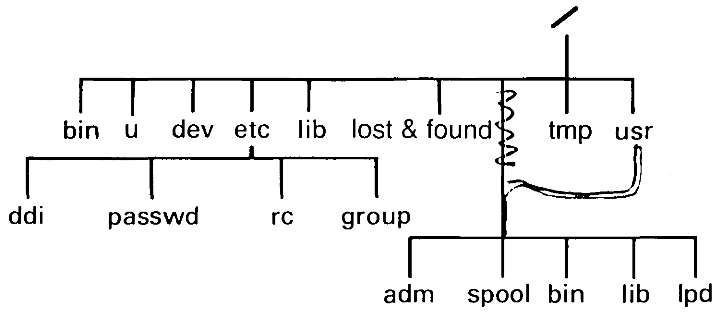
/tmp

/u

Each of these file systems resides on a separate minidisk. The system automatically mounts all four file systems when it initializes. Under normal conditions, it does not matter that there are four file systems—they mount automatically and function like a single file system. However, multiple file systems are convenient for certain system management tasks (for example, backing up, restoring, and repairing file systems) because they allow you to isolate a part of the system while you work on it.

Major Files and Their Functions

The AIX Operating System contains hundreds of files, even before anyone creates work files on the system. (If you are curious about just how many files the operating system contains, enter the command `ls -Ra` to display the path name of every file in the system.) You do not need to work directly with most of these files, or even to know where they are located in the file system. There are a few files, however, that are important in routine system management. You should be familiar with what these files contain and where they are located in the root file system. Figure 1-5 on page 1-16 shows these files and their relationships.



AUS105156

Figure 1-5. The Base AIX File System

Following are brief descriptions of the major AIX system files:

/
: The highest directory in the file system hierarchy, usually called the ***root*** directory.

/dev
: A directory containing the special files that allow input and output operations to system devices (such as fixed disks, diskette drives, and terminals). (For more information on special files and system devices, see "The Input/Output System" on page 3-17.)

/etc
: A directory containing most of the commands required for system startup, and maintenance.

/etc/ddi
: A directory containing files that describe system devices (for example, printers). **ddi** stands for **device dependent information.**

/etc/master
A file that contains information about system devices used by the **config** program to create configuration files.

/etc/passwd
A file containing the information that defines users for the system (including encrypted passwords).

/etc/group
A file containing the information that defines groups of users for the system.

/etc/rc
A file containing the system startup routine (*run commands*). The usual way to change how the system starts up (for example, to specify what programs run automatically) is to modify this file.

/u
A directory containing the home directories of all system users.

/usr
A directory containing other directories of commands.

/usr/adm
A directory containing the commands used in system accounting operations.

/usr/spool
A directory containing the ***spool***ed files to be sent between users, between the system and devices, and between systems.

/usr/lpd
A directory containing commands used to send data to system printers and to collect accounting statistics on printer usage.

/usr/bin
A directory containing commands typically available to all system users and not contained in **/bin**.

/usr/lib
: A directory containing system libraries and system management, text processing, and other commands not found in **/usr/bin** or **/bin**.

/bin
: A directory containing the basic commands required to run the system. Additional commands are contained in **/usr/bin** and **/usr/lib**.

/lib
: A directory containing files that make up the C language programming library and parts of the C compiler.

/tmp
: A directory containing temporary files created by processes.

Finding and Viewing System Files

As you learn about the AIX system, you probably will find it very helpful to be able to locate directories and files and look at their contents. The following three commands make it easy for you to do so:

find

Use the **find** command to search the file system for a file (or directory) when you know its name. The command:

```
find / -name filename -print
```

searches the file system, from the root directory (/) down, for *filename*. If it finds a file with that name, it displays the complete path name for that file. If *filename* is not in the file system, the **find** command displays nothing when it completes.

ls

Use the **ls** (list directory) command to list the contents of a directory. The command

```
ls dirname
```

lists the names of the entries in the directory *dirname*. The name of the directory can be a path name. You can use **ls** command flags to get different types of information about the items in a directory. For information about the different ways to use **ls**, see *Using the AIX Operating System* and **ls** in *AIX Operating System Commands Reference.*

(like 'more') **pg**

Use the **pg** (page) command to display the contents of a file one page at a time. The command

```
pg filename
```

displays the contents of *filename* one page at a time. At the bottom of each page on the screen, the **pg** command displays : (colon). To display the next screen of information, press **Enter**. For more information on the **pg** command, see *Using the AIX Operating System* or **pg** in *AIX Operating System Commands Reference.* (You also can use the **cat** [concatenate] command to display the contents of a file; however, the **cat** command scrolls through the entire file at once, rather than one page at a time.)

For more information about these commands and the different ways in which you can use them, see the appropriate entries in *AIX Operating System Commands Reference.*

Chapter 2. Routine System Management

CONTENTS

About This Chapter

This chapter explains the routine tasks associated with managing an AIX system. You may do some of the tasks described in this chapter daily, for example, starting up and shutting down the system, mounting and unmounting file systems, and backing up file systems. Other tasks you probably will do less often—setting up user accounts and creating new file systems. However, understanding the information in this chapter will help you use the AIX system effectively and keep it running smoothly.

Starting the System

To start the AIX system under normal circumstances, turn on the power switch for each system component. If there is a diskette in drive 0 (A), the system will attempt to initialize from that diskette. If there is no diskette in the drive, the system attempts to initialize from the fixed-disk. Under normal circumstances, the system should initialize from the fixed-disk. When the system successfully initializes from the fixed-disk, it displays the copyright notice and the `login` prompt. At that time, the system is ready for normal use. For an explanation of how the initialization procedure works, see "System Initialization."

For certain system management tasks, you occasionally need to run a special version of your system: the ***maintenance system***. For an explanation of the maintenance system, see "Running the Maintenance System" on page 2-6.

System Initialization

After loading the kernel into memory, the system goes through a period of internal initialization, during which it runs internal checks, initializes all memory, initializes some devices, and analyzes the root file system. After completing its internal initialization, the kernel starts the **init** (system initialization) process. All other processes on the system (except the scheduler) can be traced back to **init**. Most important of the processes that **init** creates are the shell and the processes that allow users to gain access to the system; that is, **init** runs the **getty** command for each ***port*** (display station) defined on the system. (For more information about the initialization processes, see **init** and **getty** in *AIX Operating System Commands Reference.*)

If the kernel is loaded from the maintenance diskette, **init** initializes the maintenance system. The maintenance system consists of only a shell with superuser authority, the **init** process, and the scheduler. (For more information about the maintenance system, see "Running the Maintenance System" on page 2-6.) To go from the maintenance system to normal operation, run **shutdown** and then press **SOFT IPL**.

When the system is initialized for normal operation, **init** creates a shell process to run the commands in the **/etc/rc** file. The commands in **/etc/rc** prepare the system for users to log in. If the previous shutdown was not orderly, **init** runs the **fsck** command to check the consistency of the root file system and all other file systems normally checked (as specified in **/etc/filesystems**).

To alter the way the system starts up, edit the **/etc/rc** file. Generally, **/etc/rc** contains three types of commands:

Housekeeping

One of the first commands in **/etc/rc** is **vrmconfig**, which configures the VRM for use with the AIX Operating System. Several commands in **/etc/rc** make certain that the system is in proper running condition, regardless of its condition when it was last stopped. The **fsck** command, which should be the first command in **/etc/rc**, checks the consistency of file systems and, as far as possible, corrects file system problems. If **fsck** cannot automatically repair the file system damage it detects, it causes the shell to stop the system. You cannot start the system until you repair the file system damage by running the maintenance system as described under "Running the Maintenance System" on page 2-6. Other housekeeping commands remove files from temporary directories, initialize queues, save logged data, restore certain files to their original (default) states, and set up the ***ports*** that give display stations access to the system.

Mounting

The **rm -f /etc/mnttab** command in **/etc/rc** restores the mount table to its default state. Then the **/etc/mount all** command mounts all of the file systems described in **/etc/filesystems**.

Starting daemons

Daemons are processes that should start when the system starts and run until the system stops. The two most important daemons started by **/etc/rc** are **cron** and **qdaemon**.

The **cron** daemon runs the **sync** system call at 30-second intervals, completing any unfinished disk I/O. Thus, if the system fails, data on the disks cannot be more than one minute out of date. You can also use the **cron** daemon to start processes (specified in **/usr/lib/cron/crontab**) at preset times. For example, **cron** can start the processing of system accounting data late at night.

The **qdaemon** provides queued access to certain system resources, such as printers. For more information on queues, see "Using the Queueing System" on page 3-20.

Once it successfully completes the initialization process, **init** starts the necessary ***getty*** processes to allow display stations to use ports. After creating the initial loggers, the only function of **init** is to create loggers or set up new ports as required.

Running the Maintenance System

The AIX maintenance system provides you with a way to perform certain system management tasks without starting the system in the normal manner. You can use the maintenance system for one of two reasons:

- The system will not initialize. The maintenance system may allow you to correct the problem that prevents your system from operating normally.
- You need to perform system management tasks that would be complicated by other processes running on the system. The maintenance system is a very limited form of the operating system, consisting of just the parts of the system required to perform certain system management tasks.

The maintenance system is most useful for backing up and repairing file systems or for making any change to the system that might be complicated if other processes are running. For example, during normal startup, the operating system tries to correct any file system damage. However, some types of damage cannot be corrected automatically. In such cases, the operating system displays a message indicating that the operator must help correct the damage. To make the repairs, start the maintenance system and do one of the following:

- Use the **check a file system** maintenance command (a version of the **fsck** command described under "Checking and Repairing File Systems—The fsck Command" on page 3-8).
- Start the **standalone shell** and run **fsck** on the damaged file system.

Note: If you catch file system damage immediately, it is usually simple to repair. However, if the system runs on a damaged file system, the damage can spread, in some cases to the point that the entire file system is unusable.

You should run **fsck** only on unmounted file systems, and you should not need ***daemon*** processes while you use the maintenance system (a ***daemon*** process is a process that runs continually in the background, such as the process that manages printer queues). Therefore, when you start the maintenance system, none of the usual file systems are mounted and none of the usual daemon processes are started.

The remainder of this section explains how to start and use the maintenance system with the AIX Operating System Installation/Maintenance diskette.

To Start the Maintenance System

- If the system is not powered on:

  1. Insert the AIX Operating System Installation/Maintenance diskette into diskette drive 0 (or A).
  2. Turn on the power to all components as usual.

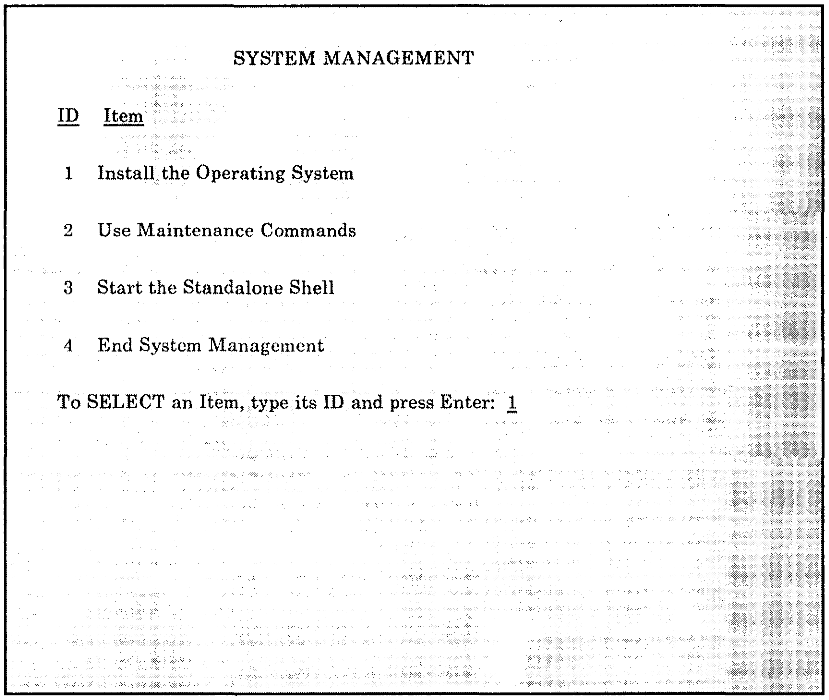
- If the system is powered on:

  1. Be certain that all other users log out of the system.
  2. Enter:

     ```
     shutdown
     ```

  3. When **shutdown** completes, insert the AIX Operating System Installation/Maintenance diskette into diskette drive 0 (or A).
  4. Press **SOFT IPL**.

Once initialized, the maintenance system displays the **SYSTEM MANAGEMENT** menu:

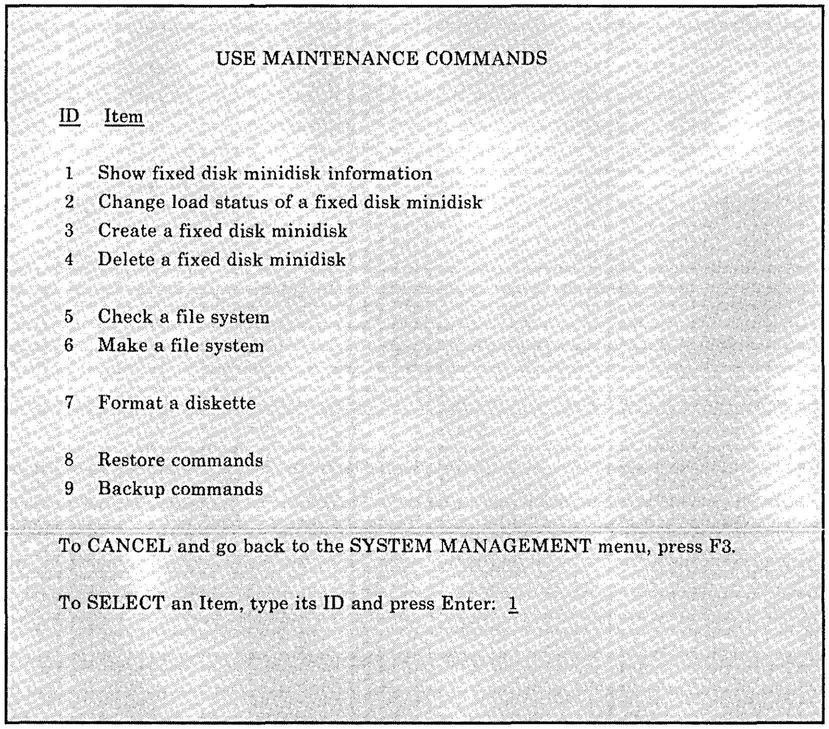


```
                    SYSTEM MANAGEMENT

ID  Item

 1  Install the Operating System

 2  Use Maintenance Commands

 3  Start the Standalone Shell

 4  End System Management

To SELECT an Item, type its ID and press Enter: 1
```

Select item 2 to use the maintenance commands explained under "Using Maintenance Commands" on page 2-10. Select item 3 to use the ***standalone shell*** explained under "Using the Standalone Shell" on page 2-11. To stop the maintenance system, select item 4. Then, to initialize the system for normal operation, press **SOFT IPL.**

Using Maintenance Commands

When you choose item 2 (**Use Maintenance Commands**) on the SYSTEM MANAGEMENT menu, the maintenance system displays the **USE MAINTENANCE COMMANDS** menu:



```
                    USE MAINTENANCE COMMANDS

ID  Item

 1  Show fixed disk minidisk information
 2  Change load status of a fixed disk minidisk
 3  Create a fixed disk minidisk
 4  Delete a fixed disk minidisk

 5  Check a file system
 6  Make a file system

 7  Format a diskette

 8  Restore commands
 9  Backup commands

To CANCEL and go back to the SYSTEM MANAGEMENT menu, press F3.

To SELECT an Item, type its ID and press Enter: 1
```

To select an item, enter its number.

For more information about the function of each maintenance command, see the following:

Item ID	**Reference**
1-4	Information on installing the AIX Operating System in *Installing and Customizing the AIX Operating System.*

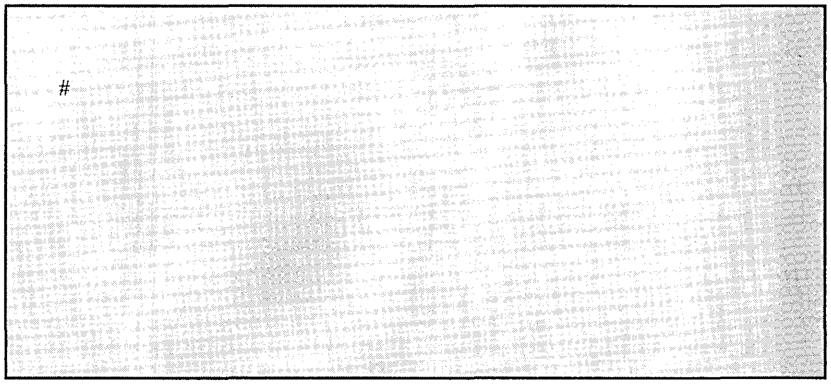
5	"Checking and Repairing File Systems—The fsck Command" on page 3-8.
6	"Creating and Mounting File Systems" on page 2-50.
7	"Formatting Diskettes" on page 2-55 and **format** in *AIX Operating System Commands Reference.*
8-9	"Backing up Files and File Systems" on page 2-58.

Note: When you choose items 8 and 9 on the **USE MAINTENANCE COMMANDS** menu to backup a file system, the backups and restores are by i-node only.

Each item leads you through a series of menus that allow you to accomplish a particular task.

Using the Standalone Shell

When you choose item 3 (**Start the Standalone Shell**) on the SYSTEM MANAGEMENT menu, the maintenance system displays the following screen:



```
#
```

Notice that the standalone shell prompt is # rather than the $ of the standard shell.

The standalone shell is most useful for tasks that you cannot perform while the operating system is running, such as repairing a file system.

When you start the standalone shell, the **init** command runs automatically, much as it does when you start the system for normal operation. Once the standalone shell is started, the following commands are available:

backup	(make backup copies of files and file systems)
chparm	(change system parameters)
clri	(clear i-nodes)
cp	(copy files)
dd	(convert and copy files)
ed	(edit files)
format	(format diskettes)
fsck	(check and repair file systems)
fsdb	(debug file systems)
init	(initialize system)
ln	(link files)
ls	(list directory contents)
maint	(maintenance program)
mkdir	(make directories)
mkfs	(make file systems)
mknod	(create special files)

mount	(mount file systems)
mv	(move files)
rm	(remove files)
rmdir	(remove directories)
sh	(run a shell)
stty	(set display station characteristics)
sync	(update storage from buffers)
restore	(restore backed up files and file systems)
tctl	(send subcommands to the streaming tape device)
umount	(unmount file systems)

Most of these commands are discussed in other sections of this book (see the index) or in *AIX Operating System Commands Reference.*

File systems used during normal mode operation may be mounted from the standalone shell. The **/mnt** directory may be used for this. For example, the **root** directory of the normal system is created when the AIX Operating System is installed. The **/dev/hd0** minidisk contains this file system. To access files on the normal **root** file system from the standalone shell, use the command:

```
mount /dev/hd0 /mnt
```

Likewise, **/usr** is found on the **/dev/hd2** minidisk, **/tmp** is found on the **/dev/hd3** minidisk, and **/u** is found on the **/dev/hd1** minidisk. Thus, you could also use the command:

```
mount /dev/hd2 /mnt/usr
```

The commands of the standalone shell may be extended by mounting **/dev/hd0** and **/dev/hd2** and running commands that exist on these file systems.

1. To examine data or to run normal mode commands, enter:

   ```
   mount /dev/hd0 /mnt
   mount /dev/hd2 /mnt/usr
   mount /dev/hd1 /mnt/u
   mount /dev/hd3 /mnt/tmp
   PATH=/mnt/bin:/mnt/usr/bin:$PATH
   cd /mnt
   ```

2. To check the **root** file system, enter:

   ```
   fsck /dev/hd0
   ```

3. To check the **/usr** file system, enter:

   ```
   fsck /dev/hd2
   ```

4. To make backups by name, enter:

   ```
   mount /dev/hd0 /mnt
   mount /dev/hd2 /mnt/usr
   mount /dev/hd1 /mnt/u
   mount /dev/hd3 /mnt/tmp
   backup -i
   cd /mnt
   ```

   (Enter the file names relative to the current working directory, that is, **etc/system**, **bin/cp**.)

To end the standalone shell, press **END OF FILE**.

Stopping the System

The AIX system can run multiple processes, including background processes, at the same time. To prevent damage to the file system, you should bring all processes to an orderly stop before you turn off the power to the computer. The **shutdown** command (described from a user's perspective under "Starting the System" on page 2-4) is the usual way to bring the system to an orderly stop.

During the default shutdown, all users are notified through the **wall** command of the impending system shutdown. The **hold** command prevents any new logins. After the specified number of *seconds* (60 by default), the system stops the accounting and error-logging processes. **shutdown** then runs the **killall** command to end any remaining processes and runs the **sync** command to flush all memory resident disk blocks. Finally, it unmounts the file systems and sends the appropriate signal to **init** and prints the message: `....Shutdown completed.....` At this point, you can safely turn off the power switch on the system unit.

Note: If you have a single user system, you may wish to do a `shutdown -f` to eliminate the message from the **wall** command and the delay.

For information related to stopping the system, see "Disk Buffering—the sync Command" on page 3-5.

Managing User Accounts

Giving users access to the system is an important system management task. Even if you are the only user on the system, you may need to create accounts, remove accounts, and change account information occasionally. You can perform these account management tasks with the **users** command described under "Creating, Changing, and Removing Accounts—The users Command." To manage the accounts on your system most effectively, you also should be familiar with the information under "Types of User Accounts" on page 2-29, "Using Different Log In Names" on page 2-33, and "User Account Files" on page 2-34.

Note: Another name for **users** is **adduser**.

Creating, Changing, and Removing Accounts—The users Command

The **users** command is a tool for managing user accounts. **users** provides the following subcommands:

add	Creates new user accounts and new groups.
change	Changes account and group information, such as user name, password, and group ID.
delete	Removes accounts or groups from the system.
help	Displays the **users** subcommands.
invalidate	Makes a user password invalid.
quit	Puts changes into effect and ends **users**.
show	Displays user or group account information.

To run the **users** command, you must be a member of the system group or have superuser authority.

Starting and Stopping users

To Start users

- Enter users.

After you start **users**, you can use any of its subcommands to perform your account management tasks. Enter ? (question mark) to display the list of subcommands:

```
Available commands are:

a[dd]          user or group
c[hange]       user or group
d[elete]       user or group
h[elp]         print this message
i[nvalidate]   a user
q[uit]         finalize changes and exit
s[how]         user or group

<DEL> will exit without changing any of the files.
>
```

Note: The DEL key referred to in this message corresponds to **INTERRUPT**.

To select a subcommand, type its initial letter (for example, a for **add**) and press **Enter**. If the subcommand requires you to specify user or group, use the abbreviation u or g (for example, a u tom).

To show the information about a user, enter: show user *username* (or s u *username*). To show the information about a group, enter : show group *groupname* (or s g *groupname*).

To display help information, enter: h.

To Stop users

- To stop **users** without making any changes, press **INTERRUPT**.

OR

- To put changes you have made into effect and stop users, enter `q`.

The following sections explain how to use the remaining **users** subcommands (**add**, **change**, **delete**, and **invalidate**).

Using the a[dd] Subcommand

To Add a User

1. Enter: `users` (If **users** is already started, go to step 2.)

2. After the > prompt, enter: `a u` *username.* (Use lowercase letters for the names of users you add to the system.)

3. To add user information, enter: `n` after `OK? (y)`.

4. To add the user's full name:

   a. Enter: `fu` after the `Field?` prompt.
   b. Enter the user's full name (for example, `john doe`,

5. To add a password:

   a. Enter: `pa` after the `Field?` prompt.
   b. Enter the password.

6. To add other information:

   a. After the `Field?` prompt, enter enough of the field name to identify it.
   b. Enter the information.

7. When your changes are complete, press **Enter** after the `Field?` prompt. **users** displays the new information (passwords are encrypted).

8. If the changes are correct, press **Enter** after the `OK? (y)` prompt to add the new user to the system. (If the changes are not correct, enter `n` and then make the necessary corrections).

9. Press **Enter** after the prompt `Standard new user initialization? (y)`.

10. Enter another subcommand or stop **users**.

Note: You are not allowed to change the information in the **opt. groups** field from the **add user** subcommand. To add a user to a group, use the **change group** subcommand.

You can set the values for other items with the same procedure used to set `Fullname` and `Password`. For more information about user accounts, see "Types of User Accounts" on page 2-29.

To Add a Group

1. Enter: **users**. (If **users** is already started, go to step 2.)
2. After the > prompt, enter `a g` *groupname*.
3. To accept the displayed information, press **Enter** after the `OK? (y)` prompt. (To add group information, enter `n` after the `OK? (y)` prompt.)
4. To add a group password:

   a. Enter `pa` after the `Field?` prompt.
   b. Enter the group password.
5. To add members to a group:

   a. Enter `m` after the `Field?` prompt.
   b. Enter `a`.
   c. Enter the `username` of the user you are adding.

   Note: Be sure to type the name correctly because there is no check to verify if this name is defined.
6. To end this procedure, press **Enter** after the `Field?` prompt.
7. If the changes are correct, press **Enter** after the `OK? (y)` prompt to add the group to the system.
8. Enter another subcommand or stop **users**.

For more information about groups and the fields used by the **add group** command, see “The Group File” on page 2-38.

Using the c[hange] Subcommand

To Change User Information

1. Enter: `users` (If **users** is already started, go to step 2.)
2. After the > prompt, enter `c u` *username*
3. To make changes to the displayed information, enter `n` after the `OK? (y)` prompt.
4. Enter the name of the field you want to change after the `Field?` prompt.
5. Enter the information for the field you specified.
6. Use the same method to change the information in other fields.
7. When your changes are complete, press **Enter** after the `Field?` prompt.
8. If the changes are correct, press **Enter** after the `OK? (y)` prompt. **users** displays the `[UPDATED]` message.
9. Enter another subcommand or stop **users**.

For more information about the fields for which you can change values, see "The /etc/passwd File" on page 2-35.

Note: You cannot change the information in the **opt. groups** field from the **c u** subcommand. To add a user to a group, use the **change group** subcommand.

To Change Group Information

1. Enter: `users`. (If **users** is already started, go to step 2.)
2. After the `>` prompt, enter `c g` *groupname.*
3. To make changes to the displayed information, enter `n` after the `OK? (y)` prompt.
4. Enter the name of the field you want to change after the `Field?` prompt. (For example, to change the members shown as part of the group, enter `m` or `member`.
5. Enter `a` or `d`
6. Enter the user name you wish to add or delete. Be sure to type the name correctly because there is no check to verify if this name is defined.
7. When your changes are complete, press **Enter** after the `Field?` prompt.
8. If the changes are correct, press **Enter** after the `OK? (y)` prompt. **users** displays the `[UPDATED]` message:
9. Enter another subcommand or stop **users**.

Using the d[elete] Subcommand

To Delete a User

1. Determine if you want to save the files in the user's directory. If you do, change the name of the directory, change the owner of the directory, or follow the instructions in step 3 for saving files. If you do not want to save the files, remove all files (including the hidden files such as the **.profile** file) from the directory of the user that is to be deleted.

2. Enter: `users`. (If **users** is already started, go to the next step.)

3. After the > prompt, enter `d u` *username*. **users** displays the prompt `Should the login directory (/u/`*username*`) be removed? (y)`. If you want to save the files in that directory, type `n` and press the Enter key.

   If you do not want to save the files, press the Enter key, and you should see the `[DELETED]` message.
   (If you have not removed all files from the user's login directory, **users** displays the message, `The user cannot be deleted because the user's login directory is not empty.`

4. To verify that the user is deleted, enter `s u` *username*. **users** should display a message indicating that there is no such user.

5. Enter another subcommand or stop **users**.

Notes:

1. Be sure to change the ownership of (or delete) all files owned by the user who is being deleted, including files which may reside in directories other than the ***/u/username*** directory. Failure to do so may cause problems later. The owner of a file is identified by the user number (UID). Later, when a new user is added to the system, the same number may be used by the

system again. The new user can then own any of the files which were not removed or for which ownership was not changed. For information about changing the ownership of a file, see the **chown** command in *AIX Operating System Commands Reference.*

2. When you delete a user, you have the choice of removing all of the files in that user's login directory or answering `no` to the prompt, `Should the user's login directory be removed?` If you answer `no` to the prompt, the entry for `username` is removed from the **/etc/passwd** file, but the **/u/username** directory will not be removed. You would not have to copy the files you want to save to another directory, but you would need to change the ownership of the files in the **/u/username** directory and of any other files owned by that user.

To Delete a Group

1. Enter: `users`. (If **users** is already started, go to the next step.)

2. After the > prompt, enter `s g` *groupname*. Write down the names of all the users listed under `Members`.

3. After the > prompt, enter `s u` *username* for one of the members of the group. Check to see if the group you want to delete is listed under `Group`. If it is, use the **c u *username*** subcommand to change the **Group** field. Repeat this step for each user you listed in the previous step.

   Note: See the example below for an alternate method for determining which users use the group you want to delete as their primary group. If there is a large number of members in the group, you may prefer the alternate method.

4. After the > prompt, enter `d g` *groupname*. **users** displays the `[DELETED]` message.

5. To verify that the group is deleted, enter `s g` *groupname*. **users** should display a message indicating that there is no such group.

6. Enter another subcommand or stop **users**.

Note: Before deleting a group, change the group field for all users using that group as their primary group. The next group added is assigned that group id. Users may get incorrect group assignments.

You can use an alternate method to delete a group. In the following example, the group is named **group1**. Use information in the **/etc/group** and **/etc/passwd** files to determine which users have **group1** as their primary group.

Enter `pg /etc/group` to display the group file. For more information about the **/etc/group** files see "The Group File" on page 2-38. The following example shows **group1** with a group number (GID) of 200:

```
system::0:root,su
staff::1:user2,user3,user4,user5
bin::2:root,su,bin
sys::3:root,su,bin,sys
adm::4:root,su,bin,adm
mail::6:root,su
usr::100:guest
group1::200:user3,user4,user1
```

Note that the group number for group1 is 200. You can also get this information by using the **s g group1** subcommand after starting **users**. The group number is shown in the GID field.

Enter `pg /etc/passwd` to display the password file. For more information about the **/etc/passwd** file, see "The /etc/passwd File" on page 2-35. The following example shows **group1** with a group number (GID) of **200** for the primary group:

```
root::0:0::/:
su:*:0:0::/:
daemon:*:1:1::/etc:
netmail:*:1:1::/usr/spool/qftp:/usr/lib/INnet/waxsrvr
bin:*:2:2::/bin:
sys:*:3:3::/usr/sys:
adm:*:4:4::/usr/adm:
uucp:*:5:1::/usr/spool/uucppublic:/usr/lib/uucp/uucico
adduser:*:0:0::/usr/adm:/etc/adduser
guest:*:100:100::/usr/guest:
name::201:0::/u/name:
user1:utGzXG(TBLEso:200:200::/u/user1:/bin/sh
user2:6uxKlvlyz4eGc:202:1::/u/user2:/bin/sh
user3:Iu44UqcUnatjg:202:1::/u/user2:/bin/sh
user4:Yu101nqNFPurY:202:1::/u/user2:/bin/sh
user5:kumNeVh8GqHT2:202:1::/u/user2:/bin/sh
```

Notice in the above example, that **user1** is the only entry with **200** as the group number field. It is the only entry in the above example with **group1** as the primary group. Using the information from the **/etc/group** and the **/etc/passwd** files, you can change the primary group for **user1** by using the **users** command as described below:

1. Enter: `users`.
2. After the > prompt, enter `c u user1`.
3. To change the displayed information, enter `n` after the `OK? (y)` prompt.
4. To change the primary group:

   a. Enter: `gr` after the `Field` prompt.
   b. Enter the name for the new primary group.

5. Press Enter after the `Field?` prompt.
6. If the changes are correct, press Enter after the `OK?` prompt. **users** displays the `[UPDATED]` message:
7. After the > prompt, enter `d g group1` to delete the group. **users** displays the `[DELETED]` message:
8. To verify that the group is deleted, enter `s g group1`. **users** should display a message indicating that there is no such group.
9. Enter `q` to stop **users**.

Note: When deleting a group, check and see if any users have it as their primary group. If they do, change the primary group for those users. Failure to do so may cause problems later. The primary group for a user is identified by the group number (GID). Later, when a new group is added to the system, it may be assigned the same GID that was used by the deleted group. Users whose primary group was not changed would then have the new group as their primary group.

Files also need to be checked. It may not be practical to change the group ownership of all files created by a deleted group. However, there may be particular files which you may want to reassign. Use the **chgrp** command to change the group ownership of a file. For more information about the **chgrp** command, see *AIX Operating System Commands Reference.*

Because of the possible problems involved in using the **delete group** subcommand, use it sparingly. Give careful consideration to the consequences before deleting a group.

Invalidating and Reinstating Users

You can prevent a user from logging in to the system by invalidating the user. You do not have to delete a user's files to invalidate the user.

To Invalidate a User

1. Enter: `users`. (If **users** is already started, go to step 2.)
2. After the > prompt, enter `i` *username.* **users** displays the `Invalidating` *username* message.

   Note: Do not specify `u` with the **i** subcommand.
3. Enter another subcommand or stop **users**.

To reinstate the user, use the **change** subcommand to clear the user's program field.

Types of User Accounts

Each entry in the **/etc/passwd** file identifies an account. ("The /etc/passwd File" on page 2-35 describes the **/etc/passwd** file.)

There are two types of accounts on the AIX system:

- The account with superuser authority
- User accounts.

If you know the user name and password associated with an account, you can log in as that user. Thus, you can have one or more user accounts for your regular work and also, when necessary, use the account with superuser authority for system management work.

root–The Account with Superuser Authority

The user name ***root*** identifies the one user who operates without any restrictions–the user who has ***superuser authority***. (Another name for root is ***su***, for superuser.)

When you have superuser authority, you are immune from all permission checks in the system, you have read, write, and execute permission for every file in the system, you can issue any system call, and you can cancel any process.

Because none of the usual protections apply when you have superuser authority, you should be extremely careful when you are logged in as root. Make it a practice never to log in as root when you can log in as a regular user and accomplish the same task. On a system that serves several different people, it is good practice to strictly limit who knows the root password, and to change the password often, in order to reduce the chance of accidental damage.

You can obtain superuser authority in three different ways:

- After you log in with your normal user name, enter the **su** (switch user) command and supply the root password when prompted to do so. The shell prompt changes (usually from `$` to `#`) to remind you that you have superuser authority. To return to normal user status, press **END OF FILE**. This is usually the most convenient way to obtain and give up superuser authority.

- Log in with the one of the user names `su` or `root`. Instead of the usual `$` (shell) prompt, the system displays the root prompt (usually `#`), to remind you that you have superuser authority. After you finish the work that requires superuser authority, either log out (press **END OF FILE** and the `$` prompt returns) or run the **shutdown** command to stop the system.

- Start the maintenance system (see "Running the Maintenance System" on page 2-6). The maintenance system shell has superuser authority.

User Accounts

All user accounts are subject to normal system checks and protections. To reduce the chance of accidental damage to the system, always work with a user account except when you must have superuser authority.

There are two types of user accounts: those created with the **users** program (described under "Creating, Changing, and Removing Accounts—The users Command" on page 2-16) and those supplied with the system to perform certain system management tasks. Accounts created with **users** might be called ***ordinary user accounts***. These accounts are commonly identified with a particular person. They are used to run application programs (for example, the shell, text editors, or electronic spreadsheets) and store the data associated with them. As with any account, there is an entry in the **/etc/passwd** file for each ordinary user account. The home (or log in) directories for ordinary user accounts are located in the **/u** directory (for example, `/u/jones`).

The second group of user accounts is supplied with the system. They give users access to certain data and the ability to perform certain system management tasks which otherwise would require superuser authority.

Following is a list of these special user accounts:

bin The user **bin** owns most of the directories, libraries, and programs or commands provided with the system. While most users have **x** (execute) permission for the files owned by **bin**, only **bin** can modify those files. Normally, you should log in as **bin** to install new programs and libraries.

In most respects, **bin** has ordinary user privileges. However, because **bin** owns the very important directories **/bin**, **/usr/bin**, **/lib**, and **/etc**, any

	mistakes you make while logged in as **bin** could affect the entire system. **bin** also owns the directory that contains the special files for system disks, which are write and read protected from other system users. When you run a command that must read system disks (for example, **df**), the system sets the user name associated with that command to **bin**. (Such commands are called ***suid*** [set user id] programs; the **suid** feature can work with any user name, including root, depending upon the requirements of the program.)
users	The user **users** (or **adduser**) is limited to one function: creating and maintaining user accounts. After logging in, **users** runs the account maintenance program **users** (which is described in detail under "Creating, Changing, and Removing Accounts—The users Command" on page 2-16).
adm	The user **adm** owns the administrative and accounting files in **/usr/adm**. The **adm** user allows you to isolate billing from other system management tasks.

In addition to these special user accounts, there is also a special user group—the system group, or group **0**. Certain commands (in the directory **/etc**) which are not available to regular users can be run by any member of the system group.

Two good reasons for having a system group are to:

- Limit the number of users who need to know the root password. Members of the system group can do many system management jobs without obtaining superuser authority. However, because members of the system group do not have superuser authority, any mistakes they make are less likely to cause serious problems.

- Increase the level of security provided to ordinary system users. Superuser authority gives a user complete access to every file on the system. Members of the system group do not have such access. Thus, members of the system group can perform system management tasks with less chance of compromising the security of data stored on the system.

Using Different Log In Names

Once you are logged in to the system, the most convenient way to change the account name you are using is with the **su** (switch user) command. To obtain superuser authority with the **su** command, simply enter **su** and then enter the superuser password when the system prompts you for it:

```
$ su
password:
# _
```

Notice that when you obtain superuser authority, the shell prompt changes from $ to #. To return to your original user name, press **END OF FILE**.

You also can use **su** to change your user name to that of any other user on the system, provided you know the required password:

```
$ su username
password:
$ _
```

Notice that you must supply the **su** command with *username* any time except when you are obtaining superuser authority. To return to your original user name, press **END OF FILE**.

For more information about the **su** command, see **su** in *AIX Operating System Commands Reference.*

When you use a number of different accounts, and especially if you often change from one account to another with the **su** command, you may forget either your current user name or the user name you used when you logged in to the system. Use the **id** command to determine your current user name:

```
# id
uid=0(root) gid=0(system)
#
```

Regardless of your current user name, you can use the **logname** command to determine the user name you used to log in to the system:

```
$ logname
jones
$
```

You can also use the **who am i** command to determine your user name:

```
$ who am i
jones      console       Apr 4  15:04
$
```

In addition to your user name, the **who am i** command also gives you the name of the display station you are using (**console** in this example) and the date and time that you logged in.

User Account Files

To determine the authority associated with a user account, the system refers to two files:

/etc/passwd

/etc/group

The next two sections explain the information these files contain and how you can modify these files as necessary to maintain user accounts on your system.

The /etc/passwd File

The **/etc/passwd** file contains all information that defines a user and contains one line for each user on the system. Each line contains seven fields separated by : (colons), for example:

```
bud:Ym2CnRPol7Dyg:200:1:Bud Smith/3000;tech:/u/bud:/bin/sh
 |       |         |  |        |               |      |
 1       2         3  4        5               6      7
```

Figure 2-1. The Fields in a Password File Entry

The fields, from left to right, are:

1. User name
2. Encrypted password
3. User number (**UID**)
4. Group number (**GID**)
5. Optional information field
6. Home (login) directory
7. Initial program (***login shell***).

Following are descriptions of the information in each field:

User name
The **user name** field contains the name (up to eight characters long) with which a particular user logs in to the system. The system always uses the name in this field to refer to that user.

Encrypted password
If this field is empty, the user does not have a password. (A user without a password does not receive a `password` prompt.) It is always good practice to use passwords, even on a single-user system.

If a user forgets their password, you can give them access to the system by deleting the password field from the line for their account in **/etc/passwd**. Whenever you delete a password, encourage (or require) the user to set a new password immediately. (For information on how to change a passwd, see *Using the AIX Operating System.*) You can also use the **users** command to set a new password for a user once you have deleted the old password field from **/etc/passwd**.

A password field may also contain optional **aging** information, separated from the encrypted password by a comma. Aging information consists of encoded numbers that specify the maximum and minimum age (in weeks) for the password and the week during which the password was last changed. A user whose password expires is forced to set a new password before logging in to the system. Only a user with superuser authority can change a password before it is at least as old as the minimum age. If security is a major concern on your system, you can use the aging feature to force users to change their passwords frequently. To force a user to set a password before beginning to work on the system, set the maximum and minimum times to zero and make sure there is not an indication of the time the password was last changed. The **users** command makes it easy to arrange for all new accounts to have standard aging parameters, or to change the aging time limits for a selected user when you suspect that a password is no longer secure.

User number and **Group number**

The user number (**UID**) and group number (**GID**) define the identity of users for security purposes. All file protection on the AIX system is based on three sets of permissions, those for **owner**, **group**, and **others**. Every process started by a user is identified with that user's user number and group number. Thus, the user number and group number together with permissions make it possible to control the files to which a particular user has access.

Optional information

The fifth field of the password file contains assorted information related to the user. It consists of three optional subfields in the form:

fullname/filesize;siteinfo

fullname is the real name of the person whose user name appears in the first field.

filesize specifies the maximum size (in blocks) of files that the user is allowed to create. If you do not set the maximum file size, you can omit the separator (/), and a standard, system-wide value will apply to that user's files. (For more information on setting maximum file size, see **ulimit**, described under **sh**, and **login** in *AIX Operating System Commands Reference.*)

siteinfo can contain whatever information you wish to include. If you do not include the *siteinfo* field, you can omit the separator (;).

Home (log in) directory

The sixth field contains the name of the user's home (login) directory. The **login** program places the user in this directory at login. Typically, this directory is owned by the user and all the user's private files are kept in the directory tree below it. When you add a new user to the system, the **users** command automatically creates a login directory for the user.

Initial program (log in shell)

The last field in the password file contains the name of the program that the user runs upon logging in (the login shell). Usually this is the **/bin/sh**, or shell program (described in *Using the AIX Operating System* and under **sh** in *AIX Operating System Commands Reference*). If this field is blank, the system automatically starts the **/bin/sh** program for that user. You can specify another program in this field if, for example, the user requires a different command interpreter.

Figure 2-2 on page 2-38 shows a typical password file. The entries for certain users (for example, **root** and **bin**) appear on all systems, but with different encrypted passwords. This sample password file includes users in different groups and users with special login programs:

```
root:8eCGXCGuI6HR2:0:0::/:
users:z1Bclk147ta4Q:0:1::/usr/adm:/etc/adduser
daemon:mchxEqsYbPOMY:1:1::/:
bin:tb1tZt2qTqfuk:2:2:/3000:/bin:
adm:7dyLNacEX9oss:4:4::/usr/adm:
sync:4U8zktFt.GEj6:20:1::/:/bin/sync
chris:Wjpfqst3yXpOQ:201:1:Chris Cooper;tech:/u/chris:
heinz:mIk2beSZBxVq6:202:1:Heinz Hart;mgmt:/u/heinz:
ted:w.UVCyYgX28Es:203:1:Ted Black;tech:/u/ted:
jim:ckSsEP4GH/cwE:204:1:Jim Mori;tech:/u/jim:
bud:Ym2CnRPol7Dyg:200:1:Bud Smith/3000;tech:/u/bud:/bin/sh
```

Figure 2-2. Sample /etc/passwd File

It is good practice to reserve some number of user numbers (50 to 200) for use by programs that are created on your system and require special privileges. The remaining user numbers then are available for ordinary users.

The Group File

You can assign a user to one or more groups. Each group shares certain protection privileges. For example, you may want to place users in the same group because they work on the same project and need access to a common set of files. Or you may create a group (like the system group) to give certain users special privileges. Thus, you can use groups to increase or restrict the privileges of the users assigned to them.

When a user logs in, the system assigns the user to the group specified by the group number in that user's entry in **/etc/passwd**. The user can then use the **newgrp** command (described in *AIX Operating System Commands Reference*) to change the group associated with all processes the user creates. The group under which the user is currently working is called the primary group.

The AIX system allows users to belong to several groups at the same time. A user's primary group is the one specified in the **/etc/passwd** file. However, you can use the **users** command (described under "Creating, Changing, and Removing Accounts—The users Command" on page 2-16) to add users to other groups. (Also use **users** to create new groups.) Any files created by a user belong to that user's primary group, but the user has access to all files accessible to any group to which he belongs. To change the primary group, use the **newgrp** command (described under **newgrp** in *AIX Operating System Commands Reference*).

The file **/etc/group** defines which users make up each group. Each line in **/etc/group** defines a group and consists of four fields separated by colons:

1. Group name
2. Encrypted password
3. Group number (GID)
4. Permission list (user names separated by commas).

The **group name** is a string of up to eight characters used to refer to the group. If there is a **password**, any user who attempts to enter the group is required to use it. The **group number** is simply the number assigned to that group. The **permission list** contains all users who have permission to enter the group, for example:

```
system::0:root,bin,jim,steve
staff::1:
managers::200:heinz,ted
```

The **users** command usually puts new users in the **staff** group. You can add users to groups and create new groups as necessary. It may be useful to establish a group for new users; such a group, often called **other** allows new users to work on the system before they are assigned to a particular group. As with user numbers, it is good practice to reserve some group numbers for uses unique to your system (perhaps 50 to 200 in all, with the first 10 to 20 reserved for use in system management tasks). The remaining group numbers can be assigned to users.

For users to change their primary group, they either must be on the list of users permitted to enter that group or, if the group has a password, know the password for that group. The one exception to this rule is that a user always can return to the group specified in the password file (the primary group). That is, a user does not need to give a password or appear in the permission list in order to return to their primary group.

Note: Group passwords are rarely necessary. In practice, the permission list usually provides adequate group security, and most systems do not use group passwords.

Tailoring the User Environment

Certain variable factors control how the shell operates. The way these variables are set determines the shell's ***environment***. Several files control the environment and thereby control many of the programs that users run. While the AIX system includes prototypes for these files, you may need to change the prototypes, or add to them, to adapt the system to your needs.

This section will be easier to understand if you are somewhat familiar with the AIX shell. For information about the AIX shell, see *Using the AIX Operating System*.

The shell provides named variables and mechanisms for assigning string values to them, testing them, and substituting them into commands. In addition to their use as program variables for the shell as a high-level programming language, some of these variables control how the shell works. Furthermore, **exported** shell variables are passed in the process environment from the shell to the programs that it runs as user commands. Since exported variables can affect how any program runs, they provide potentially wide-ranging control over the user environment. Some of the commonly used environment variables are discussed under **sh** in *AIX Operating System Commands Reference*; others are discussed in the same book in conjunction with specific commands. This section discusses where the shell obtains exported variables other than those explicitly exported by the user.

/etc/environment

Before any user logs in to the system, the **init** process (the main process involved in starting the system) reads the file **/etc/environment**. The **init** process passes variable assignments made in **/etc/environment** to each process it creates (its ***child*** processes). These variables automatically become part of the list of exported shell variables. To modify the **/etc/environment** settings, use a text editor to change the **/etc/environment** file.

Note: **/etc/environment** can contain only variable assignments, not shell commands.

Unless the values shipped with your system are correct, you should set at least the following variables in **/etc/environment**:

TZ This **TZ** (time zone) variable controls how your system translates its standard time (Universal Coordinated Time, also known as Greenwich Mean Time) into local time. The form of the TZ variable assignment is:

```
TZ=nnnhddd
```

- *nnn* is the three-character name of your normal time zone.
- *h* is the number of hours by which your time zone normally lags behind standard system time. For locations east of Greenwich, you can use a variable of the form *-h* to specify the number of hours by which your time zone leads standard system time.
- *ddd* is the three-character name for your time zone during daylight savings time. (Omit this field if daylight savings time is not observed locally.)

Some typical TZ assignments are:

TZ=EST5EDT	**E**astern Standard Time, **5** hours behind Greenwich, **E**astern **D**aylight **T**ime.
TZ=CST6CDT	**C**entral Standard Time, **6** hours behind Greenwich, **C**entral **D**aylight **T**ime.
TZ=MST7MDT	**M**ountain Standard Time, **7** hours behind Greenwich, **M**ountain **D**aylight **T**ime.
TZ=PST8PDT	**P**acific Standard Time, **8** hours behind Greenwich, **P**acific **D**aylight **T**ime.

TZ=GMT0 Greenwich Mean Time, **0** hours behind Greenwich. Note that, as this example shows, when daylight savings time is not observed locally, the *ddd* field must be omitted.

TZ=CET-1 Central European Time, **1** hour before Greenwich. Note that *h* is specified hours before Greenwich and, again, the *ddd* field is omitted.

PATH Executable files (programs or commands) are located in various directories. The **PATH** variable contains a list of directory names (separated by colons) that the shell and other commands use when searching for an executable file. In the PATH assignment, a : (colon) that is not preceded by a directory name stands for the current directory. If the PATH variable is not otherwise set, its default assignment is:

```
PATH=:/bin:/usr/bin:/etc::
```

With this assignment, a command searches the current directory (:) first, then the directory **/bin**, and finally the directory **/usr/bin**.

Two common reasons for setting the PATH variable to something other than its default are:

- To increase the efficiency of searches. In some cases, a PATH that searches the current directory last is most efficient. The following assignment causes the current directory to be searched last: **PATH=/bin:/usr/bin:/etc::**

- To include other directories in the search. For example, if you have a directory named **/local** containing commands that you have created, you can include it in the search path with the assignment **PATH=/bin:/usr/bin:/local:/etc::**

umask The file creation mask, **umask**, controls the **mode** (permissions) set for new files by specifying what permissions are not given. **umask** is set by the **umask** command in **/etc/profile**.

If no **umask** were set, files would be created with mode **777** (read, write, and execute permission for **owner**, **group**, and **others**). Typical settings of the **umask** variable are:

umask=022 Files are created in mode **755**. **Group** and **others** do not have write permission.

umask=002 Files are created in mode **775**. **Others** do not have write permission.

umask=007 Files are created in mode **770**. **Others** do not have permissions; **owner** and **group** have all permissions.

For a general discussion of permissions, see *Using the AIX Operating System.*

filesize The **filesize** variable sets a default value for the system-wide maximum file size (in 512-byte blocks). The default file size limit is 2048 blocks or slightly more than one million bytes. You can override the default setting for individual users by setting a different value for **filesize** in the **/etc/passwd** file. (For more information on setting maximum file size, see **ulimit** described under **sh** and **login** in *AIX Operating System Commands Reference.*)

Login

The **login** program adds the following values to the environment:

LOGNAME (user name)

TERM (display station, or **terminal**, type from **/etc/ports**)

HOME (log in directory).

/etc/profile and $HOME/.profile

An initial user shell includes in its environment the variables passed to it by **login**. Then, before the shell accepts any input from the user, it runs the commands in **/etc/profile** and in the file **.profile** in the user's login directory. Thus, **/etc/profile** should contain any commands that all users need to run when they log in (for example, the **news -n** command, which informs users of items in the system news facility). At the same time, **/etc/profile** should not contain nonessential commands. (Although the system runs all commands in **/etc/profile**, individual users can override the variable assignments made in **/etc/profile** with assignments in their private profile files, **$HOME/.profile**.)

Two environment variables should be set in **/etc/profile** and marked for export: **MAIL** and **MAILMSG**. The shell checks the last modification date of the file **MAIL** to determine whether the user should be notified of the receipt of new mail, and **MAILMSG** is the notification message itself. If you use the MAIL facility on your system, the following variable assignments should appear in **/etc/profile**:

```
MAIL=/usr/mail/$LOGNAME
MAILMSG="[You have new mail]"
export MAIL MAILMSG
```

You also may set the **TIMEOUT** variable in **/etc/profile**. The value of **TIMEOUT** is specified in minutes. When a user remains at command level for *TIMEOUT* minutes without entering a command, the shell exits and logs off the user. Generally, the **TIMEOUT** variable is most useful on systems that serve a number of users or on systems where security is a major concern (where, for example, a display station left unattended for a long time could give unauthorized people access to classified data).

If there are commands that you believe most users will want to run when they log in, but that are not appropriate for **/etc/rc**, you can create a default **.profile** file to be added automatically to the home directories of new users. The **users** command and the autolog function makes it possible to set up default **.profile** files through the **/usr/adm/newuser.usr** command file, which runs when a user is added to the system. For more information on **users**, see "Managing User Accounts" on page 2-16.

Information about File Systems—The /etc/filesystems File

Information about file systems is stored in the file **/etc/filesystems**. Most of the file system maintenance commands use **/etc/filesystems** to relate file system names to corresponding devices and to provide information about file systems to those commands.

The **/etc/filesystems** file is organized into ***stanzas*** that describe file systems. A stanza has the same name as the file system that it describes, and contains a series of *attribute* = *value* pairs that specify the characteristics of the file system. Stanzas named **default** contain information common to several file systems.

Following is part of a sample **/etc/filesystems** file:

```
*   This file describes all of the known file systems.  It
*   is used by most of the file system maintenance routines
*   to map file system names into the corresponding device
*   names.  It also provides the information which tells fsck,
*   df, mount and other programs what file systems to operate
*   on by default.

default:
        vol        =  "AIWS"
        mount      =  false
        check      =  false
        free       =  false
        backupdev  =  /dev/rfd0
        backuplen  =  720

/:
        dev        =  /dev/hd0
        vol        =  "root"
        mount      =  automatic
        check      =  false
        free       =  true

/mnt:
        dev        =  /dev/fd0
```

Stanzas can contain attributes in addition to the ones shown in the example. Following is a list of the possible **/etc/filesystems** attributes and a brief explanation of their meanings:

account Determines file system to be processed by the accounting system. The value can be either **true** or **false**.

backupdev Used by the **backup** and **restore** commands to determine the backup device associated with each file system. The value is usually the name of a diskette or magnetic tape special file.

backuplen Used by the **backup** command to determine the size of the default backup device associated with each file system.

backuplev Used by the **backup** command to determine the default backup level for each file system. For an explanation of backup levels, see "Backing up Files and File Systems" on page 2-58.

boot Used by the **mkfs** command to initialize the boot block of a new file system. The value of **boot** specifies the name of the load module to be placed into the first block of the file system.

check Used by the **fsck** command to determine the file systems to be checked. The value can be **true**, **false**, or some number that corresponds to a particular phase of the **fsck** program. For more information about **fsck**, see "Checking and Repairing File Systems—The fsck Command" on page 3-8.

cyl Used by the **mkfs** command to initialize the free list and superblock of a new file system. The value is set the number of blocks in one cylinder.

dev Identifies the block special file where the file system resides. System management programs use this attribute to relate file system names to the appropriate device names.

free	Used by the **df** command to determine which file systems are to have their free space displayed. This value is set either **true** or **false**.
mount	Used by the **mount** command to determine whether to mount the file system. If the value of **mount** is **true**, then the **mount all** command mounts the file system. The most convenient way to accomplish routine file system mounting is to place the **mount all** command in the **/etc/rc** file. The value of mount can be set **true**, **false**, or **readonly**. When the **readonly** value is set, the file system is mounted, but its contents cannot be changed. The **mount** attribute for the root file system has a special value: **automatic**. The **automatic** value causes the root file system to be mounted whenever the system is initialized and prevents the **mount all** command from attempting to mount the already-mounted root file system.
size	Used by the **mkfs** command to specify the size of a new file system. The value of **size** is some number of 512-byte blocks.
skip	Used by the **mkfs** command to initialize the free list and superblock of a new file system. The value of **skip** is some number of 512-byte blocks to be skipped between data blocks (the ***interleave factor***).
vol	Used by the **mkfs** command when it initializes the label on a new file system. The value is a volume label with a maximum length of six characters.

Creating and Mounting File Systems

A file system is a complete directory structure confined to a single minidisk or diskette. Before you can use a minidisk or diskette to store data, create a file system on it. Use the **mkfs** (make file system) command to create file systems.

Note: The **mkfs** command is not the only way to create a file system on a minidisk. For more information on creating file systems, see *Installing and Customizing the AIX Operating System.*

The general form of the **mkfs** command is:

/etc/mkfs *name size*

where *name* is the name of the minidisk on which the file system is to be created and *size* is the total size of the file system in 512-byte blocks. Allocate more space than you need because the i-nodes use up to 5% of the blocks. In order to create a new file system, the **mkfs** command must have the following information:

dev	The name of the device on which the file system is to be created.
size	The size of the new file system in 512-byte blocks.
vol	A volume label.
cyl	The number of 512-byte blocks per cylinder on the disk. This parameter is used with **skip** to form the **interleave specification**. For an explanation of the interleave specification, see **mkfs** in *AIX Operating System Commands Reference.*
skip	Used with **cyl** to form the **interleave specification**.
boot	The name of a file containing a **bootstrap** (initialization) program to be loaded into the boot block of the new file system.

Note: "Information about File Systems—The /etc/filesystems File" on page 2-47 discusses these and other file system attributes more fully.

You can supply these parameters to **mkfs** in one of three ways:

- From information contained in **/etc/filesystems**.

  1. Use a text editor to create a stanza for the new file system in **/etc/filesystems**. The most convenient way to create this stanza is to copy a similar stanza, give it the label you want to assign to the new file system, and then modify it as necessary. Save the modified **/etc/filesystems** file.

  2. At the $ (shell) prompt, enter the command `/etc/mkfs` *label*.

  A stanza in **/etc/filesystems** contains, in one place, all of the information that defines that file system. Thus, if you need to know something about how a file system is defined, it is very easy to get the information you need if the file system has a stanza in **/etc/filesystems**.

- On the command line along with the **mkfs** command. For information on supplying parameters in this way, see **mkfs** in *AIX Operating System Commands Reference.*

- From the special file for the device on which the file system is to be created. If you do not supply the parameters for the new file system in one of the ways just described, **mkfs** takes them, if possible, from the device driver for the device on which the file system is to be created. If you want to take the parameters from a device driver, simply specify the path name for the device on which you want to create the file system. For example, the following command creates a file system on the **/dev/hd0** device:

  ```
  /etc/mkfs /dev/hd7
  ```

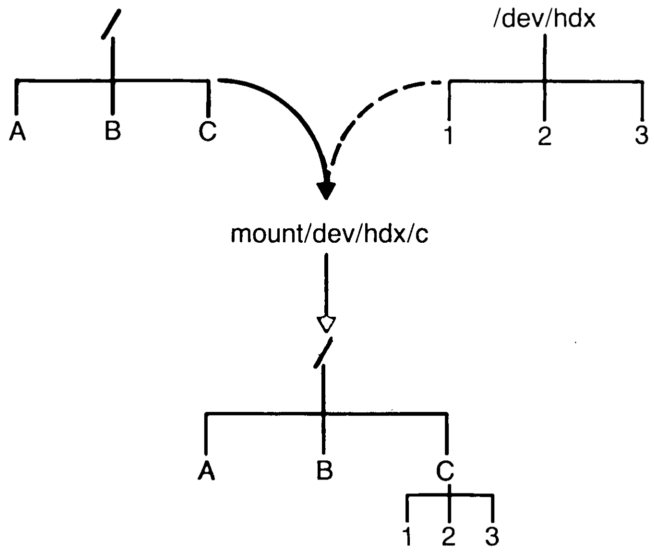
  You cannot create a file system on a device that is not already defined in **/dev**.

Mounting and Unmounting File Systems

File systems are independent from each other and from the operating system. Each file system is associated with a different **device** (a minidisk or a diskette). Before you can use a file system, it must be connected to the existing directory structure (either to the root file system or to another file system that is already connected). The **mount** command makes this connection. During the start up process, the system automatically mounts the file systems with the line `mount = true` in their **/etc/filesystems** stanza. However, you must use the **mount** command to mount any file system that:

- Does not have a stanza in **/etc/filesystems**
- Has the line `mount = false` in its **/etc/filesystems** stanza.

The **mount** command attaches one file system to another at a specified directory.



AUS105152

Figure 2-3. Mounting a File System

The general format for the **mount** command is:

```
mount directory
```

where *directory* is the name of the directory on which the file system is to be mounted. For example, to mount a file system identified in **/etc/filesystems** as `/controls` (`mount = false`), enter:

```
mount /controls
```

If you have superuser authority, you also can specify the device that contains the file system to be mounted. The format for **mount** when you specify both the device and directory is:

```
mount device directory
```

For example, to mount the file system on device **hd9** on the directory `controls`, use the command:

```
mount /dev/hd9 /controls
```

If there is a stanza in **/etc/filesystems** for the directory to be mounted, simply specify the name of the stanza with the **mount** command. For example, **mount /u** mounts the file system described by the following **/etc/filesystems** stanza onto the **/u** directory:

```
/u:
        dev       =  /dev/hd1
        vol       =  "/u"
        mount     =  true
        check     =  true
        free      =  true
```

Notice that the **mount** attribute in this stanza is set to **true**, which means that this file system is automatically mounted by the **mount all** command in **/etc/rc**. Thus, under normal circumstances, you do not have to use **mount** to mount this file system.

After you mount a file system, *directory* functions as the root directory of that file system. Also, the operating system records the presence of this file system in the file **/etc/mnttab**.

Use the **mount** command by itself (without *device* or *directory*) to list what file systems are mounted, where they are mounted, when they were mounted, and whether they are **writable** (that is, whether their contents can be modified).

You can disconnect mounted file systems with the **umount** command. The general format for the **umount** command is `umount` *filename*, for example:

```
umount /dev/fd2/accounts
```

where *filename* is either the special file for a mounted device or the name of the directory on which a device is mounted. To unmount all mounted file systems, use the command `umount all`.

Creating and Mounting Diskette File Systems

In general, the procedures for creating and using file systems are the same, whether the file systems reside on minidisks or diskettes. However, before you use diskette file systems, you should be aware of the information in this section.

Notes:

1. If you are not familiar with the proper way to handle diskettes, please read the material on diskette handling in *Guide to Operations* before you begin to use diskettes for AIX file systems.

2. If you want to transfer data to or from a DOS formatted diskette, see "Using DOS Formatted Diskettes" on page 2-56.

Formatting Diskettes

Before you can create a file system on a diskette, you must use the **format** command to prepare the diskette to contain data.

Note: Formatting erases any data stored on the diskette.
Following is the procedure for formatting diskettes:

1. Insert a diskette into the diskette drive. If you have more than one diskette drive, insert the diskette into drive 0 (A).

2. Enter: `format`

For more information about the **format** command, see **format** in *AIX Operating System Commands Reference.*

Creating Diskette File Systems

After you have formatted a diskette, you must create a file system on it:

1. Insert a formatted diskette into the diskette drive (or simply leave the diskette in the drive if you have just formatted it). If you have more than one diskette drive, insert the diskette into the drive 0 (A).

2. Enter: `/etc/mkfs /dev/fd0`

Mounting and Unmounting Diskette File Systems

To use a diskette file system, insert the diskette into the diskette drive and enter a command of the form:

`mount` *directory*

The standard directory for mounting a diskette file system is **/diskette0**. (If you have a second diskette drive, it is associated with the directory **/diskette1**.)

Warning: Do not remove a mounted diskette. Damage to the diskette file system could result.

When you finish your work with the diskette file system, unmount it with a command of the form:

```
umount directory
```

In the **/etc/filesystems** stanza for **/diskette0**, the value for **mount** is **true, removable**. Thus, the system does not automatically mount the diskette file system at startup.

If you want to mount a diskette on a directory other than **/diskette0**, use a **mount** command of the form: **mount** *device directory*. For example, if you want to mount a diskette file system on the directory **/mountfs**, insert the diskette in drive 0 and enter:

```
mount /dev/fd0 /mountfs
```

Using DOS Formatted Diskettes

With the **dosread** and **doswrite** commands, you can use diskettes to transfer data between the AIX system and a computer that uses the Disk Operating System (DOS). The **dosread** command copies a DOS file from the diskette to the AIX file system. The **doswrite** command copies a AIX file to a DOS diskette. Both commands perform the necessary translations required to make the data usable on the destination system.

The most common form of the **dosread** command is:

```
dosread -a dosfilename filename
```

where:

> **-a** specifies translations usually required for text files.
>
> *dosfilename* is the name of the file you want to copy from the DOS diskette.
>
> *filename* is the name you want to copy the file to in the AIX file system.

The most common form of the **doswrite** command is:

```
doswrite -a filename dosfilename
```

where:

-a specifies translations usually required for text files.

filename is the name of the file you want to copy from the AIX file system.

dosfilename is the name you want to copy the file to on the DOS diskette.

For more information, see **dosread** and **doswrite** in *AIX Operating System Commands Reference.*

Backing up Files and File Systems

Once your AIX system is set up and in use, your next consideration should be backing up the file systems (there is one file system per minidisk). Whether due to system malfunction or user error, file systems and the data they contain can be damaged or lost. If you take a careful and methodical approach to backing up your file systems, you should always be able to restore recent versions of files or file systems with little difficulty. This section discusses different file and file system backup procedures and how those procedures can be combined into a dependable backup policy.

Types of Back Ups

Backup procedures rely upon the following commands:

cvid Use the **cvid** command to backup the VRM minidisk. You should always have a backup of the current VRM minidisk. Before you modify the VRM minidisk (for example, by installing a licensed program or an update, or using the **mvmd** command), make certain that you have a backup of the current VRM minidisk. If not, backup the VRM minidisk before you make the changes. Always make a new backup of the VRM minidisk after you modify the VRM minidisk. The VRM minidisk backup procedure is explained under "Backing Up the VRM Minidisk" on page 2-62.

backup Use the **backup** command to backup individual files or entire file systems. "Individual File Backup" on page 2-68 explains how to backup individual files. "Using the backup and restore Commands" on page 2-64 explains how to backup complete file systems with the **backup** command.

To use **backup** from the maintenance system, select `Backup commands` from the **USE MAINTENANCE COMMANDS** menu, and then select `Back up a file system`.

Note: When you select `Back up a file system` from the **USE MAINTENANCE COMMANDS** menu, the backup is by i-node only. (For information about the maintenance system, see "Running the Maintenance System" on page 2-6.)

restore Use the **restore** command to read files backed up with **backup** to a device or directory. For information on how to restore individual files, see *Using the AIX Operating System.* "Using the backup and restore Commands" on page 2-64 explains how to restore complete file systems. (The **restore** command can restore individual files from a complete file system backup.)

To use **restore** from the maintenance system, select `Restore commands` from the **USE MAINTENANCE COMMANDS** menu, and then select `Restore a file system`.

Note: When you select `Restore a file system` from the **USE MAINTENANCE COMMANDS** menu, the restore is by i-node only. (For information about the maintenance system, see "Running the Maintenance System" on page 2-6.)

dd Use the **dd** (device-to-device copy) command to backup entire file systems. The **dd** command is a faster way to backup entire file systems. However, you cannot restore individual files from a **dd** backup. "Backing Up Complete File Systems with the dd Command" on page 2-71 explains how to perform file system backups with the **dd** command.

You also can use **dd** from the **USE MAINTENANCE COMMANDS** menu to:

- Backup a file system.

  Select `Backup commands` and then select `Back up a minidisk image.`

- Restore a file system.

Select `Restore commands` and then select `Restore a minidisk image`.

(For information about the maintenance system, see "Running the Maintenance System" on page 2-6.)

It is very important for you to understand how to use these commands to protect the users of your system from loss of data. You should be familiar with the information in:

- "Running the Maintenance System" on page 2-6
- "Backup Media"
- "Backing Up the VRM Minidisk" on page 2-62
- **cvid** in *AIX Operating System Commands Reference*
- "Using the backup and restore Commands" on page 2-64
- **backup** in *AIX Operating System Commands Reference*
- **restore** in *AIX Operating System Commands Reference.*
- "Backing Up Complete File Systems with the dd Command" on page 2-71
- **dd** in *AIX Operating System Commands Reference*

Backup Media

Diskettes are the standard backup medium. Unless you specify a different value for **backupdev** in **/etc/filesystems**, the **backup** command automatically writes its output to **/dev/rfd0**, the device name for the diskette drive.

When you install a streaming tape drive, the **devices** command automatically changes the value of **backupdev** to **/dev/rmt0**, the device name for the tape. The **rmt0** value makes the tape rewind when it is finished backing up a file system. If you change the value to **rmt4**, the tape does not rewind.

Note: You may need to change the value for **backuplen**. For example, a **backuplen** of 2700 represents a 300-foot tape with nine tracks (300x9 = 2700).

If you remove a tape drive from the system, **devices** changes the value of **backupdev** back to **/dev/rfd0** and the value of **backuplen** back to 2400.

After you install a tape device, the **backup** command uses the tape as the backup device for complete file system backups (the procedure described under "Using the backup and restore Commands" on page 2-64). However, even with a tape installed, the **backup** command still backs up individual files to diskettes (for information about file backups, see *Using the AIX Operating System*).

Backing Up the VRM Minidisk

Use the **cvid** command to backup your VRM minidisk files onto a diskette. You should always have a backup of your current VRM. With a current VRM backup, you are prepared to reinstall the most recent version of the VRM should a system failure or other error require it.

Backup the VRM immediately after you install it and back it up again after each change you make to it. The following commands modify the VRM, and you should backup the VRM after using any of them:

installp	Modifies the VRM when you install a licensed program.
updatep	Modifies the VRM when you install an update.
mvmd	Modifies the VRM directly, by adding, changing and deleting files. Generally, **mvmd** is used by **installp** and **updatep**; that is, you do not ordinarily enter **mvmd** commands on the command line.

(For more information on **installp**, **updatep**, and **mvmd**, see *Installing and Customizing the AIX Operating System* and *AIX Operating System Commands Reference.*)

This section describes how to use **cvid** to create a VRM diskette. You must have superuser authority or be a member of the system group to use the **cvid** command. (See *AIX Operating System Commands Reference* for a discussion of the **cvid** command. For an explanation of any error messages that may occur as you use this command, see *Messages Reference.*)

Note: Before you change the VRM, make certain that you have a backup of the current VRM. If not, backup the VRM before you change it.

The command format is:

```
cvid dev -f -v proto
```

where:

dev	Specifies the device on which the file system is created, as in fd0 (diskette drive 0). It can also be the name of a file system.
-f	Specifies the file system label for the new file system. The default label is **vrmmnt**.
-v	Specifies volume label for the new file system. The default volume label is **VRMIBM**.
proto	Specifies the name of the prototype file. Default is **/vrm/vproto**.

To Back Up Your VRM Minidisk

1. Insert a formatted diskette into the diskette drive.

2. After the system prompt, type `cvid`, plus any needed parameters.

3. Press **Enter**.

4. When the VRM files have been copied to the diskette, remove the new VRM install diskette and store it in a safe place.

While the **cvid** command is running, the system creates a file system, using the **mkfs** command, and copies files from your VRM minidisk.

Using the backup and restore Commands

The **backup** command allows you to backup files selectively or to backup entire file systems. Similarly, you can use the **restore** command to restore all or part of the backups you create with **backup**.

Both **backup** and **restore** are available from the **USE MAINTENANCE COMMANDS** menu in the maintenance system. For information on the maintenance system, see "Running the Maintenance System" on page 2-6.

Note: If you only want to backup and restore complete file systems, it may be faster to use the **dd** command than it is to use **backup** and **restore**. Among the factors that determine which backup method is faster are the size of the minidisk to be backed up and how many of its blocks are allocated.

This section explains how to use **backup** and **restore** for complete file systems.

Note: If you use diskettes as your backup media, have several formatted diskettes ready to use before you enter the **backup** command.

To Backup a File System with backup

1. Prepare the backup medium:

   - Make certain that the tape device is ready to operate, or

   - Insert a formatted diskette into drive 0.

2. Enter:

   ```
   backup -0 -u filesystem
   ```

where *filesystem* is the name of the file system to be backed up.

(The system prompts you for additional backup media as required).

To Restore a File System

1. Load the tape or diskette containing the file system to be restored. (If you restore from diskettes, and the backup occupies more than one diskette, insert the first diskette of the group into drive 0).

2. Enter:

   ```
   restore -r filesystem
   ```

   where *filesystem* is either the name of a physical device or a directory name listed in the **/etc/filesystems** file. The **-r** flag is for i-node backups only.

(The system prompts you for additional media as necessary, for example, for the second of a group of diskettes.)

Volume Backups

The **backup** command individually backs up each file in a file system. Thus, you do not have to restore an entire file system if all you need to do is restore specific files. Also, you can restore a file or group of files to any file system that is large enough to accommodate them.

To backup an entire file system, use a command of the following form:

```
backup -0 -u filesystem
```

Following is an explanation of the parts of this command:

-0	A flag that indicates the **level** of the backup. Level **0** causes every file in the file system to be backed up. Use other backup levels (1-9) for incremental dumps, as is explained under "Incremental Backups" on page 2-67.
-u	A flag that causes **backup** to record the date and level of the backup in the **/etc/ddate** file. **/etc/ddate** contains the date of the last backup of each file system at each level.
filesystem	The name of the file system to be backed up. *filesystem* can be either the name of the physical device that contains the file system or the name of the file system's root directory.

Unless you specify otherwise on the command line, the **backup** command uses information in **/etc/filesystems** to determine:

dev	The device, or minidisk, that contains the file system.
backupdev	The device (for example, a diskette drive) on which the backup is to be made.
backuplen	The size in blocks of the backup device (diskette or tape) to be used (for computing the amount of data that will fit on each diskette).

Note: When you restore a complete file system, the **restore** command organizes the files efficiently, but does not reduce the total amount of storage space needed for the file system. The **-m** flag backs up an entire minidisk as an exact image. An image backup is appropriate for backing up very large AIX file systems or minidisks that do not contain AIX file systems. A backup by minidisk (**backup -m**) must be restored by minidisk (**restore -m**).

Incremental Backups

A level 0 backup can require a considerable amount of time and a large number of diskettes to complete. Because only a fraction of the files on your system can change from day to day, it is not necessary for you to make a level 0 backup daily. Instead, you can make **incremental** backups—backups of only the files changed since a previous backup.

An incremental backup backs up all files changed since the last backup at the next lower level. For example, the following level 1 **backup** command backs up all files changed since the last level 0 backup:

```
backup -1 -u filesystem
```

The parameter that indicates the level of the backup (*-1* in this example) can be any number from 0 through 9. Thus, a level 4 backup backs up all files that have changed since the last level 3 backup, and so forth. You can use the backup levels to develop a very dependable backup system. For example, you might make level 0 backups monthly, level 1 backups weekly, and level 2 backups daily. Then, should you have to restore a complete file system, you would restore from the latest level 0 backup, then from the latest level 1 backup, and finally from the latest level 2 backup. For many systems, it should be adequate to use only two levels of backup (for example, level 0 weekly and level 1 daily or level 0 monthly and level 1 weekly).

Individual File Backup

You can also use the **backup** command to backup individual files, for example, when some or all of the files that belong to a particular user will not be needed for an extended period. Backup copies of individual files are also a convenient means for exchanging data among different systems (for example, you backup the files from your system onto a diskette and then the user of another system restores the files from the diskette). To backup individual files, use the **-i** (backup by name) flag with the **backup** command.

The following sequence backs up three files to a backup device defined in **/etc/filesystems**:

```
$ backup -i
Please mount volume 1 on /dev/backup device
. . . and type return to backup
file1
file2
file3
END OF FILE
Done at date and time
n blocks on n volume(s)
$ _
```

You can also use the **backup** command with a list of file names created with an editor. In the following example, **backup** makes backup copies of the files named in the file `list`:

```
backup -i < list
```

To backup all files and subdirectories of the current directory, use the command:

```
find . -print | backup -i
```

Guidelines for Backup Policies

No single backup policy can meet the needs of all AIX system users. A policy that works well for a system with one user, for example, could be inadequate for a system that serves five or 10 different users. A policy developed for a system on which many files are changed daily would be inefficient for a system on which data changes infrequently. Only you can determine the best backup policy for your system, but the following general guidelines should help:

- **Make sure you can recover from major losses.**

  Can your system continue to run after any single fixed disk fails? Can you recover your system if all of the fixed disks fail? Could you recover your system if you lost your backup diskettes or tape to fire or theft? Although these things are not likely, any of them is possible. Think through each of these possible losses and design a backup policy that would enable you to recover your system after any of them.

- **Use your backups periodically.**

  Diskettes, diskette drives, and tape devices can be unreliable. A large library of backup tapes or diskettes is useless if their data cannot be read back onto a fixed disk. Thus, it is a good idea to make certain that your backups are useable. Try to restore files periodically (for example, to **/dev/null**), just to make sure that they can be read. If you use diskettes for your backups and have more than one diskette drive, try to read diskettes from a different drive than the one on which they were created. Therefore, you may want the security of repeating each level 0 backup with a second set of diskettes. If you use a streaming tape device for backups, you can use the **tapechk** program to perform rudimentary consistency checks on the tape. For more information about the **tapechk** command, see **tapechk** in *AIX Operating System Commands Reference.*

- **Keep old backups.**

  You probably will develop some cycle for re-using your backup media–tapes or diskettes. However, you should not re-use all of your backup media. Sometimes it may be months before you, or some other user of your system, notices that an important file is damaged or missing. You should save old backups for just such possibilities. For example, you could have three cycles of backup tapes or diskettes:

  - Once per week, recycle all daily diskettes except the one for Friday.

  - Once per month, recycle all Friday diskettes except for the one from the last Friday of the month. This makes the last four Friday backups always available.

  - Once per quarter, recycle all monthly diskettes except for the last one. Keep the last monthly diskette from each quarter indefinitely, perhaps in a different building.

- **Check file systems before backing them up.**

  A backup that was made from a damaged file system may be useless. Before making your backups, it is good policy to check the integrity of the file system with the **fsck** command (covered under "Maintaining the File System" on page 3-4).

Your system should not be in use when you make your backups. If you backup a file system while it is in use, files can change while they are being backed up. The backup copy of such a file would not be accurate.

Finally, it is always good policy to backup your entire system before any hardware testing or repair work is performed or before you install any new devices, programs, or other system features.

Backing Up Complete File Systems with the dd Command

When you want to backup (and restore) only complete file systems, the **dd** (device-to-device copy) command gives you a potentially faster alternative to **backup** and **restore**. The **backup -m** command actually uses **dd** and is a convenient way to backup file systems. However, this section describes how to use the **dd** command.

The **dd** command is available from the **USE MAINTENANCE COMMANDS** menu in the maintenance system. To use **dd** from the maintenance system:

- To backup a file system, select `Backup commands` and then select `Back up a minidisk image.`

- To restore a file system, select `Restore commands` and then select `Restore a minidisk image.`

For information on the maintenance system, see "Running the Maintenance System" on page 2-6.

To backup a file system with **dd**, you need to specify the following parameters:

if	The name of the input file. Use the value of **dev =** in the **/etc/filesystems** stanza for the appropriate file system. **dd** works more quickly with the raw versions of devices (that is, the ones which begin with **r**).
of	The name of the output file. Use the device name for the streaming tape device (**/dev/rmt0**).
bs	The input and output block sizes, in bytes.

The following **dd** command backs up 80 blocks from the file system on `/dev/hd`*n* to the streaming tape, `/dev/rmt0`, using a block size of **512** bytes:

```
dd if=/dev/rhdn of=/dev/rmt0 bs=512 count=80
```

To restore a file system backed up with **dd**, use the **dd** command again, copying the file system from the backup device to the minidisk.

Note: The **dd** command may be less efficient than the **backup** command if the file system being backed up does not contain many files. Also, if you restore a backup made with **dd** to a different minidisk, its size may change.

Managing the AIX Distributed Services Environment

Previous versions of the AIX operating system allowed you to mount and unmount only local minidisks. With the enhanced mount capabilities of AIX, however, you can create file trees that are independent of the file systems on individual minidisks. In addition, the Distributed Services licensed program allows you to use both local and remote directories and files to build these file trees. The following terms apply to AIX directory and file structures possible with the enhanced AIX mount capabilities.:

file system
The complete structure of directories and files contained on a single minidisk. To mount a file system, you mount the device that contains it (for example: `mount /dev/hd7 /mnt`).

virtual file system
The structure created by mounting a directory or file (as opposed to mounting a minidisk). Each directory or file mount creates a new virtual file system.

file tree
The complete directory and file structure of a particular node, starting at the root directory (/). A file tree is the product of all minidisk, directory, and file mounts that have been performed. The path name to any accessible directory or file is determined by the structure of the file tree.

These additional terms apply to a Distributed Services network:

node
An individual system connected to the network.

client
In an interaction between nodes, the node that requests resources.

server
In an interaction between nodes, the node that provides resources.

The following three figures represent the creation of a file tree (on node D174) from parts of different file systems; each file system resides on a different node in a Distributed Services network. First, each node mounts its file system that contains the **/u** directory. The figure shows a portion of this file system for each node:

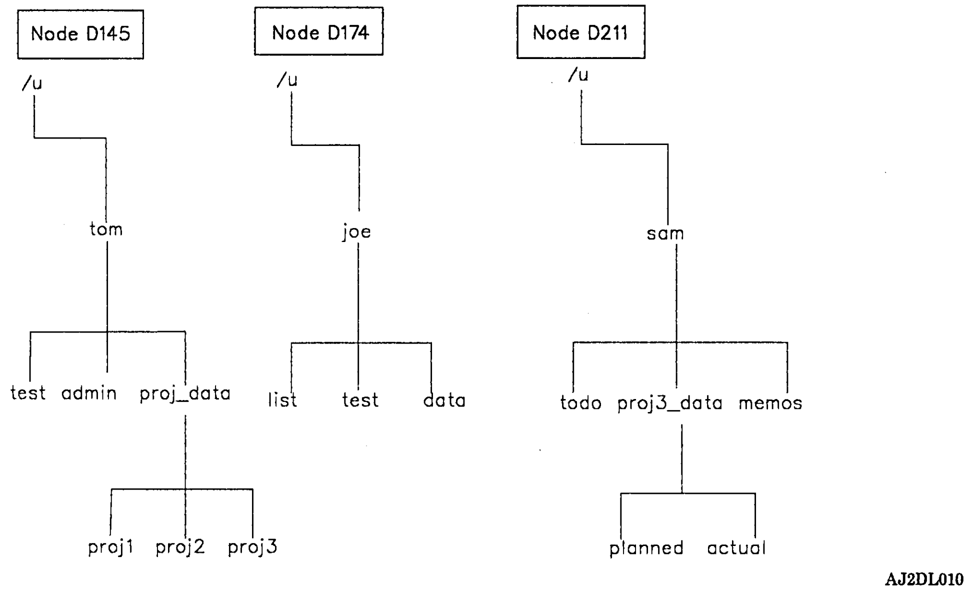


Figure 2-4. Creating a File Tree: File System Mounts

Next, node D174 mounts a directory from node D145 with the command:

```
mount -n D145 /u/tom/proj_data /u/joe/data
```

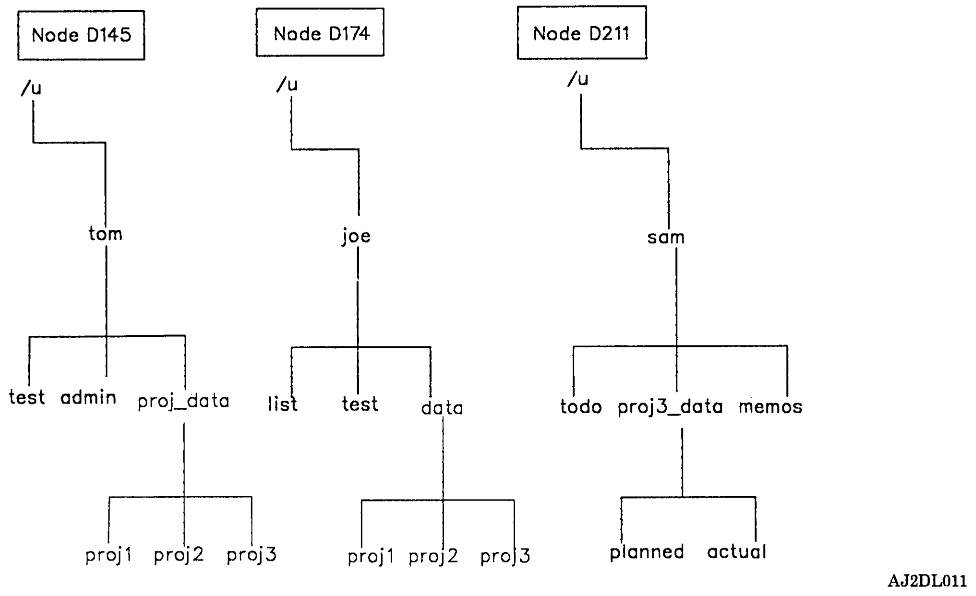


Figure 2-5. Creating a File Tree: Directory Mount

This mount creates a virtual file system (/u/joe/data), and the file tree on D174 now includes a remote directory structure. Generally, it does not matter to a user whether a particular directory or file is local or remote.

Finally, node D174 mounts a directory from node 211 with the command:

```
mount -n D211 /u/sam/proj3_data /u/joe/data/proj3
```

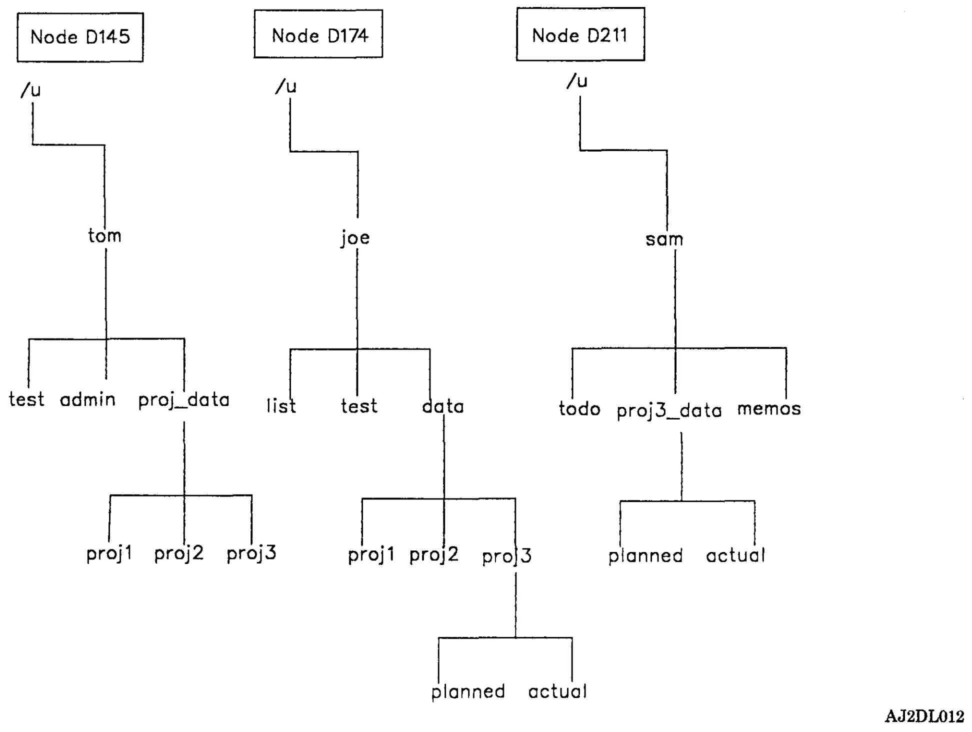


Figure 2-6. Creating a File Tree: The Completed Structure

The structure below the /u/joe directory now contains directories and files that physically reside on three different nodes. However, the file tree created by the directory mounts works just as it would if the same structure existed on a single (minidisk) file system.

Depending upon your requirements, your Distributed Services network can be simple or elaborate. If, for example, your network is small and its users perform their own mounts from the command line, your Distributed Services network will be relatively simple to configure and maintain. However, if you have a larger network and require a high degree of uniformity from node to node, your planning, configuration, and maintenance of Distributed Services will be more complex.

Certain AIX commands are enhanced to make them more useful in a distributed AIX environment. Other commands work differently in a distributed AIX environment than they do in an AIX environment without Distributed Services. For example:

- The **chown** (change owner) and **chgrp** (change group) commands, which change information recorded in i-nodes, can use either local or remote IDs.
- The **find** command expression, **-node** *nodename*, restricts the search to files and directories that exist on a specific node. Without **-node**, **find** searches a complete file tree.
- The **li** and **ls** commands, which list directory contents, can differentiate between local and remote items. (For more information about using **li** in a distributed AIX environment, see "Listing Directory Contents— The li (List) Command" on page 2-93.)
- The **-n** flag for the **mount** command lets you display the mount table of a remote node.

Other commands that either work differently in a distributed AIX environment, as well as those that are unique to Distributed Services, are discussed at the appropriate place in the remainder of the Distributed Services managing and customizing information as well as in *AIX Operating System Commands Reference.*

The following sections explain how Distributed Services works and also give guidance for managing a Distributed Services network. "Customizing Distributed Services" on page 3-35 explains the specific steps involved in creating the network.

Distributed Services File Systems

Before you read this section, you should be familiar with the information about file systems, minidisks, and mounts in "The File System—Background for System Management" on page 1-8 and "Creating and Mounting File Systems" on page 2-50, and with the terms defined in "Managing the AIX Distributed Services Environment" on page 2-73.

Note: Minidisks must be mounted locally before any of their contents can be mounted by remote nodes.

The **/etc/filesystems** file can contain stanzas for file and directory mounts just as it does for file system mounts. Also, a user with the appropriate authority can perform directory and file mounts from the command line.

Background for System Management

Each **mount** command creates a description of a virtual file system and adds the description to a master list of virtual file systems on that node. Each **umount** command removes a description from the list. The description contains information about the mounted object, the object that is mounted over, and the relationship of the virtual file system to other virtual file systems in the list.

A ***vnode*** describes a file or directory in a specific file system (since the same file or directory can be part of more than one file system). The vnode points to the i-node that describes the file on the minidisk that contains it.

The file system ***subtree*** is the structure of vnodes beneath each mounted directory. Each vnode contains a pointer to the virtual file system description. The file system of a particular system consists of all subtrees created by mounts on that system.

A ***file handle*** contains the file's device number, i-node number, i-node generation number, and type. The file handle, combined with the node ID of the system that contains the file, uniquely identifies the file on a specific system.

The following system maintenance commands can be run only on file systems (that is, on minidisks, not on virtual file systems or file trees):

- **backup -m**
- **restore -m**
- **fsck**

Mounting and Unmounting Files and Directories

When used to mount directories or files, the general format of the **mount** command is:

`mount` *pathname1 pathname2*

where *pathname1* is the path name of the object to be mounted and *pathname2* is the path name of the object to be mounted over.

The following flags to the **mount** command apply specifically to file and directory mounts:

-n *nodename* Specifies a remote system. If the object to be mounted (*pathname1*) is remote, you must use the **-n** flag in a command of the form:

`mount -n` *nodename pathname1 pathname2*

nodename is the name of the remote system that contains the object to be mounted. *pathname1* is the path name to the object on the remote system. *pathname2* is the path name to the object on the local system to be mounted over. If run without parameters, **mount -n** *nodename* displays the mount table at the remote node, *nodename* (the remote equivalent of running **mount** without flags locally).

-t *string* Specifies mounts of **type** *string*. **type** is an attribute that can be included in the **/etc/filesystems** stanza.

-i Causes a remote file tree structure to be ***inherited***. That is, the **-i** flag enables **mount** to replicate, on a local node, the structure of a remote file tree. The inherited structure includes all file and directory mounts below the mounted directory (*pathname1*). To replicate the structure, **mount -i** individually performs each directory or file mount contained in the file tree to be inherited.

The **umount** command unmounts files and directories as well as local minidisks. To unmount a file or directory, use a **umount** command of the form:

```
umount pathname2
```

where *pathname2* is the the file or directory over which another file or directory is mounted).

Note: You cannot unmount an object by naming the object itself (for example, *pathname1* in the previous **mount** command example).

To unmount only the remote mounts from a specific node, use the **-n** flag, for example:

```
umount -n nodename
```

To unmount all remote mounts, use the **allr** flag, for example:

```
umount allr
```

The **umount all** command unmounts all remote files and directories as well as all local minidisks.

Users can issue arbitrary file or directory mounts (**mount** *directory1 directory2*, **mount** *file1 file2*) or mounts described in **/etc/filesystems** if they:

- Have search permission to the directory or file to be mounted
- Own the directory or file to be mounted over
- Have write permission in the parent directory of the file or directory to be mounted over.

In addition, members of the system group can issue any mount described in the **/etc/filesystems** file. A user with superuser authority can issue any valid **mount** command.

Users can unmount any mount that they are allowed to issue. Members of the system group can unmount any mount described in **/etc/filesystems**.

The remainder of this section discusses ways to use the file system mount capabilities in a Distributed Services environment.

Mounting File Trees That Contain Mounts

Once a minidisk file systems are mounted, you can assemble objects from them into file trees with directory and file mounts. Ordinarily, when you mount a directory, you mount only some part of a file system, not a file tree. However, you can replicate file trees by using ***inherited mounts***. For example, if a directory has two directories mounted in its subtree, then:

- An ordinary **mount** command performs only one mount, that of named directory.
- An inherited mount performs three mounts: the named directory and then each of the two directories mounted in its subtree.

Thus, a single inherited mount command may result in numerous individual mounts.

There are two ways to perform inherited mounts:

- Issue the **mount -i** command.

- Place the **mount = inherit** attribute in the appropriate **/etc/filesystems** stanzas.

Inherited mounts are convenient when:

- Users at different nodes require access to the same data, and

- The physical location of the data may have to be changed regularly to balance system requirements and resources.

With inherited mounts, you can rearrange the location of data in one file system without requiring users at other nodes to update their **/etc/filesystems** file or automatic mount procedures.

The other nodes have a single **/etc/filesystems** stanza to describe the mount. However, inherited mounts are performed only when the mount command runs; that is, if the file tree is modified while the system is in use, the inherited mounts are not dynamically updated. Therefore, when it is necessary to change file trees that are inherited by other nodes, it is best to inform users that the changes will be made at specific times.

An inherited mount consists of multiple, individual directory or file mounts. To unmount an inherited mount, you must do one of the following:

- Unmount each directory or file individually, in the reverse order from that in which it was mounted.
- Use the **umount all** command.

You cannot unmount an inherited mount by naming the directory that was originally mounted, or by using the **umount allr** or **umount -n** commands. Generally, it is most convenient to design inherited mounts that:

- Are performed automatically at system start
- Do not require modification while users are logged into the system
- Can be unmounted with the **umount all** command as part of shutting down the system.

Performing Routine Mounts Automatically

The way in which remote mounts are managed depends upon the nature of the particular Distributed Services network. In some cases, it may be sufficient simply to allow users to perform mounts as needed; that is, there is no set network file system structure that must be in place in order for the network to be useful. In other cases, however, it is convenient to create a file system structure that automatically incorporates routine remote mounts. This section discusses techniques that can help you manage remote mounts.

If you routinely mount specific remote directories, it is convenient to mount them automatically. Because the nodes in a Distributed Services network may start at different times, relative to each other, remote mounts may fail when they are first issued. This section explains how to develop an automatic mount procedure that continues to issue the mounts until they succeed.

An automatic mount procedure involves the following AIX and Distributed Services components:

/etc/filesystems	The file containing file system descriptions (see "Information about File Systems—The /etc/filesystems File" on page 2-47).
type = *string*	An attribute that can appear in an **/etc/filesystems** stanza.
-t *string*	A **mount** argument that causes the **mount** command to mount all directories or files whose **filesystems** stanzas contain the attribute **type** = *string*). In addition, when you specify the **-t** flag with a file or directory mount, the **mount** command checks to see if that file or directory is already mounted. If it is, **mount** does not attempt to mount it again. This flag is necessary because the **mount** command treats duplicate mount requests differently depending on whether it is a file system mount or a file or directory mount. If a file system is already mounted, **mount** mount will not mount it again. However, **mount** does duplicate file or directory mounts unless you specify the **-t** *string* argument.
shell programs	User-written programs that issue remote mounts, monitor their status, and reissue or reschedule remote mounts, as needed.

Note: In the following examples, all user-written shells are assumed to:

- Belong to UID 0 and GID 0
- Reside in the **/etc** directory
- Have the permissions **754** (read, write, execute by owner; read, execute by group; read by others).

/etc/rc.ds	An initialization file run by **init** to configure Distributed Services at system start. (This is also a shell program.)

For a sample automatic mount shell program, see "Automatic Mount Procedures" on page 3-74.

Distributed Services ID Translation

The AIX Operating System uses a combination of identification numbers (IDs) and permissions to provide data security. When a user logs in, the **/etc/passwd** file associates a user ID (UID) with the login name. Similarly, the **/etc/group** file associates one or more group IDs (GIDs) with the login name. When a user starts a process (runs a command), AIX associates the user's UID and GID(s) with the process. AIX also associates a UID and GID with each file or directory. Unless they are explicitly changed, the UID and GID associated with a file are those of the user who created it.

AIX also associates three permissions (read, write, and search or execute) for three classes of users (owner, group, and other) with each file or directory. A file's i-node contains information about its permissions, UID, and GID. A process can use a file only if the appropriate correlation between IDs (UID and GID) and permissions exists. (For more information about file permissions, see "Protecting Files and Directories" in *Using the AIX Operating System.* For more information about i-nodes, see "I-nodes" on page 1-11.)

The same user name may correspond to different IDs on each node in a Distributed Services network. To satisfy the requirements of AIX permission checking, Distributed Services has the following facilities for associating the IDs on one node with those on another node:

inbound translation	Translates the IDs of incoming requests into their local equivalents.
outbound translation	Translates the IDs of outgoing requests into their network equivalents.

The following terms describe the different forms that IDs can have in a distributed AIX environment:

local ID	A numeric ID associated with a user or group name defined in the local **/etc/passwd** or **/etc/group** file. This ID is found in local i-nodes and is associated with locally running processes.

remote ID	A numeric ID associated with a user or group name defined in a remote **/etc/passwd** or **/etc/group** file. This ID is found in remote i-nodes and is associated with a process running at the remote node.
network ID	A numeric ID associated with a communication request; the ID that is passed between nodes on the network.

ID translation is the process of converting between a local ID or remote ID and a network ID. In respect to the local node, ID translation occurs in two directions:

forward translation
The translation of IDs associated with running processes. A local ID is transmitted to a remote node. The remote node translates the ID to determine whether the process has access to the remote node and to do AIX permission checking.

backward translation
The translation of IDs found in i-nodes. The local node translates a remote ID into its local equivalent to determine which local IDs own a remote file or directory.

At least two IDs (UID and GID) are associated with each process. Distributed Services translates IDs according to information in tables on each node. After a remote ID is translated to its local equivalent, AIX checks permissions by referring to the local **/etc/passwd** and **/etc/group** files. Thus, data security in a distributed AIX environment is comparable to that of an AIX system without Distributed Services.

Distributed Services loads profiles that contain the translation data into the AIX kernel. Appendix B, "Getting Started With Distributed Services Customization Commands" on page B-1 explains how to create and modify the profiles.

Inbound Translation

Each request from a client contains node ID (NID), UID, and GID information, and requires permission checking at the server. When a server receives a request, it uses the ID information of the request to select a row from the translation table. The row provides the equivalent local ID that AIX uses to perform the permission checking.

Note: The examples in this section demonstrate UID translation. Distributed Services translates GIDs in the same way.

Figure 2-7 on page 2-88 represents inbound translation. When the client node (NID 108133EB, nickname D94) places a request on the network, the local ID and network ID of the requesting process are the same (201). When the server node (NID 208131AA, nickname D87) receives the request, it translates the network ID into its local equivalent (301). To determine whether to give the process access to a file, the server node performs normal AIX permission checking; in this case, the file's owner (UID 301) has all three permissions for the file (-rwxr--r--).

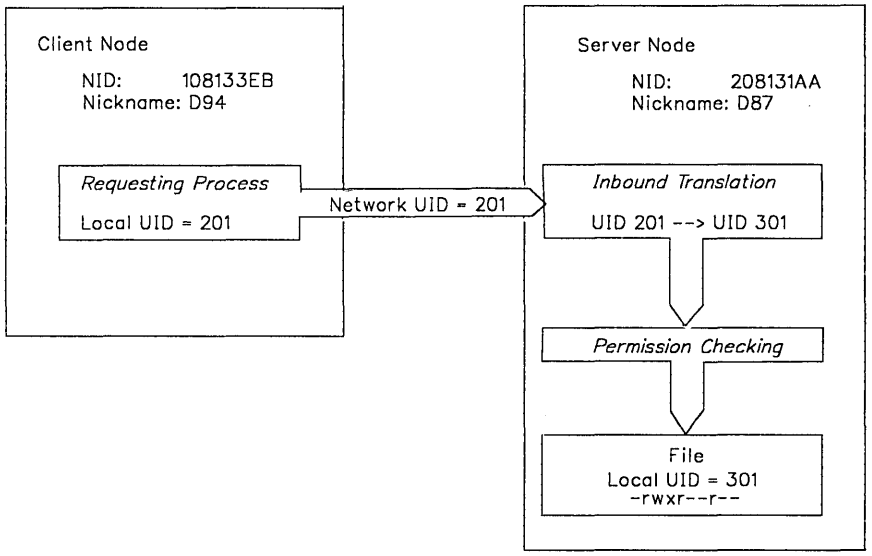


Figure 2-7. Inbound ID Translation

To perform the translation shown in Figure 2-7, the translation table on D87 contains the following data:

Usr/Grp Name	U/G	Local ID	Network ID Outbound	Network ID Inbound	Originating Node Name/Nickname
tom	U	301	201	201	D94

Node D87 translates the inbound network ID 201 into the local UID 301. The local UID 301 corresponds to the local user name tom.

In some cases, it is convenient to use wildcard entries in translation tables. An asterisk (*) in a field indicates that a specific ID is not required. In the following table, for example, a request with the inbound network ID 210 translates to the local

UID 208, regardless of which node the request comes from:

Usr/Grp		Local	Network ID		Originating Node
Name	U/G	ID	Outbound	Inbound	Name/Nickname
tom	U	301	201	201	D94
don	U	208	210	210	*

The following example shows a wildcard character in the NID and inbound network ID fields of the third line. Any incoming request that does not match either of the first two lines in the table is mapped to the local guest ID.

Usr/Grp		Local	Network ID		Originating Node
Name	U/G	ID	Outbound	Inbound	Name/Nickname
tom	U	301	201	201	D94
don	U	208	210	210	*
guest	U	200	200	*	*

If a line in the translation table explicitly matches the UID and NID of an incoming request (that is, there are no wildcards), Distributed Services always performs that translation. However, with wildcards, an incoming request can match more than one translation line. According to the following table, a request with the inbound UID 210 and the NID D94 matches three different local IDs (208, 212, and 200):

Usr/Grp		Local	Network ID		Originating Node
Name	U/G	ID	Outbound	Inbound	Name/Nickname
tom	U	301	201	201	D94
don	U	208	210	210	*
jean	U	212	212	*	D94
guest	U	200	200	*	*

In such cases, since an inbound network ID can be translated to only one local ID, Distributed Services selects one of the possible translations according the the following priorities:

Priority	Inbound UID	Inbound NID
1	*explicit*	*
2	*	*explicit*
3	*	*

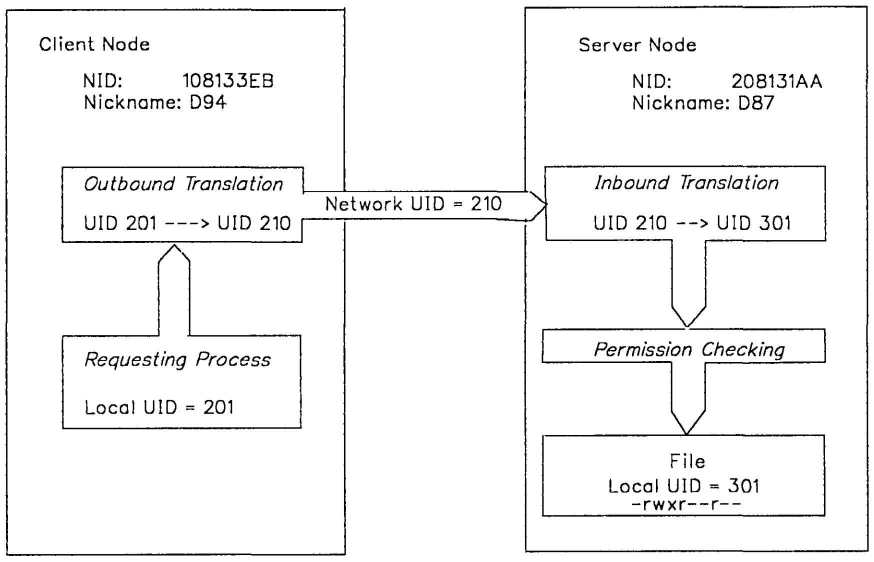
If no match occurs, Distributed Services rejects the request.

Note: Wildcards simplify ID translation by making it less restrictive. Consider the security requirements of your network when you decide how extensively to use ID translation wildcards.

Outbound and Inbound Translation

The principle of outbound translation is the same as that of inbound translation—one ID is translated into another according to data contained in a table. Outbound translation, however, occurs at the client node when the client places a request on the network. Outbound translation produces a network ID which can be different from both the local ID and the remote ID.

Figure 2-8 on page 2-91 represents inbound and outbound translation working together. Before placing a request on the network, node D94 translates the local ID of the process (201) to produce the network ID (210). Upon receiving the request from the network, node D87 performs inbound translation and permission checking as usual.



AJ2DL008

Figure 2-8. Outbound and Inbound ID Translation

When used together, outbound and inbound translation can simplify certain network management tasks. For example, the same user (tom) might have a different UID on each of five nodes in a network:

User Name	Node	UID
tom	Node_A	300
tom	Node_B	214
tom	Node_C	461
tom	Node_D	302
tom	Node_E	270

Without outbound translation, each of the five lines in the translation table at a given node must have a different value for the inbound network UID:

Usr/Grp		Local	Network ID		Originating Node
Name	U/G	ID	Outbound	Inbound	Name/Nickname
tom	U	300	-	300	Node_A
tom	U	300	-	214	Node_B
tom	U	300	-	461	Node_C
tom	U	300	-	302	Node_D
tom	U	300	-	270	Node_E

With outbound translation, however, each node can translate the local ID to a common network ID (a Social Security number, for example). While the table at each node still must have five rows for the user in the example, each row has the same value for the inbound network UID (that is, you have to gather less information to set up the translation tables):

Usr/Grp		Local	Network ID		Originating Node
Name	U/G	ID	Outbound	Inbound	Name/Nickname
tom	U	300	250	250	Node_A
tom	U	300	250	250	Node_B
tom	U	300	250	250	Node_C
tom	U	300	250	250	Node_D
tom	U	300	250	250	Node_E

You can further simplify the translation tables by using wildcards in conjunction with outbound translation. For example, the five lines required for the user in the previous example can be reduced to a single line if you use a wildcard for the NID:

Usr/Grp		Local	Network ID		Originating Node
Name	U/G	ID	Outbound	Inbound	Name/Nickname
tom	U	300	250	250	*

In practice, a network administrator should maintain a registry of user names and their corresponding network IDs. Then, when a user is added to a node on the network, the person who creates the account can refer to the registry for the common network IDs.

Listing Directory Contents— The li (List) Command

Use the **li** (list) command to list the contents of one or more directories. The general form for the **li** command is simply `li`, which lists the contents of your current directory.

You can do several other things with the **li** command as well. For example, you can list the contents of directories other than the current directory. You also can use the **li** command flags to get different types of information about directory contents. (For information about the **li** flags, see "li Command Flags" on page 2-94.)

To List Directory Contents

Enter: `li` *name*

where *name* is the name of a directory or file.

The Current Directory

To list the contents of your current directory, enter `li`:

```
$ li
book          examples      memo.schedule memo.vac
emdefs.p      mail.test     memo.space    translation
emkeys.o      memo.equip    memo.test
$ _
```

Used without flags, the **li** command simply lists the names of the files and directories in your current directory.

Other Directories

To list the contents of directories other than your current directory, use a command of the form `li` *pathname*. In the following example, the **pwd** command shows that the current directory is **/u/mark**, and the **li** command lists the contents of a different directory, **/u**:

```
$ pwd
/u/mark
$ li /u
a342374     dale        filesystems mark        profsvcs
billd       david       george      mel         roller
brown       ds          gorg        melanie     root
ctw         eacono      laurie      monroe      scheetz
dagitz      etc         lost+found  morley      tcpip
$ _
```

As this example shows, the **li** command ordinarily sorts directory and file names alphabetically.

Note: Your listing of the **/u** directory will not be exactly like this one. Generally, it will contain the names of login directories for users on your system, but may also contain other files or directories.

li Command Flags

In its simple form, the **li** command lists only the names of files and directories. However, several flags that you can use with **li** give you more information about the files and directories or change the way the listing displays. To use flags, enter a command of the form `li` *-flag name.*

The following table lists some of the most useful **li** command flags:

Flag	**Action**
-a	Lists **a**ll entries. Without this flag, the **li** command does not list the names of entries that begin with **.** (period), such as relative directory names, **.profile**, and **.login**.
-k	Provides information about files and directories that exist on remote systems (in a Distributed Services environment). A **-k** listing shows the type, permissions, remote system that contains the file, remote user ID and group ID, size, time of last modification, and name for each file or directory listed.
-l	Lists in **l**ong format. An **-l** listing shows the type, permissions, number of links, owner, group, size, time of last modification, and name for each file or directory listed.
-S*flag*	Controls how **li** sorts and displays the listing. The sorting options include: Last access time (**-Sa**) Last modification time (**-Sm**) Name (**-Sn**) Size (**-Ss**) Creation time (**u**). The **-r** flag reverses the sorting order (for example, **li -Snr** sorts by name in reverse alphabetical order).

Figure 2-9. li Command Flags

If you want to use more than one flag, type all of the flag names together in one string. For example, **li -alSnr** lists all entries in long form, sorting them in reverse alphabetical order.

The following example shows a long (**-l**) listing of a current directory:

```
$ li -l
Drwxr-xr-x   2 mark  staff   176  Apr 16 00:23 book
-rw-rw-rw-   1 mark  staff  7258  Feb 26 19:21 emdefs.p
-rw-rw-rw-   1 mark  staff  2224  Feb 26 19:21 emkeys.o
drwxr-xr-x   2 mark  staff    80  Apr 17 14:24 examples
-rw-r--r--   1 mark  staff    28  Feb 26 10:33 mail.test
-rw-r--r--   1 mark  staff   440  Apr 17 11:01 memo.equip
-rw-r--r--   1 mark  staff   220  Apr 17 11:01 memo.schedule
-rw-r--r--   1 mark  staff   220  Apr 17 11:01 memo.space
-rw-r--r--   1 mark  staff    28  Apr 17 13:11 memo.test
-rw-r--r--   1 mark  staff   110  Apr 17 11:01 memo.vac
-rw-r--r--   1 mark  staff  1873  Apr 15 14:24 translation
total 60 blocks
$ _
```

The **li -l** command returns the following information:

Field	Information
`-rw-rw-rw-`	File type and permissions set for each file or directory. The first character in this field indicates file type: - (hyphen) for ordinary files d for directories b for block special files c for character special files p for pipe (first in, first out) special files F for remote ordinary files D for remote directories B for remote block special files C for remote character special files P for remote pipe special files. Remaining characters indicate what read, write, and execute permissions are set for owner, group, and others. For more information on permissions, see "Changing Permissions—The chmod (Change Mode) Command" on page 3-45.
`2`	Number of links to each file. For an explanation of ***file links***, see "Linking Files—The ln (Link) command" on page 3-27.
`mark`	User name of the file's owner.
`staff`	Group to which the file belongs.
`7258`	Number of characters in the file.
`Feb 26 19:21`	Date and time the file was created or last modified.
`emdefs.p`	Name of the file or directory.
`total 60 blocks`	Number of 512-byte blocks taken up by files in this directory.

Figure 2-10. li -l Command Information

The following example shows a long listing of all files in the current directory, including files whose names begin with a period:

```
$ li -la
drwxr-xr-x  4 mark staff    272  Apr 17 13:10 .
drwxrwxr-x 26 mel  system   448  Apr  8 18:21 ..
-rw-rw-rw-  1 mark staff     33  Apr 17 14:28 .estate
-rw-rw-r--  1 mark staff    541  Apr 17 13:10 .profile
-rw-rw-r--  1 mark staff    531  Feb 24 12:02 .profile.bak
Drwxr-xr-x  2 mark staff    176  Apr 16 00:23 book
-rw-rw-rw-  1 mark staff   7258  Feb 26 19:21 emdefs.p
-rw-rw-rw-  1 mark staff   2224  Feb 26 19:21 emkeys.o
drwxr-xr-x  2 mark staff     80  Apr 17 14:28 examples
-rw-r--r--  1 mark staff     28  Feb 26 10:33 mail.test
-rw-r--r--  1 mark staff    440  Apr 17 11:01 memo.equip
-rw-r--r--  1 mark staff    220  Apr 17 11:01 memo.schedule
-rw-r--r--  1 mark staff    220  Apr 17 11:01 memo.space
-rw-r--r--  1 mark staff     28  Apr 17 13:11 memo.test
-rw-r--r--  1 mark staff    110  Apr 17 11:01 memo.vac
-rw-r--r--  1 mark staff   1873  Apr 15 14:24 translation
total 84 blocks
$ _
```

In the previous example, the D file type indicates that book is a remote directory (a directory that exists on a different system).

The following example shows a long listing of that directory:

```
$ li -l book
Frw-r--r--  1 mark   staff  7258  Apr 16 00:22 appendix
Frw-r--r--  1 mark   staff    28  Apr 16 00:22 chap.1
Frw-r--r--  1 mark   staff   440  Apr 16 00:22 chap.2
Frw-r--r--  1 mark   staff   110  Apr 16 00:22 chap.3
Frw-r--r--  1 mark   staff   220  Apr 16 00:22 chap.4
Frw-r--r--  1 mark   staff  1873  Apr 16 00:22 chap.5
Frw-r--r--  1 mark   staff   220  Apr 16 00:22 cover.ltr
Frw-r--r--  1 mark   staff  2224  Apr 16 00:22 glossary
Frw-r--r--  1 mark   staff    28  Apr 16 00:22 preface
total 52 blocks
$ _
```

The long listing of a remote directory resembles that of a local directory with two exceptions. First, the type field (F in this example) indicates that all items in the directory are remote. Second, the user and group names are translated into their local equivalents. For example, if:

- The local user name mark corresponds to the local UID (user identification number) 300,
- The local UID 300 corresponds to the remote UID 200, and
- The remote UID 200 corresponds to the remote user name jones,

the long listing shows mark as the file's owner; mark is the local equivalent of jones on the remote system.

The **-k** flag displays information about remote files and directories, including their remote UID and GID (group identification number); that is, the UID and GID that **li -k** displays are not translated into their local equivalents. The following example shows an **li -k** listing of the remote directory book:

```
$ li -k book
Frw-r--r--  dept998   200  1  7258  Apr 16 00:22 appendix
Frw-r--r--  dept998   200  1    28  Apr 16 00:22 chap.1
Frw-r--r--  dept998   200  1   440  Apr 16 00:22 chap.2
Frw-r--r--  dept998   200  1   110  Apr 16 00:22 chap.3
Frw-r--r--  dept998   200  1   220  Apr 16 00:22 chap.4
Frw-r--r--  dept998   200  1  1873  Apr 16 00:22 chap.5
Frw-r--r--  dept998   200  1   220  Apr 16 00:22 cover.ltr
Frw-r--r--  dept998   200  1  2224  Apr 16 00:22 glossary
Frw-r--r--  dept998   200  1    28  Apr 16 00:22 preface
total 52 blocks
$ _
```

Certain fields in an **li -k** listing are the same as they are in an **li -l** listing (type, permissions, size, date, and name). However, **li -k** displays the name of the remote system (`dept998`), the remote UID (200), and the remote GID (1), instead of the link count, local user name, and local group name that **li -l** displays.

There are other flags to the **li** command that you may find useful as you gain experience with the AIX system. All of the **li** command flags are explained under **li** in *AIX Operating System Commands Reference.* Another AIX command, **ls**, has features similar to those of **li**, and is also described in *AIX Operating System Commands Reference.*

Displaying Remote IDs: Special Conditions

Note: This section explains how **li** uses the **fullstat** system call to determine what remote file IDs to display. It is primarily intended to help you understand why, in some cases, **li** may display IDs other than those that you expect to see.

In the previous examples of how **li** lists remote IDs, each remote ID corresponds to a specific local ID and the translated ID of the file is the same as the local ID of the requesting process. To indicate this unambiguous condition, the **fullstat** system call (used by **li**) returns the translated ID and the CALLER tag.

Following is an example of the CALLER condition:

- Local translation entries:

```
Usr/Grp          Local        Network ID          Originating Node
 Name     U/G     ID      Outbound     Inbound     Name/Nickname
-----------------------------------------------------------------

 tom       U      201       2000         2000      *
```

- Remote translation entries:

```
Usr/Grp          Local        Network ID          Originating Node
 Name     U/G     ID      Outbound     Inbound     Name/Nickname
-----------------------------------------------------------------

 jones     U      205       2000         2000      *
 system    G        0          0            0      *
```

With these translation entries, if:

	UID	**GID**
Requesting process	201	1 (concurrent group 0)
Remote file	205	0

then:

	UID	**GID**
li -k returns	205	0
li -l returns	tom	staff

However, it is possible to set up translation tables that create the following special cases:

OTHER
The translated ID is the same as some local ID other than that of the requesting process. **fullstat** returns a translated ID and the OTHER tag. If the network ID translates to more than one local ID, **fullstat** returns the local ID that appears first in the translation table. For an example of conditions that cause **fullstat** to return the

OTHER tag, see “Example: The OTHER Condition” on page 2-102.

SOMEONE
The inbound ID is a wildcard and, therefore, can match any of the local IDs. **fullstat** returns the SOMEONE tag. **li** displays the value of **netsomeone** (a system parameter set in the **/etc/master** file), typically 65534. For an example of conditions that cause **fullstat** to return the SOMEONE tag, see “Example: The SOMEONE Condition” on page 2-103.

NOONE
The inbound ID matches no local ID. **fullstat** returns the NOONE tag. **li** displays a hyphen (−) to represent the value of **netnoone** (another system parameter that is set in the **/etc/master** file, typically to 65535). For an example of conditions that cause **fullstat** to return the NOONE tag, see “Example: The NOONE Condition” on page 2-104.

Example: The OTHER Condition:

- Local translation entries:

Usr/Grp Name	U/G	Local ID	Network ID Outbound	Network ID Inbound	Originating Node Name/Nickname
karin	U	201	2000	2000	*
tom	U	202	2001	2001	*
tj	U	203	2001		

- Remote translation entries:

Usr/Grp Name	U/G	Local ID	Network ID Outbound	Network ID Inbound	Originating Node Name/Nickname
karin	U	203	2000	2000	
jones	U	205	2001	2001	*
system	G	0	0	0	*

With these translation entries, if:

	UID	GID
Requesting process	201	1 (concurrent group 0)
Remote file	205	0

then:

	UID	GID
li -k returns	205	0
li -l returns	tom	system

Example: The SOMEONE Condition:

- Local translation entries:

```
Usr/Grp           Local          Network ID            Originating Node
 Name      U/G     ID       Outbound      Inbound      Name/Nickname
-------------------------------------------------------------------------

 karin      U      201         2000          2000      *
 tom        U      202         2001          2001      *
 tj         U      203         2001
```

- Remote translation entries:

```
Usr/Grp           Local          Network ID            Originating Node
 Name      U/G     ID       Outbound      Inbound      Name/Nickname
-------------------------------------------------------------------------

 jones      U      205         2001       *            *
 usr        G      100          100       *            *
```

With these translation entries, if:

	UID	GID
Requesting process	201	1 (concurrent group 0)
Remote file	205	0

then:

	UID	GID
li -k returns	205	0
li -l returns	65534	65534

Example: The NOONE Condition:

```
 Usr/Grp            Local          Network ID        Originating Node
  Name     U/G       ID       Outbound      Inbound   Name/Nickname

  karin     U        201        2000         2000     *
  tom       U        202        2001         2001     *
```

- Remote translation entries:

```
 Usr/Grp            Local          Network ID        Originating Node
  Name     U/G       ID       Outbound      Inbound   Name/Nickname

  watt      U        205        2002         2002     *
```

With these translation entries, if:

	UID	GID
Requesting process	201	1 (concurrent group 0)
Remote file	205	0

then:

	UID	GID
li -k returns	205	0
li -l returns	–	–

For more information about the **fullstat** system call (and the related **stat** system call), see *AIX Operating System Technical Reference.*

Chapter 3. Maintaining the AIX Operating System

CONTENTS

About This Chapter

This chapter covers the tasks required to maintain the AIX Operating System and to adapt the system to your needs. Anyone who manages an AIX system will perform some of the tasks described in this chapter (for example, maintaining the file system). Other tasks described here only concern you if you modify your system (for example, installing applications and local commands). A third set of tasks may become more important the longer you use the system (for example, refining your security policies).

Maintaining the File System

Each time a file is created, changed, or deleted, the operating system performs a series of file system updates. These updates, when written to a disk, produce a ***consistent*** file system. If all of the updates do not complete successfully, the result can be an inconsistent file system. Some inconsistencies may not be severe, but others, if not promptly corrected, can spread and eventually make an entire file system unusable. The **fsck** (file system consistency check) command analyzes a file system, locates inconsistencies, and often can correct certain inconsistencies automatically.

Causes of File System Damage

The major causes of file system damage are:

- A system halting (or failing) before it completes all pending file system updates, for example, due to power failure
- Poor operating procedures
- Failure of a disk drive or drive controller
- Physical damage to a diskette.

Generally, when a system halts before it completes its pending file system updates, damage to the file system is minor and easily corrected. File system damage that results from poor operating procedures or from fixed-disk or disk controller failure, however, can be severe enough to destroy an entire file system.

Disk Buffering—the sync Command

Any operation involving a file requires data to be retrieved from the fixed disk (read into memory), returned (written) to the disk, or both. These read and write operations also are known as disk input and output, or ***disk I/O***. To make some disk I/O more efficient, the operating system maintains a ***buffer*** (temporary storage area) in memory for recently accessed data blocks. The operating system writes data from the buffer to the disk only when:

- There is a **sync** system call.
- There is an **fsync** system call.
- The system needs the buffer for another purpose.

Disk buffering can increase the efficiency of I/O operations because it allows repeated reads and writes to the same block to occur without the block being physically read from the disk or written to the disk each time. The system is also able to schedule certain I/O operations (especially read operations) so that they occur in an order that is more efficient for the disk, rather than in the order in which they were requested. Together, disk buffering and disk scheduling help match the demand for I/O operations to the physical characteristics of the fixed-disk.

Although it may improve the efficiency of disk I/O, disk buffering can indirectly cause one type of file system damage—if the system stops before writing all data from the buffer back to the disk, the data on the disk is not current (and the data in memory is lost). Thus, it is extremely important for buffered data to be written to the disk (with the **sync** command) before the system is stopped. The **shutdown** command automatically runs the **sync** command. Failure to run **sync** (with **shutdown**) before stopping the system is a common cause of file system damage, but one that is very easy to avoid.

Even if the system fails, which usually means that you do not have time to run the **sync** command, any structural damage to the file system is usually easy to find and repair. The **fsck** command often can automatically repair this kind of damage. Thus, before you use the file system, you check it with **fsck** (either use the **Check a file**

system item on the AIX Operating System Installation/Maintenance diskette, or run **fsck** from the standalone shell, as described under "Running the Maintenance System" on page 2-6). Otherwise, if files are created on the damaged file system, the damage can become much worse.

The **dfsck** command lets you check two file systems on two different drives at the same time. Flags specified with **dfsck** are passed to **fsck**, letting you interact with two **fsck** programs. **dfsck** prints the file system name for each message that it sends. Thus, when you answer a message from **dfsck**, prefix the response with either 1 or 2 to indicate the first or second file system. For more information about **dfsck**, see "**fsck**" in *AIX Operating System Commands Reference.*

Note: The **fsck -p** command should be included in **/etc/rc** so that it always runs during normal startup. To check two file systems on two different drives at the same time during normal startup, include the **dfsck** command in **/etc/rc**.

Examples

Following are two examples of how a file system can become inconsistent. In each example, the system stops before all disk updates are complete.

- Example 1. Block Allocation Inconsistency

  When the operating system allocates a new block to a file, two things must be updated:

  - The i-node, to indicate that the file contains a new block
  - The free list, to indicate that a block that was free is no longer available.

  If the system stops between the first and second update operations, the i-node shows that the new block is allocated to the file, while the free list still shows that the block is available. There is an inconsistency in the file system.

The **fsck** command often can find and correct such inconsistencies if it is run on the file system immediately. However, if you begin to use the file system again without correcting the inconsistency, more serious problems can occur. For example, the block shown as both allocated (in the i-node) and free (in the free list) could be allocated to a second i-node; that is, the same block would be part of two different files or directories. File system damage of this type can spread quickly and be extremely difficult to correct; it can result in the appearance of strange information in files and the loss of some files or even an entire file system.

- Example 2. File Deletion Inconsistency

  When the operating system deletes a file, up to five things must be updated:

  - The directory entry that points to the file must be cleared.
  - The reference count in the i-node for the file must be reduced by one.
  - If the reference count goes to zero, all blocks in the file must be added to the free list.
  - If a large number of blocks are freed, a new link in the free chain must be written.
  - The i-node for the file must be made available again.

  Depending upon which of these update operations is completed at the time the system stops, different types of file system inconsistencies can occur.

Note: Lost data does not necessarily produce file system inconsistency. Generally, the data stored in ordinary files has nothing to do with the consistency of the file system. For example, if you begin a **write** operation and the system stops before the operation completes, the file on the disk simply does not get updated. While you have lost data, that loss may not affect the file system. The **fsck** command can only determine whether the structure of the file system is internally consistent; it cannot check the data within files.

Checking and Repairing File Systems—The fsck Command

A file system contains redundant information about its structure. If the file system is consistent, each instance of a particular piece of information is identical. The **fsck** command uses this redundant information to detect and, in many cases, correct file system inconsistencies.

The **fsck** program makes several passes through a file system, performing a different phase of checking in each pass. When it detects an inconsistency, **fsck** reports the error condition, displaying a message on the screen and waiting for a response. For an explanation of the **fsck** messages and what actions you should take to respond to them, see *Messages Reference.*

The file **/etc/rc** contains a list of the commands run each time the system starts. Generally, **/etc/rc** contains the **/etc/fsck -p** command. This line starts **fsck** before the system mounts any file systems. If the **/etc/filesystems** stanza for a file system contains either of the following lines, **fsck** checks the file system at this point:

```
check=true

check=group.
```

(**fsck** checks all file systems with the same *group* at the same time.)

Note: A ***stanza*** is a group of lines that define a part of the system. A stanza in **/etc/filesystems** defines a file system.

The **-p** (preen) flag causes **fsck** to run automatically, correcting all simple inconsistencies without requiring any action from the user. The **-p** flag enables **fsck** to repair most inconsistencies that result from the system stopping unexpectedly.

Without the **-p** flag, **fsck** runs interactively, displaying each error condition that it locates and waiting for you to decide whether corrections should be made. For information about other **fsck** flags, see "**fsck**" in *IBM RT PC AIX Operating System Commands Reference.*

It is very important to run **fsck** on unmounted file systems. The **fsck** command goes completely through the file system several times, often comparing data collected on one pass with data collected on another. Thus, any activity on the file system can interfere with the ability of **fsck** to check and repair file system damage.

The fsck Consistency Checks

Following is a list of the types of file system inconsistencies that **fsck** checks for, together with descriptions of how they can be corrected:

- Superblock Inconsistencies

  The superblock is the most frequently updated part of a file system and it is therefore the most likely to be inconsistent after the system stops unexpectedly. Every time a block or i-node is allocated or deallocated, the superblock should be updated.

  The **fsck** program checks all information about the file system in the superblock to make sure that it is consistent with the standard requirements for a file system. This information (including file system size, number of blocks allocated to i-nodes, cluster size, and interleave factors) never changes during normal operation. Should any of these values change, there is evidence that the file system is severely damaged. In such cases, **fsck** reports the condition and does nothing else with the file system. Usually the only way to recover such a file system is to restore it from a backup.

  If the information about the file system seems to be consistent and reasonable, **fsck** uses the file system and i-list size to validate all block numbers. All reasonable blocks are between the end of the i-list and the end of the file system. Any reference to a block not within that range is not valid and probably indicates severe file system damage.

- Free block list

  The free list is a series of logically connected blocks which are available (that is, they are not allocated to any file). The superblock contains the first block of the free list. The **fsck** command checks the structure of the free list and also checks the block numbers of every block in the list. **fsck** makes separate checks to make sure that no allocated blocks appear in the free list and that all blocks that are not allocated do appear in it. **fsck** can repair any inconsistencies found in the free list, usually by constructing a new free list that contains all blocks found not to be allocated.

- Free block count

  The superblock contains a count of the total number of free blocks within the file system. **fsck** compares this count to the number of blocks it found free within the file system. If the counts disagree, **fsck** can replace the count in the superblock with the actual free block count.

- Free i-node count

  The superblock contains a count of the total number of free i-nodes within the file system. The **fsck** command compares this count to the number of i-nodes it found free within the file system. If these counts disagree, **fsck** can replace the count in the superblock with the actual free i-node count.

• I-node Inconsistencies

An individual i-node is not likely to be inconsistent as is the superblock. There is, however, a danger that the i-nodes for files that are in use when the system stops will be inconsistent.

The **fsck** command checks the list of i-nodes in sequence, starting with i-node 1 and continuing to the last i-node in the file system. **fsck** checks each i-node for inconsistencies in format and type, link count, duplicate blocks, bad blocks, and i-node size.

- I-node format and type inconsistencies

  Each i-node contains a ***mode word***—a description of the type and state of the i-node. There are five i-node types:

  - Regular
  - Directory
  - Block special
  - Character special
  - First-in-first-out (FIFO, also called a named pipe).

  An i-node whose type is not one of these five does not have a valid type.

  There are three i-node states:

  Allocated
  Unallocated
  Neither allocated nor unallocated.

  An i-node that is neither allocated nor unallocated is not correctly formatted. The format of an i-node can be damaged when incorrect data are written to the i-node list (for example, as a result of a system failure).

- I-node link count inconsistencies

  Each i-node contains a count of the total number of directory entries that refer to the i-node (the ***reference count***). **fsck** verifies the reference count in each i-node by scanning the entire directory structure, beginning with the root directory, and counting the actual number of links to that i-node.

  If the stored link count is not zero and the actual link count is zero, the file is inaccessible. If the inaccessible file contains any data, **fsck** links the file into the **lost+found** directory on the file system. If the file contains no data, **fsck** deallocates the i-node for that file.

If the actual link count is neither zero nor equal to the stored link count, it is likely that a directory entry was added or removed without the i-node being updated. **fsck** can replace the stored link count with the correct value.

– Inconsistencies from duplicate and invalid blocks

  Each i-node contains a list, or pointers to lists (indirect blocks), of all the blocks claimed by that i-node. **fsck** first determines whether each block number in the list is valid and then compares the block number against a list of allocated blocks. If it finds inconsistencies, **fsck** displays the names of all files that contain invalid or duplicate blocks.

  Inconsistencies of this type usually occur when you create or extend files on a file system that has duplicate blocks in its free list. If the duplicate block contains data in both files, the damage is not severe. If the file system uses the duplicate block for a directory or an indirect block, however, severe inconsistencies in the file system structure may occur. Although such inconsistencies are serious, you can prevent them by routinely checking every file system with **fsck** before you mount it. (For example, do not use the **fsck -f** command in **/etc/rc**. The **-f** [fast] flag saves time by not checking file systems that were cleanly unmounted the last time they were used. The extra precaution of running **fsck -p** from **/etc/rc** reduces the chance that your system will mount a damaged file system.)

  Generally, **fsck** corrects such inconsistencies by deleting the files with invalid or duplicate blocks, but not automatically—that is, you must respond to a prompt from **fsck** before it will destroy the files. Before allowing **fsck** to destroy such damaged files, you should examine the files to determine whether you can save any important data.

- Size inconsistencies

  In each i-node is a ***size field*** that indicates the number of bytes in the file associated with the i-node. **fsck** can determine whether the number of bytes is consistent with the number of blocks allocated to the file. In the case of directories, **fsck** can also make sure that the size is a multiple of 16 (the length of a directory entry).

  If it finds size inconsistencies, **fsck** displays a warning message. Size errors are not serious, but **fsck** does not have sufficient information to correct the size. You can correct the problem by copying the data from the old file into a new file and then deleting the old file.

- Indirect Block Inconsistencies

  There are three types of indirect blocks: ***single-indirect***, ***double-indirect***, and ***triple- indirect***. A single-indirect block contains a list of some of the block numbers associated with an i-node. Each entry in a single-indirect block is a data block number. A double-indirect block contains a list of single-indirect block numbers. A triple-indirect block contains a list of double-indirect block numbers.

  Indirect blocks are associated with a particular i-node. Thus, inconsistencies in indirect blocks directly affect the associated i-node. **fsck** can check for inconsistencies such as blocks that are already claimed by another i-node and block numbers that are outside the range of the file system. **fsck** finds and corrects indirect block inconsistencies just as it does inconsistencies between the i-node and data blocks.

- Data Block Inconsistencies

  There are two types of data blocks: plain and directory. Plain data blocks contain the information stored in a file. Because an ordinary file may contain any type of data, **fsck** cannot check the validity of the contents of a plain data block.

  Directory data blocks contain directory entries. Each directory data block can be checked for inconsistencies involving directory i-node numbers pointing to unallocated i-nodes, directory i-node numbers greater than the number of i-nodes in the file system, incorrect directory i-node numbers for the current and parent directories (. and .., respectively), and directories disconnected from the file system.

If a directory entry i-node number points to an unallocated i-node, then **fsck** tries to remove that directory entry. This condition might occur if the data block containing the directory entry were modified and written to the disk, but the system stopped before the i-node could be updated.

If a directory entry i-node number points beyond the end of the i-node list, **fsck** tries to remove that directory entry. This condition occurs if incorrect data are written into a directory data block.

The directory i-node number entry for . should be the first entry in the directory data block. Its value should be equal to the i-node number for the directory itself.

The directory i-node number entry for .. should be the second entry in the directory data block. Its value should be equal to the i-node number for the parent directory (or the i-node number of the directory itself if the directory is the root directory).

The **fsck** command checks the internal connections of the file system in general. If it finds a file or directory not linked into the file system, **fsck** links the file or directory to the **lost+found** directory. This condition can be caused by i-nodes being written to disk and the system stopping before the corresponding directory data blocks are updated. Items linked to **lost+found** are identified by their i-node number (for example, if **fsck** finds a file whose i-node number is `1234`, it links the file to `/lost+found/1234`).

Repairing File Systems by Destroying Files

Under some conditions, the only way **fsck** can restore consistency to a file system is to clear i-nodes—that is, by destroying files. For example, if a block is allocated to two files, **fsck** clears both files. This section explains how to keep your loss of possibly useful data to a minimum.

Before letting **fsck** destroy a file, be sure you know which file is to be destroyed and what is wrong with the file. Most files damaged when a system stops unexpectedly are temporary files and they usually are of little value. However, if it is necessary to destroy a permanent file, you should try to salvage its contents and, if the file belongs to another system user, notify that person that the file is to be destroyed. Before **fsck** destroys a file, it lists all the names linked to that file, and prompts you to determine whether each link to the file can be removed. Only when all links are removed can **fsck** destroy the file.

Often, the name of a file indicates whether the file's contents are important. For example, temporary files may have names like `ctm5a.P5024` and often are in special directories like **/tmp** or **/usr/tmp**. Like temporary files, editor backup files (files with a `.bak` extension) and object modules (files with a .o extension) usually are not important. However, it is likely that a file named `boss/employees/raises` is quite important, and you should make every effort to salvage its contents. If you cannot tell from its name whether a file is important, try to salvage its contents.

If you know the i-number of a file, you can find the name of the file with the **ncheck** program. For example, if **fsck** indicates that two files, i-numbers 408 and 1212, on the **/u** file system with have a duplicate block, you could find the names of those files by entering:

```
ncheck -i 408 1212 /u
```

For more information about the **ncheck** command, see **ncheck** in *AIX Operating System Commands Reference.*

When a file contains one or two bad or duplicate blocks, you probably can salvage most of its contents by copying them into a new file before destroying the original (damaged) file. To copy the contents of the file, mount the file system temporarily (and **readonly**, if possible) and use the **cp** command. It is safest to copy the file onto another (consistent) file system. If you want to copy the file within its original file system, first run **fsck** to make sure the free list is consistent.

You can destroy a damaged file in several different ways. If a file contains invalid or duplicate blocks, **fsck** removes all links to the file and then destroys the file by deallocating its i-node (assuming that you respond *y* to the **fsck** prompts). You can destroy a file directly with the **clri** (clear i-node) command. However, it is safer to let **fsck** destroy the file, since **fsck** automatically restores the file's data blocks to the free list; **clri** does not do this.

You should not use the **rm** (remove file) command to destroy files for two reasons:

- You must mount the damaged file system to run **rm**. Mounting the damaged file system can compound the damage.
- **rm** frees all bad and duplicate blocks. Thus, if you removed the damaged file with **rm**, the same type of damage is likely to occur the next time the system allocates those blocks.

Any time you destroy files, run **fsck** on the file system before using it again. **fsck** eliminates inconsistent directory references and makes sure that the free list is correct.

The Input/Output System

An I/O (input/output) system is a mechanism for transferring data among system devices (for example, when you print a file, the I/O system transfers the data from the file to the printer). The AIX I/O system is independent of the particular devices that produce output or receive input. If three devices (for example, a printer, a fixed-disk, and a display station) can all take input in the same form, the I/O system makes no distinction among them. To understand how the I/O system works, you should first have a general understanding of device drivers and special files.

Device Drivers and Special Files

A ***device driver*** is a program that makes the logical connection between the system and a device. The kernel contains a device driver for each type of device on the system. For example, if your system has two identical printers, you need only one device driver to run both of them; if you have two different types of printers, your system needs a separate device driver for each printer type.

Each device has a ***class***, a ***major device number***, and a ***minor device number***. There are two device classes, ***block*** and ***character***, and each class is associated with a group of device drivers. (For descriptions of the device classes, see "Block I/O System" on page 3-18 and "Character I/O System" on page 3-18.) The major device number indicates which driver the device uses. The minor device number passes additional information to the driver (for example, to specify which of several identical printers should be used).

Device characteristics are recorded in ***special files*** (ordinarily located in the **/dev** directory). A process can have read and write access to a special file just as it does to an ordinary file. However, the read and write operations on special files produce device I/O. To perform I/O operations, a process simply has to access the appropriate special file; it does not have to deal with the characteristics of the device itself.

Block I/O System

A model block I/O device consists of randomly addressed memory blocks of a uniform size: 2048 bytes. The blocks are addressed 0, 1, . . . up to the size of the device (because the addresses start at 0, the highest-numbered address is the device size less one). A block device driver creates this model on a physical device.

Input and output for block devices is through a group of buffers. The system maintains a buffer for each block device. When a device issues a read request, the system searches the buffers for the requested block. If the block is in the buffer, the system makes it available to the device without any physical I/O. If the block is not in the buffer, the system performs a physical I/O operation, replacing the least recently used block in the buffer with the requested block. The system handles block write operations similarly.

Generally, block (buffered) I/O increases system efficiency. First, block I/O can greatly reduce the number of read and write operations required to perform I/O. Second, block I/O allows the operating system to schedule I/O for the most efficient system operation. In some instances, however, block I/O is a disadvantage. If the system stops unexpectedly, there may be physically incomplete I/O—changes made to the buffers that were not yet written to the disk.

Note: Not all file I/O is done with the block I/O system. Files can be ***mapped*** directly into memory. A single segment of a mapped file can be shared by more than one program. Changes made to mapped files show up in the file immediately.

Character I/O System

The character I/O system includes all devices that do not fit the block I/O model. Unlike block I/O requests, character I/O requests go directly to the device driver.

A model character device places characters directly onto a queue until the queue contains the maximum number of characters. As soon as there are any characters in the queue, the I/O operation starts, removing the characters from the queue one at a time and sending them to the device. When the number of characters falls to a specific level, the process passing characters to the queue refills the queue to its maximum capacity.

However, many devices in the character class are more accurately described as ***raw*** devices. For example, I/O for both fixed-disks and streaming tapes occurs in units of bytes which have no relationship to characters. Raw devices are included in the character class only because they are not block I/O devices.

Using the Queueing System

A ***queue*** is an ordered list of jobs waiting to be done by the computer. The main job of the AIX queueing system is to manage printer use, especially on systems that have more than one printer. However, you also can use the queueing system to control access to other system resources, as is explained under "Friendly and Unfriendly Backends" on page 3-28. You can adapt the queueing system to your requirements easily. For example, by changing one word in a master configuration file, you can set a printer to take jobs in the order in which they are submitted or according to their sizes. Because your requirements of the queueing system may change periodically, you should understand how the system works and how to modify it—the subjects of this section.

Parts of the Queueing System

The queueing system has four parts:

The print program
The **print** program accepts requests for print jobs and places them in the queue directory, **/usr/lpd/qdir**. (For more information on the **print** command, see *Using the AIX Operating System.*)

The qdaemon program
A ***daemon*** is a process that starts when you start your computer and runs until you shut your computer down. The **qdaemon** is a daemon that keeps track of print job requests (or other output) and the printers (or other devices) available to handle them. The **qdaemon** process maintains queues of outstanding requests and sends them to the proper device at the proper time. The **qdaemon** also records printer usage data for system accounting purposes.

The backend program
A ***backend*** program sends output to a particular device (a printer, for example). You do not run a backend program directly; rather, **qdaemon** runs a backend program,

sending it the names of the files to be printed and any print flags that you specify. The backend returns this and other information to the **qdaemon** through a status file in the **/usr/lpd/stat** directory. You can use the **print -q** command to interpret and display this status information, such as how many pages were printed, what the status of the printer is, and how much of the print job is finished.

The configuration file

The ***configuration file***, **/etc/qconfig**, describes the configuration of available queues and devices. Both the **print** command and the **qdaemon** process refer to the configuration file.

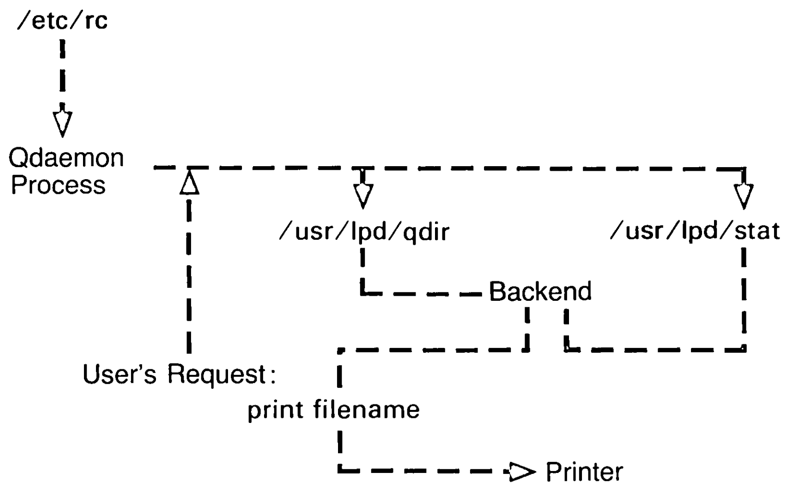


Figure 3-1. How the Queueing System Works

Queues and Devices

A queue is simply an ordered list of requests for a particular service. A ***device*** is something that can handle those requests one at a time. Each queue must be handled by at least one device, but it often may be handled by more than one.

Configuration

Often you do not have a choice about which device services which queue. For example, if your system has only one printer, then that printer must service the print queue. There are times, however, when you should consider carefully the relationship between queues and devices. For example, if your system has two identical printers located next to each other, you probably will want both printers to service a single printer queue. You could accomplish this by placing the following stanzas in the configuration file **/etc/qconfig**:

```
lpr:
        argname = -l
        device = lpdev0, lpdev1

lpdev0:
        file = /dev/lp0
        backend = /usr/lpd/lp

lpdev1:
        file = /dev/lp1
        backend = /usr/lpd/lp
```

The first stanza describes the **lpr** queue. The `argname` line sets the flag used to request this queue, `-l`. A print command requesting this queue has the form `print -l` *filename*. The `device` line specifies which devices service the queue. The second and third stanzas describe the printers that service the queue. In each case, `file` is the special file associated with the printer, and `backend` is the file that contains the printer backend program. Once a print request reaches the top of the queue, the backend sends the file to the next available printer.

In another situation, you might want to define a different queue for each printer. For example, if the two identical printers are in different buildings, or if there is a difference in speed or quality between the two printers, you (or those who use your system) should be able to select a particular printer. In this case, the stanzas in **/etc/qconfig** could be:

```
lpa:
        argname = -la
        device = lp0

lp0:
        file = /dev/lp0
        backend = /usr/lpd/lp

lpb:
        argname = -lb
        device = lp1

lp1:
        file = /dev/lp1
        backend = /usr/lpd/lp
```

In this example, there are two queue stanzas, `lpa` and `lpb`. Following each queue stanza is a stanza describing the printer that services that queue.

Queue and Device Names

The names of queues and devices are similar, but there are important differences in how they work. Most system users do not use device names, since a **print** command is actually a request for a queue, not for a device.

Two lines in the **/etc/qconfig** file relate to the naming of queues:

- ***queue name***. This is the name of the stanza that describes the queue (like `lpa` and `lpb` in the previous example). The queueing system refers to a queue by its queue name and uses the queue name in any messages about the print job (including responses to the **print -q** command).

- ***argname***. This is the name you use with the **print** command to request a specific queue (like `-la` and `-lb` in the previous example).

Although the `queue name` and `argname` can be the same, you may find it helpful to make them different so that, as in the previous example, the `queue name` can look like the name of a printer and `argname` can look like a command flag.

When more than one printer services a queue, you can request a particular printer by naming it explicitly. (Ordinarily, your request would be serviced by the first device on that queue that became available.) To name a device explicitly, use the `argname` of the queue serviced by that device, followed by a colon (:) and the `device number` of the printer. (Devices are numbered independently for each queue, starting at 0; device numbers refer to devices in the order in which they appear in **/etc/qconfig**.) The following stanza (taken from an earlier example) defines a queue service by two printers:

```
lpr:
        argname = -l
        device = lpdev0, lpdev1
```

The command **print -l:0** requests the printer `lpdev0`; the command **print -l:1** requests the printer `lpdev1`.

All references to devices by the **print** command and **qdaemon** are based upon the names contained in the **/etc/qconfig** file. Thus, within the guidelines given in "Changing the Configuration File" on page 3-31, you can choose your own names for queues and devices.

Status Control

By setting the status of queues or printers to `up` or `down`, you can control access to them. A status setting of `up` or `down` refers to how the system treats the queue or device, not to the physical state of any system component.

A queue is available to both the **print** command and the **qdaemon** if its stanza in **/etc/qconfig** contains the line:

```
up = TRUE
```

or if the stanza does not contain the line up = (since the default value for up is TRUE). A queue is not available if its stanza contains the line:

```
up = FALSE
```

A queue that is up works normally. However, if a queue is down:

- The **qdaemon** command does not send jobs to devices on the queue.
- The **print** command does not accept requests for the queue.
- The **print -q** command shows that all devices on that queue are OFF.

The only way to set the status of a queue on or off is to change the **/etc/qconfig** file. Generally, you should not set the status of a queue to off unless you know that printers that service the queue cannot be used for several days (for example, if they are to be taken down for repair).

You can set the status of a device more easily, using the **print -dd** (device down) or **print -du** (device up) command. To set status down for the first device servicing queue 1, use the command:

```
print -1:0 -dd
```

To set status up for the same device, use the command:

```
print -1:0 -du
```

In both examples, the device name -1:0 consists of the queue name (-1), a colon (:), and the device number (0).

The **qdaemon** does not send jobs to down devices, but **print** accepts requests for their queues. It also is possible for a backend to set the status of a device to off. Once a device is down, either because of a **print** command or the action of a backend, you must use the **print** command to set its status back to up.

The **qdaemon** records device status in a status file in the directory **/usr/lpd/stat**. When you start the system, **qdaemon** checks this status file before trying to run any device. Thus, you do not have to reset the status of devices each time you stop and restart your system.

Job Order

The ***discipline*** line in the **/etc/qconfig** file determines the order in which printers service the requests in a queue. In a queue stanza, the line:

```
discipline = fcfs
```

sets the order to *first come first served.* If there is no `discipline` line in the queue stanza, requests are serviced in first-come-first served (`fcfs`) order. The line:

```
discipline = sjn
```

in a queue stanza sets the order to *shortest job next.*

Each print job also has a ***priority number*** that can be changed with the **print** command flag **-pr**=*n*. Print jobs with higher priority numbers (*n*) are handled before requests with lower priority numbers. Any system user can alter the priority of a print request with a command such as:

```
print -pr=20
```

The default priority number is 15. The maximum priority number for ordinary users is 20. The maximum priority number for privileged users (su and members of the system group) is 30.

Accounting

Accounting information consists of the user number, user name, and number of pages printed for each request. The backend program determines what constitutes a page. You can cause the queueing system to produce accounting records by placing the following line in the appropriate queue stanza in **/etc/qconfig**:

```
acctfile = /usr/adm/qacct
```

If a queue stanza does not contain an `acctfile` line or contains the line:

```
acctfile = FALSE
```

the queueing system does not produce accounting records for that queue.

The **qdaemon**, which produces the accounting records, does not create the accounting file. Thus, you can substitute another file name for **/usr/adm/qacct** if you choose.

You also can use a separate file to record the accounting data for each queue. Using separate accounting files can help you collect usage statistics for each queue (a useful ability if, for example, you are looking for ways to improve the efficiency of your queueing system) or allow you to assess different charges to the users of your system for the use of different printers.

For a full discussion of AIX facilities for accounting and monitoring system activity, see Chapter 5, "Running System Accounting" on page 5-1 and Chapter 6, "Using the System Activity Package" on page 6-1.

Backends

The `backend` line of the stanza for a device in the **/etc/qconfig** file determines what backend the **qdaemon** runs for that device. A line like:

```
backend = /usr/lpd/lp
```

specifies the name of the backend program and includes any flags that are to be passed to that program. A line such as:

```
file = /dev/lp
```

causes **qdaemon** to send the backend's standard output to this file (in this case, the special file associated with a printer).

The line:

```
access = both
```

gives the backend both `read` and `write` access to the file. The line:

```
access = write
```

or the absence of an `access` line in the stanza gives the backend only `write` access to the file. If `file` does not have a value, **qdaemon** ignores the value of `access`.

The default value for `file` is `FALSE` and is only used by backends that access devices without help from **qdaemon**.

Friendly and Unfriendly Backends

Ordinarily, the actions of the queueing system and a backend (as described so far) are neatly integrated. As long as a backend program follows certain conventions for communicating with **qdaemon** and the **print** command, it is called a ***friendly backend***. The queueing system also can handle ***unfriendly backends***—backends that do not follow these communication conventions. An unfriendly backend can be virtually any program; **qdaemon** requires no special communication or coordination with an unfriendly backend. The `friend` line in a queue stanza:

```
friend = TRUE
```

or

```
friend = FALSE
```

indicates whether a backend is friendly or unfriendly. If there is not a `friend` line in the stanza, the default value is `TRUE`.

You can use the queueing system to provide organized access not only to system devices (often printers), but to other system resources as well. These applications of the queueing system typically use unfriendly backends. For example, on a system that has a log file (**/etc/logfile**) available to all users, only one user at a time should be allowed to edit the file. By adding the following stanzas to **/etc/qconfig**, you can require all access of the file to be through the queueing system (that is, one user at a time):

```
log:
        argname = -lf
        device = logdev
        friend = FALSE

logdev:
        file = /etc/logfile
        backend = /bin/cat
```

To add a paragraph to the logfile, you now must first write the paragraph into a file (named, for example, `temp`) and then use the **print** command:

```
print -lf temp
```

When the queueing system services your request for access to **/etc/logfile**, the **print** command adds the contents of `temp` to **/etc/logfile**. The (unfriendly) backend in this example is a standard command (**cat**). The **qdaemon** writes the contents of the temporary file to the end of **/etc/logfile**, not the beginning. Any system users who do not have `write` permission to **/etc/logfile** can change the file only by going through the queueing system.

You also can use the queueing system to require queued access to a program. For example, suppose a program on your system (`/bin/bigjob`) requires so much processing that, if two people run it at once, your entire system slows down seriously. You can make the program accessible only through the queueing system by adding the following stanzas to **/etc/qconfig**:

```
hog:
        argname = -b
        friend = FALSE
        device = bigjob

bigjob
        backend = /bin/bigjob
```

To run `bigjob`, enter `print -b` *name* (*name* is the name associated with the queued request and is used when you cancel the job, change its priority, or display its status, and is not necessarily the name of a file). The queueing system services requests for the `bigjob` program one at a time. Queued access is not appropriate for interactive programs.

Burst Pages

The term ***burst pages*** refers to printer output formatted so that each sheet of paper contains one page of output (on continuous form paper, the pages can be *burst* apart at the perforations). With the appropriate instructions in **/etc/qconfig**, most friendly backends print burst pages on their devices. For example, the following lines usually appear in the device stanza for a line printer:

```
header = always
trailer = group
feed = 3
```

The first line tells the backend to print a header page that gives the name of the job, who requested it, its date, and other information before every file printed. Other possibilities for the `header` line are:

```
header = group
```

which requests a header before every group of files for a single user, and:

```
header = never
```

(or the absence of a `header` line) which eliminates headers completely.

The `trailer` line in the device stanza tells the backend to print a trailer page, which gives the name of the user, only after a group of files is printed. (You also can use the `group` and `never` values with trailers.)

The `feed` line specifies that three `feed` (blank) pages are to be printed whenever the printer becomes idle. The feed operation pushes paper out of the printer so that it is easier to tear off. If the feed line in the stanza is:

```
feed = never
```

the backend is not invoked when the machine becomes idle, and therefore feed pages are not printed. The result of `feed = never` is usually the same as the result of:

```
feed = 0
```

except that the 0 value invokes the backend, but tells it not to feed any pages. For some printers, `feed = never` and `feed = 0` produce different results. For example, some line printers make a loud noise if the paper fold rests under the spinning drum. The backend for such a printer might interpret `feed = 0` to mean that three blank lines should be fed, moving the fold off of the printer drum.

The `align` value is another way to control the number of extra pages between print jobs. The following line in the device stanza tells the backend to print an aligning form feed before any job requested after the printer is idle:

```
align = TRUE
```

This value applies only to printers using continuous form paper. As a rule, alignment on such printers is unnecessary, since backends maintain proper paper alignment. However, on some continuous form printers, it is possible for users to get the paper out of alignment when they remove their jobs. If this is a problem on your system, the alignment line will help. The absence of an `align` line is the equivalent of the line:

```
align = FALSE
```

Changing the Configuration File

Both **print** and **qdaemon** read **/etc/qconfig** when they start. **qdaemon** starts when you start the system; **print** starts each time someone requests a print job. Thus, if you change **/etc/qconfig**, **print** will read the new version of the configuration file the next time it runs, while **qdaemon** continues to rely on the original version of **/etc/qconfig**. To avoid problems due to this inconsistency, give the command:

```
print -rr
```

which causes **qdaemon** to reread **/etc/qconfig** and reinitializes itself, based on the new version of **/etc/qconfig**, after all of its running jobs complete. (You also can cause **qdaemon** to reread the configuration file by shutting down and restarting the system, but the **print -rr** command usually is more convenient.)

You should be alert to two common problems associated with installing a new **/etc/qconfig**:

- If you install a new queue, someone may request that queue before you reinitialize **qdaemon**.
- If you remove an old queue, someone may request that queue at the last minute and find that it is no longer valid.

In either case, **qdaemon** logs an error message. You must determine whether the message refers to a new queue (in which case there is no longer a problem once you reinitialize **qdaemon**) or to an old queue (in which case the problem will exist until you remove the obsolete queue entries from **/usr/lpd/qdir**).

Keeping the qdaemon Running

Under normal circumstances, **qdaemon** starts when the system starts, runs until the system shuts down, and should require no attention from you. Under unusual circumstances, however, the **qdaemon** may stop running or be unable to perform its function. This section explains what you need to do under these conditions.

Any of the following conditions indicates that the **qdaemon** needs maintenance:

- **print** requests return the error message:

  ```
  cannot awaken qdaemon (request accepted anyway)
  ```

- **qdaemon** detects serious inconsistencies within itself and displays an error message

- **ps -ef** (the process status command that gives a full listing of all processes) does not show a process named **/etc/qdaemon** or **qdaemon**.

To restart the **qdaemon**, use the following command:

```
/etc/qdaemon
```

Generally, only privileged users can use this command. The new **qdaemon** goes through an initialization process.

If the **qdaemon** does not stay running, make sure that both **qdaemon** and **print** have the appropriate permissions. The user root owns both **qdaemon** and **print**. However, **qdaemon** and **print** must run as if they are owned by the user who starts them. The permission **s** sets the effective owner (user ID) of a process to that of the command that is running it. Thus, the appropriate permissions for these two commands are:

qdaemon -r-sr-xr--

To check these permissions, enter `ls -l /etc/qdaemon`.

If the permissions need to be reset, enter: `chmod 4454 /etc/qdaemon`. (You must have superuser authority to reset these permissions.)

print -r-sr-xr-x

To check these permissions, enter `ls -l /bin/print`.

If the permissions need to be reset, enter: `chmod 4455 /bin/print`. (You must have superuser privileges to reset these permissions.)

If you continue to have problems with **qdaemon**, you can reinitialize the entire queueing system by going through the following procedure:

1. If the **qdaemon** is running (use the **ps -ef** command to find out), end it with the **kill** command (**kill** *process id number*).

2. If any backends are running, use the **kill** command to stop them.

3. Delete the contents of the following directories:

   - **/usr/lpd/stat**

   - **/usr/lpd/qdir**

   - **/tmp/copies**

4. Restart the **qdaemon** with the command `/etc/qdaemon`.

Customizing Distributed Services

Before You Begin

For information on adding the RT PC to a communications network, see "Part 3. Communications Planning" in the *Planning Guide.*

Hardware Requirements

Install a communications adapter (see *Options Installation*):

- Multiprotocol Adapter **OR**
- Baseband Adapter.

Software Requirements

- AIX Operating System licensed program, Version 2.1.1 (see *Installing and Customizing the AIX Operating System*)
  - VRM Program
  - Base System Program
  - VRM Device Drivers
    - Multiprotocol Adapter Device Driver **OR**
    - VRM Baseband Adapter Device Driver
  - SNA Services.
- Distributed Services licensed program, Version 1.1.

In addition, you need to customize the local AIX Operating System:

- Run the **devices** command to add the Multiprotocol Adapter Device Driver or VRM Baseband Adapter Device Driver (see "Customizing System Devices" in *Installing and Customizing the AIX Operating System*):
  - Install the Multiprotocol Adapter as the **mpd0** device.
  - Install the Baseband Adapter as the **net0** device.

- Run the **minidisks** command to add local file systems (see "Customizing System Minidisks" in *Installing and Customizing the AIX Operating System*).
- Run the **users** command to add local users (see "Managing User Accounts" on page 2-16).

Information Requirements

Finally, you need to understand the basic principle of AIX system management, particularly the following topics:

- "The File System—Background for System Management" on page 1-8 and "The Base AIX File System" on page 1-15
- "Managing User Accounts" on page 2-16
- "Information about File Systems—The /etc/filesystems File" on page 2-47
- "Backing up Files and File Systems" on page 2-58

You should also be familiar with the Distributed Services information provided in the following sections:

- "Managing the AIX Distributed Services Environment" on page 2-73
- "Distributed Services File Systems" on page 2-78
- "Distributed Services ID Translation" on page 2-85
- Appendix B, "Getting Started With Distributed Services Customization Commands" on page B-1.

Configuration Topics

The discussion of customizing a Distributed Services network includes the following topics:

- "Working with Distributed Services Profiles" on page 3-38
- "Setting Up a Test Network" on page 3-51
- An overview of extended network planning:

  "Managing ID Translation" on page 3-58
  "Building Distributed File Trees" on page 3-62
  "Managing the Single System Image Environment" on page 3-70.
- A discussion of advanced customization procedures:

  "Automatic Mount Procedures" on page 3-74
  "Customizing a Server Node" on page 3-89
  "Customizing a Client Node" on page 3-92.
- A description of two configuration scenarios:

  "Scenario Two: A Text Processing Environment" on page 3-107
  "Scenario One: A Program Development Environment" on page 3-93.

In addition, customization forms are provided for your use in Appendix C, "Distributed Services Customization Forms" on page C-1.

Working with Distributed Services Profiles

Distributed Services keeps all network information in structured files called ***profiles***, which are data base files controlled by data management routines. These files are stored, by default, in the **/etc/profsvcs/pfslocal** directory.

Creating and Modifying Distributed Services Profiles

The following commands provide an interface to these profiles through which you can customize your network:

ndtable	Provides a window interface to the Distributed Network Node Table. This table contains an entry for each remote node that is known to the local node. Each entry identifies the remote node's nickname (optional), node ID (NID), node security, data link type, connection profile name, and attachment profile name. This information is stored in the Distributed Services SNA profiles.
ugtable	Provides a window interface to the Network Users/Groups Table. This table correlates a list of local user and group IDs with incoming and outgoing network IDs. The AIX Operating System uses this information to control access to the node and to do AIX permission checking. This information is stored in the Distributed Services UID/GID Translation profiles.
ipctable	Provides a window interface to the Distributed IPC Queues Table. Each entry that describes a local queue contains a queue name and an IPC key. Each entry that describes a remote queue correlates a local queue name and IPC key with a remote IPC key and node ID or nickname. This information is stored in the Distributed Services IPC Key profiles. See *AIX Operating System Programming Tools and Interfaces* for more information on using distributed message queues.

dsldxprof Allows you to modify the UID/GID Translation profiles with information stored in an ASCII file. See "Using the dsldxprof Command" on page B-19 for more information.

You can access these programs directly from the AIX Operating System command line; you can also access **ndtable**, **ugtable**, and **ipctable** through the Usability Services Interface. See Appendix B, "Getting Started With Distributed Services Customization Commands" on page B-1 for detailed discussion of these commands.

Backing Up Distributed Services Profiles

You should periodically make a backup copy of the Distributed Services profiles to protect against loss of these profiles.

To Back Up Distributed Services Profiles

1. Change directories to the directory in which the profiles reside:

   ```
   # cd /etc/profsvcs/pfslocal
   ```

2. Enter the following **backup** command:

   ```
   # ls -a | backup -iv
   ```

More Detailed Information: In addition to the normal periodic backup procedure, back up the profiles under any of the following conditions:

- Before and after you create a new set of profiles.
- Before and after you update or change a current set of profiles.
- Before and after you change the security level of SNA Services.
- Before and after you change the communications password (to protect against forgetting the password).

Enter the following commands on the AIX Operating System command line to back up the Distributed Services profiles in the standard directory:

```
# cd /etc/profsvcs/pfslocal
```

```
# ls -a ¦ backup -iv
```

To back up profiles in a different directory, use the **cd** command to change to that directory before using the **backup** command. This procedure copies all of the profiles in the selected directory to the diskette in **/dev/fd0**. If you use another method of backing up the files, make sure that the method you choose backs up the hidden files (file names beginning with a . [period]) as well as the normal files.

Restoring a Backup Copy of Distributed Services Profiles

To Restore Distributed Services Profiles

1. Shut down Distributed Services:

   ```
   # shutdown -d
   ```

2. Stop SNA Services:

   ```
   # stop sna
   ```

3. Change to the directory in which the Distributed Services profiles reside:

   ```
   # cd /etc/profsvcs/pfslocal
   ```

4. Insert the backup diskette in diskette drive 1 and enter the following **restore** command:

   ```
   # restore -xv
   ```

5. Restart SNA Services and Distributed Services:

   ```
   # rc.ds
   ```

More Detailed Information

1. Shut down Distributed Services:

   ```
   # shutdown -d
   ```

   When you specify the **-d** flag, **shutdown** issues the following commands (after sending a warning message to logged-on users and waiting 60 seconds):

   ```
   /etc/dsstate -sb
   /etc/umount allr
   /etc/dsstate -ab
   ```

   The two **dsstate** commands block all incoming and outgoing network activity. The **umount** command unmounts all remote directories.

   Note: If you have mounted local files or directories over remote directories, you must unmount the local object before the **umount** command can unmount all remote directories.

   Shutting down Distributed Services stops network traffic in and out of the node, but it does not shut down SNA Services.

   If you are the only user currently logged on, or if there is no current remote activity, you can use the "fast" version of **shutdown**:

   ```
   # shutdown -df
   ```

   When you specify both the **-d** and **-f** flags, **shutdown** does not send a warning message to other logged-on users and does not wait 60 seconds before issuing the **dsstate** commands or the **umount** command. Alternatively, you can specify some period other than 60 seconds (instead of specifying **-f**), for example:

   ```
   # shutdown -d 15
   ```

   This sends the warning message and waits 15 seconds before proceeding.

2. Stop SNA Services:

   ```
   # stop sna
   ```

   This step is necessary because AIX will not copy new versions of the SNA profiles over the files currently in use. If you are not restoring profiles into the **/etc/profsvcs/pfslocal** directory, you do not need to shut down Distributed Services and stop SNA Services. (See “Creating Alternate Sets of Profiles” on page 3-47 for more information about alternate profiles.)

3. Change to the directory that will contain the restored profiles. For example, to restore Distributed Services profiles that were backed up to diskette using the procedure described in “Backing Up Distributed Services Profiles” on page 3-39:

   ```
   # cd /etc/profsvcs/pfslocal
   ```

4. Restore the files:

   ```
   # restore -xv
   ```

5. Restart SNA Services and Distributed Services, if necessary, by running the **rc.ds** initialization file:

   ```
   # rc.ds
   ```

Loading Distributed Services Profiles

In order for AIX to use the UID/GID Translation profiles or the IPC Key profiles, they must be loaded into the running kernel. There are two ways to do this:

1. Run the **ugtable**, **dsldxprof**, or **ipctable** commands to create or modify local tables.

   Whenever a member of the **system** group or a user with superuser authority ends any of these three commands, it loads the corresponding table into the kernel.

2. Run the **dsxlate** or **dsipc** command.

The **dsxlate** command loads the Network Users/Groups Table into the kernel; the **dsipc** command loads the Distributed IPC Queues Table.

Normally, you need to run these commands under two circumstances:

a. At system start up.

   Distributed Services provides a shell initialization file, **/etc/rc.ds**, that is run by the **init** process at system startup. The default copy of this file includes lines that run both **dsxlate** and **dsipc**.

b. After remote configuration.

   When you use **ugtable**, **dsldxprof**, or **ipctable** to create or modify remote tables, the command does not load the corresponding table when you exit. Someone must run **dsxlate** or **dsipc** at the remote node to load the tables into the remote kernel.

The /etc/rc.ds File

In addition to running **dsxlate** and **dsipc** at system startup, the **/etc/rc.ds** file runs several other commands that initialize the Distributed Services environment. Figure 3-2 on page 3-44 shows the default **/etc/rc.ds** file.

After it loads the Distributed Services profiles, **/etc/rc.ds** starts SNA Services:

```
start sna
```

and starts a Baseband Adapter default attachment:

```
start /attachment/EDEFAULT&
```

If your network uses an SDLC link, change this line to the following:

```
start /attachment/SDEFAULT&
```

```
# @(#)rc.ds     5.4 87/04/13 09:29:36

echo Translate tables are being loaded ...
dsxlate

echo IPC keys are being loaded ...
dsipc

echo Starting SNA ...
start sna

echo Starting attachment ...
start /attachment/EDEFAULT&

# Dsstate specifies starting 100 kernel processes here.  The command will
# actually start only the maximum number of processes available to
# Distributed Services, which in the default case is 20 processes.
# The number 100 is used here to ensure that the maximum number of kernel
# processes (up to 100) will automatically be started without requiring the
# user to edit this file.
echo Starting kernel processes ...
dsstate -p100 -k -ce -se -sa -aa
```

Figure 3-2. Default /etc/rc.ds File

If your RT PC is part of both a Baseband Adapter network and an SDLC network, **rc.ds** should start both default attachments.

Finally, **/etc/rc.ds** runs the **dsstate** command to set the Distributed Services kernel logic. See the **dsstate** command in *AIX Operating System Commands Reference* for a discussion of all **dsstate** flags.

Recovering Distributed Services Profiles

Infrequently, you may receive an error message telling you to recover your profiles. When you receive this message, or any time that you suspect that your profiles may be damaged, run the **/etc/profsvcs/profck** command as follows:

```
# /etc/profsvcs/profck  -ds  /etc/profsvcs/pfslocal
```

This command tries to rebuild the profiles in the directory you specify. If **profck** cannot rebuild the profiles, restore a backup copy of the profiles (see "Restoring a Backup Copy of Distributed Services Profiles" on page 3-40).

You can also use the **recover** command to rebuild individual database files as follows:

```
# /etc/profsvcs/recover  /etc/profsvcs/pfslocal/pfsnid
# /etc/profsvcs/recover  /etc/profsvcs/pfslocal/pfsuidgid
# /etc/profsvcs/recover  /etc/profsvcs/pfslocal/pfsipcky
```

In this example, the first command tries to recover the Distributed Network Node Table. The second tries to recover the Network Users/Groups Table. The third tries to recover the Distributed IPC Queues Table.

See the discussions of the **profck** and **recover** commands in *SNA Services Guide and Reference* for more detailed information about these commands.

Working with Remote Profiles

Note: To change the profiles on a remote node:

- The remote node must be defined in the local Distributed Network Node Table, and the local node must be defined in the remote Distributed Network Node Table.

- You must be a member of the local **system** group or be running with local superuser authority. You must also have **system** group or superuser access to the remote node.

To change the profiles on a remote node, enter the NID or nickname of that node in the pop-up panel that appears when you run **ndtable**, **ugtable**, or **ipctable**:

```
Enter Node ID Nickname        ».............
(Default is the local id.)
```

This causes the command to mount the remote **/etc/profsvcs/pfslocal** directory and allows you to modify the profiles in that remote directory. You can access and modify these profiles just as you do local profiles with the following differences:

- If you are not a member of the **system** group or do not have superuser authority, you cannot browse remote tables.

- The Network Users/Groups Table does not display user or group names. To add a translation entry, you must specify a numeric ID.

- **ugtable** and **ipctable** do not load the Network Users/Groups Table or Distributed IPC Queues Table into a remote kernel when you end the command.

  Note that any changes you make to the Network Users/Groups Table or Distributed IPC Queues Table do not take effect until the tables are loaded into the kernel. To load the tables, do one of the following:

  – Have a user at the remote node shut down and restart Distributed Services.

  – Run the **dsxlate** or **dsipc** command at the remote node.

    If you have installed TCP/IP and have a remote login ID with superuser or system group authority, you may be able to use the **tn** command to log onto the remote node or the **rexec** command to run **dsxlate** remotely. See *Interface*

Program for use with TCP/IP for more information on running these commands.

Creating Alternate Sets of Profiles

The default Distributed Services profiles are stored in the **/etc/profsvcs/pfslocal** directory. However, you can create additional sets of profiles for Distributed Services to use, and store those sets in other directories in the directory **/etc/profsvcs**.

Creating an Alternate Set of Local Profiles

1. Create a new directory in the **/etc/profsvcs** directory to contain the new set of profiles.

2. Copy all files (* and .*) from **/etc/profsvcs/pfslocal** into the new directory.

3. Use the Distributed Services customization programs to change the tables as required. Be sure to specify the new directory name instead of accepting the default when the system asks for the `Node ID/Nickname`.

4. Copy the new profiles to **/etc/profsvcs/pfslocal** when you want the system to use them.

More Detailed Information

1. Create a new directory in the directory **/etc/profsvcs** to contain the new set of profiles (the Distributed Services customization commands expect **/etc/profsvcs** to be the parent directory.) For example, to create a new directory called `newprofs`, use the following commands:

   ```
   # cd /etc/profsvcs
   # mkdir newprofs
   ```

2. Copy all files from **/etc/profsvcs/pfslocal** into the new directory. Note that each profile database has an index file (with a corresponding file name beginning with a . [period]).

If you use the AIX **cp** command, be sure to specify both the database files and the "hidden" index files:

```
# cd /etc/profsvcs/pfslocal
# cp *  .*  /etc/profsvcs/newprofs
```

If you have installed Data Management Services, you can use its **copy** command. This command copies the index files along with the database files you specify:

```
# cd /etc/profsvcs/pfslocal
# copy * /etc/profsvcs/newprofs
```

3. Use the Distributed Services customization programs to change the profiles as required. When you run **ndtable**, **ugtable**, or **ipctable**, a pop-up panel appears:

```
Enter Node ID Nickname       ».............
(Default is the local id.)
```

   Instead of selecting the default value, enter the name of the new directory that contains the alternate set of profiles and then press **Do**. For example, enter the directory name `newprofs` to select the new set of profiles created in the previous examples. The full path name of the directory is not required because the specified directory must be in **/etc/profsvcs**. You can then change the values in the profiles in the new directory.

4. Copy the new profiles to **/etc/profsvcs/pfslocal** when you want the system to use them. (Follow the same procedures listed in "Restoring a Backup Copy of Distributed Services Profiles" on page 3-40 for stopping Distributed Services and SNA Services.)

You might also want to create alternate sets of profiles to be used on another node:

Creating an Alternate Set of Remote Profiles

1. Create a new directory in the **/etc/profsvcs** directory to contain the new set of profiles:

   ```
   # mkdir /etc/profsvcs/remoteprofs
   ```

2. Mount the remote **/etc/profsvcs/pfslocal** directory:

   ```
   # mount -n node /etc/profsvcs/pfslocal /mnt
   ```

3. Copy all files (* and .*) from **/mnt** into the new directory:

   ```
   # cd /mnt
   # cp .* * /etc/profsvcs/pfslocal/remoteprofs
   ```

4. Unmount the remote **/etc/profsvcs/pfslocal** directory:

   ```
   # umount  /mnt
   ```

5. Use the Distributed Services customization programs to change the tables as required. Be sure to specify the new directory name instead of accepting the default when the system asks for the `Node ID/Nickname`.

6. Copy the new profiles to the remote **/etc/profsvcs/pfslocal** directory when you want the remote system to use them.

More Detailed Information

1. Create a new directory in the **/etc/profsvcs** directory to contain the new set of profiles:

   ```
   # mkdir /etc/profsvcs/remoteprofs
   ```

When selecting a name for the new directory in **/etc/profsvcs**, do not use the following:

- The node ID of a node in the network
- The nickname of a node in the network.

The customization commands check the name you give in step 5 to see if it is a valid NID or node nickname before they check to see if it is the name of a directory in **/etc/profsvcs**. If it is a valid NID or nickname, they mount the remote **/etc/profsvcs/pfslocal** and access that, rather than accessing the local copy you have made in Step 3.

2. Mount the remote **/etc/profsvcs/pfslocal** directory in a temporary location:

   ```
   # mount -n node /etc/profsvcs/pfslocal /mnt
   ```

3. Copy all files (* and .*) from the remote directory (**/mnt**) into the new directory:

   ```
   # cd /mnt
   # cp .* * /etc/profsvcs/pfslocal/remoteprofs
   ```

4. Unmount the remote **/etc/profsvcs/pfslocal** directory:

   ```
   # umount  /mnt
   ```

5. Use the Distributed Services customization programs to change the tables as required. Be sure to specify the new directory name instead of accepting the default when the system asks for the `Node ID/Nickname`.

6. Copy the new profiles to the remote **/etc/profsvcs/pfslocal** directory when you want the remote system to use them. (Follow the same procedures listed in "Restoring a Backup Copy of Distributed Services Profiles" on page 3-40 for stopping Distributed Services and SNA Services and in "Loading Distributed Services Profiles" on page 3-42 for reloading remote profiles.)

Setting Up a Test Network

This section describes how to set up a limited Distributed Services test network. Since a limited network requires fewer customization decisions (and less set up), you can use it to test your mount capabilities and become familiar with the basic principles of ID translation before you plan your fully customized network.

A working Distributed Services network requires that:

1. Each node in the network be identified to SNA Services.

2. At least one network user or group be identified to the AIX Operating System running at each server node.

To accomplish step 1, build a Distributed Network Node Table at each node. Each entry in this table identifies a node to SNA Services.

The minimum required to accomplish step 2 depends on the type of access you want users to have:

- To enable all network users to have read access to a node, you need to add only one translation entry to the Network Users/Groups Table on a server node. This entry should associate all incoming IDs (*) with a local ID and can be either a user or group translation entry. As long as the Network Users/Groups Table translates one of the IDs associated with a remote user, he can issue mounts and can list directories and read files that have read permission set for `others`.

- To enable all network users to have write access to a node, you need to have at least two translation entries, one that associates all incoming UIDs with a local UID and one that associates all incoming GIDs with a local GID.

Testing Remote Mounts

To enable two nodes to mount each other's files or directories, complete the following steps:

To Test Remote Mounts Between Two Nodes

1. Query the node ID at each node:

```
# uname -m
```

2. At each node, add an entry to the Distributed Network Node Table that describes the other node.

3. Test the link:

```
# linktest NID
```

4. At each node, add one global translation entry (one that has a wildcard inbound ID) to the Network Users/Groups Table.

5. Issue a test mount:

```
# cd /
# mount -n NID /u  /mnt
```

More Detailed Information

1. Query the node ID by running the following command at each node:

```
# uname -m
```

2. Assume that the node IDs of two nodes on a Local Area Network are 20810CBF and 108133EB. To identify these nodes to each other, add the following entry to the Distributed Network Node Table at node 108133EB:

```
Remote Nickname             »............

Remote Node ID              »20810CBF

Node Security               »None
                            »Secure

Data Link Type              »Ethernet
                            »SDLC

Attachment Profile          ».......
If Left blank, Remote Node ID will be used
```

and add the following entry to the Distributed Network Node Table at node 20810CBF:

```
Remote Nickname             »............

Remote Node ID              »108133EB

Node Security               »None
                            »Secure

Data Link Type              »Ethernet
                            »SDLC

Attachment Profile          ».......
If Left blank, Remote Node ID will be used
```

3. Test the SNA Services link between the two nodes by running the **linktest** command at one of the nodes, for example:

   ```
   # linktest 20810CBF
   ```

   If the link test succeeds, proceed to the next step. If it fails, see *SNA Services Guide and Reference.*

4. Add the following entry to the Network Users/Groups Table at each node:

   ```
   User/Group Entry                    »User  »Group

   User/Group Local ID                 »100...
   (opt. if U/G name entered exists)

   User/Group Name                     »........
   (Ignored if U/G Id entered)

   Outbound Network ID                 »100........
   Inbound Network ID                  »*..........

   Originating Node ID/Nickname        »*............
   ```

   This translation entry associates all incoming UIDs (from any node defined in the Distributed Network Node Table) with local UID 100.

   These entries allow users from either node to mount a remote file or directory, assuming that they meet the ordinary **mount** requirements (see "Mounting and Unmounting Files and Directories" on page 2-79). They can also list remote directories and read or copy remote files that belong to remote user 100 or which have read permission granted to `others`.

5. Issue a remote **mount** command, for example:

   ```
   # mount -n 20810CBF /u  /mnt
   ```

   Be sure that you enter alphabetic characters in the node ID as uppercase characters.

   If the mount fails, see "Distributed Services Tuning and Problem Determination" on page 3-139.

Allowing Remote Write Access

To give network users write access to remote files on your test network, do the following:

To Allow Write Access to Remote Users

1. Add both a global UID translation entry and a global GID translation entry to the Network Users/Groups Table.

2. Issue a test mount command:

   ```
   # mount -n 20810CBF /usr/guest  /usr/guest
   ```

3. Create a file:

   ```
   # cd /usr/guest
   # echo "This is a test" > testfile
   ```

More Detailed Information

1. In order for a user to create a remote file, both his user ID and one of his group IDs must translate to a valid remote user ID and group ID because the AIX Operating System must place a local UID and a local GID in a newly created i-node.

   Once a user has write access to a remote node, regular AIX permission checking determines which remote directories or files he can modify. To give write access to the users in the two-node example discussed under "Testing Remote Mounts" on page 3-52, add the following global GID translation entry to the Network Users/Groups Table at each node:

```
User/Group Entry                    »User  »Group

User/Group Local ID                 »100...
(opt. if U/G name entered exists)

User/Group Name                     »........
(Ignored if U/G Id entered)

Outbound Network ID                 »100........
Inbound Network ID                  »*..........

Originating Node ID/Nickname        »*.............
```

2. Issue a **mount** command, for example (from node 108133EB):

   ```
   # mount -n 20810CBF  /usr/guest  /usr/guest
   ```

   The **/usr/guest** directory is the home directory of user **100 (guest)**. User **guest** also belongs to group **100 (usr)**.

3. Create a file:

   ```
   # cd /usr/guest
   # echo "This is a test" > testfile
   ```

   Once you have created this file, use the **li** command to see which user and which group it belongs to:

   ```
   # li -l
   ```

   Regardless of your local user or group ID, this file should belong to **guest** and **usr**:

   ```
   Frw-r--r--    1 guest    usr           15  May 26 15:54 testfile
   ```

Enabling Remote Configuration

To be able to customize a remote node, follow these steps before doing any other customization:

To Allow Remote Configuration from One Node

1. Select one node to be the administrative node.
2. At the administrative node, add Distributed Network Node Table entries describing each node in the network.
3. At each nonadministrative node, add a Distributed Network Node Table entry describing the administrative node.
4. At each nonadministrative node, add an entry to the Network Users/Groups Table that translates group 0 from the administrative node to local group 0

Planning the Network

Managing ID Translation

The Distributed Services ID translation facilities make possible five different types of individual translation entries. You can associate:

1. One inbound network ID (from one node) with one local ID.
2. One inbound network ID (from all nodes) with one local ID.
3. All inbound network IDs (from one node) with one local ID.
4. All inbound network IDs (from all nodes) with one local ID.
5. One local ID with one outbound network ID.

In addition, by having several instances of entry types 1, 3, or 5, each naming a different node, you can have a total of eight possible translation relationships, as illustrated in the following table:

Local IDs	Outbound IDs	Inbound IDs	Node IDs
One	One	One	One
One	One	One	Several
One	One	One	All (*)
One	One	All	One
One	One	All (*)	Several
One	One	All (*)	All (*)
One	One		
Several	One		

Figure 3-3. ID Translation: Possible Combinations

Among all these possibilities, how do you choose the ones to use in your own translation strategy?

The following are three general guidelines to follow when planning and managing ID translation:

To Plan ID Translation

1. Make read-only access the "default" access granted at each node.

   - To avoid automatically granting write access to unknown users, do not add both wildcard GID and wildcard UID translation entries to the Network Users/Groups Table.
   - Add UID translation entries only for those network users with write access to the node. Network users do not need a translated user ID for read-only access.

2. Use UID translation entries to control write access:

   - Map one network user to one local ID.

     Do not map several network IDs to one local ID unless all IDs belong to the same network user (and therefore will not be simultaneously active on the network).
   - Create a local user ID for each network user with write access to the node.

     Do not map a network user to an already existing local ID unless both IDs really belong to the same user.

3. Use wildcard NID entries with caution on a large network (especially one in which you do not have customizing authority on all nodes).

Additional Guidelines

- Use one-to-one translation entries to grant write access.

  This lessens the chances of having more than one user ID "own" a file. You should minimize the possibility of several users' writing simultaneously to the same file, unless they are using applications specifically designed to manage such access. Otherwise, the changes made by one user may be lost.

  For example, if two **ed** editing sessions have the same remote file open simultaneously, only the changes made by the session that ends first are saved. If the work from both editing sessions is to be saved, the first one to end should write the file to a different file name, the second one to the original file name. This strategy assumes, of course, that the first user knows that the second has the file open. One way to reduce the possibility of such conflicts is to use relatively restrictive file permissions. Another is to use restrictive ID translation. Using both restrictive ID translation and restrictive file permissions (for example, write only by owner) is probably the safest strategy to follow.

- Grant read-only access to users who only need to access remote commands.

- Grant read-only access to unknown users, especially if you are not responsible for adding new users to all the nodes in the network. For nodes that you have control over, you might have both a global UID and a global GID entry for your own (temporary) convenience.

- As a general rule, grant read-only access (or no access) to any remote ID that is not normally used as a login ID.

- Consider carefully the potential consequences of granting read, write, or no access to system IDs such as **root**, **system**, **mail**, and those IDs belonging to system daemons (**tftpd** and **smtpd**, for example). You may need to find out how the IDs installed by some licensed programs (such as TCP/IP) are used by these programs to judge whether or not they need to be included in your translation plan.

If you want to run remote set-user-ID or set-group-ID commands, the corresponding user or group IDs must translate to the same ID on both the server and client systems. Since most of the AIX set-user-ID programs belong to UID **0**, you must translate UID **0** to UID **0** to be able to run these set-user-ID and set-group-ID commands as remote programs.

If you want to perform remote network administrative tasks, you must translate an incoming Group ID to GID **0** at the nodes you want to customize. This does not, however, mean that you must translate GID **0** to GID **0**, just that some group on the server must translate to **system** group on the client. If you do translate GID **0** to GID **0**, any network user that is a member of the local **system** group can do such things as change your network tables.

One conservative approach to implementing remote administration is to create a special ID that can be mapped to user **root** or a special group that can be mapped to **system**. Be sure to limit access to the **/etc/passwd** and **/etc/group** files and the **users** command on the system where this login ID or group resides.

Finally, if you do not map either **root** or **system** to some local ID (**guest** or **usr**, for example), network nodes may not be able to issue mounts automatically at system start up, since these are normally done by **root** through the **/etc/rc.ds** initialization file.

Building Distributed File Trees

The following are points to consider when planning and managing distributed file trees:

To Select the Type of Mounts to Use

1. Restrict your use of file-on-file mounts to special cases:

   - The files required to implement the Single System Image Environment (see "Managing the Single System Image Environment" on page 3-70).

   - Read-only mounts of programs that you want to run remotely, but which reside in directories that you do not want to mount as a whole (perhaps to avoid the problems discussed in identifying and managing dependencies on page 3-66).

2. Use directory mounts to build those portions of the file tree to which users will have write access.

More Detailed Information

1. Restrict your use of file mounts to special cases:

   - The files required to implement the Single System Image Environment (see "Managing the Single System Image Environment" on page 3-70).

   - Read-only mounts of programs that you want to run remotely, but which reside in directories that you do not want to mount as a whole (perhaps to avoid the problems described in the discussion of identifying and managing dependencies on page 3-66).

2. Use directory mounts to build those portions of the file tree to which users will have write access.

   To understand the problems that using file-on-file mounts to build file trees can cause, you need to remember the relationship between a file tree and a file system and to remember how the enhanced mount facility has changed that relationship.

- The restrictions against linking or renaming files across file system boundaries still remain.

- The new mount facility now makes it possible for two files in the same directory to be in two different file systems and for a file to be in a different file system than its parent directory.

- Many programs (especially editors), designed for the old mount environment, assume that a file is part of the same file system as its parent directory. They may try to use the **ln** or **mv** commands to create temporary copies or backup copies of files. They may create a new file to hold the changed version of the existing file; however, that file resides in the same file system as the parent directory, not as the original file. Even if the program copies the new version back to the original file, there can be additional problems. For example, the **INed** editor generates a new i-node when it modifies a file. If a user modifies the file at the server, the client node continues to use the original i-node (which is the one actually mounted at the client node). Thus, the **INed** program should not generally be used to edit a file on the server that is mounted over a file at the client node. Since users at the server are generally unaware of the mounts issued by client nodes, a client node should not build a file tree that allows this problem to develop.

When selecting what directories or files to mount from a program server, consider the following:

To Select Mounts from a Program Server

1. The following directories contain programs that cannot (or should not) be run from a client node:
   - **/bin**
   - **/etc**
   - **/usr/lpp**
2. The following directories, in general, contain programs or support facilities that can be accessed from a client node:
   - **/usr/bin**
   - **/lib**, **/usr/lib**
   - **/include**, **/usr/include**
   - **/usr/mail**
   - **/usr/news**

More Detailed Information

1. The following directories contain programs that cannot (or should not) be run from a client node:

/bin	This directory contains the **sh**, **csh**, and **su** commands, as well as many other basic programs that you want to have available at all times.
/etc	This directory contains system administrative commands, such as **fsck**, **mkfs**, **shutdown**, **devices**, **minidisks**, **mount**, and **umount**. It also contains many system daemons, such as **cron** and **qdaemon**.

Not only do these commands need to be available at times, but many can only be run locally.

/usr/lpp	This directory contains locally installed licensed programs. Distributing licensed programs is subject to the licensing agreements, restrictions, and requirements imposed by their owners. **Note:** Licensed programs usually install some programs in other system directories, too, such as **/usr/bin**.

2. The following directories, in general, contain programs or support facilities that can be accessed from a client node:

Directory	**Possible Restrictions**
/usr/bin	This directory can contain set-user-ID and set-group-ID programs that a remote user can run only if the supporting ID translation entries exist in the Network Users/Groups Table (see page 3-61). This directory can contain licensed program-installed programs that cannot (or should not) be run by remote users.
/lib	This library contains the shared library (**sharsys.0201**) used by many AIX commands. A local copy of this library needs to be available for use when the server is off line (see the discussion of identifying and managing dependencies on page 3-66).
/usr/lib **/include** **/usr/include** **/usr/news**	
/usr/mail	To share this directory, the **mail** group (GID **6**) needs to be translated at each node.

In addition, when data files, system files, or programs reside on another node, each client node needs to identify critical dependencies and have a plan for managing those dependencies if those files or programs should be unavailable. These problems usually fall into one of two areas:

The server node is not on line when the client node starts up.

To manage this problem:

- Be sure that there are local copies of any programs that users must have access to at all times.
- Alternatively, you might customize a secondary program server. Files and directories from this system would not be mounted automatically, but would be available if the first server were inaccessible.
- Use a shell program to issue automatic remote mounts, so that they can be retried until successful. See "Automatic Mount Procedures" on page 3-74 for a complete discussion of this alternative.

The server node goes off line while the client node is using remote resources.

Depending on what system resources you are using at the server, the following are possible problems you can have under these circumstances:

- You can no longer run many local commands.

  When you attempt to run a command, you get the following message:

  ```
  Shared library: sharsys.0201 cannot be mapped.
  killed
  ```

  The command is trying to use the library **/lib/sharsys.0201**, but the **/lib** directory on the server is inaccessible because the server is off line or there is some other network problem. Without access to this library, most commands cannot run.

To solve this problem, provide a local copy of the shared library for local commands to use when **/lib** is inaccessible:

1. Create a new directory named **/etc/suidlibs**:

   ```
   # mkdir /etc/suidlibs
   ```

   The name of the new directory must be **/etc/suidlibs**, because set-user-ID programs (such as **mount** and **umount**) are already coded to look for a directory of this name.

2. Copy **/lib/sharsys.0201** to this directory:

   ```
   # cp /lib/sharsys.0201  /etc/suidlibs
   ```

3. Set the **LIBPATH** shell variable by adding the following lines to the **/etc/profile** file:

   ```
   LIBPATH=/lib:/usr/lib:/etc/suidlibs
   export LIBPATH
   ```

   The **LIBPATH** shell variable identifies which directories the shell should search for shared libraries and the order in which they should be searched.

- Users can no longer log in or run commands that read the **/etc/passwd** file.

  For example, a user tries to log in with a valid user name, but receives the following message:

  ```
  You entered a login name or password that is not valid.
  ```

  or a user tries to use the **su** command to change to a valid user ID, but receives the following message:

  ```
  Unknown id
  ```

  The AIX Operating System must read the **/etc/passwd** file to get the UID that corresponds to a user name. If the **/etc/passwd** file is remote, the local AIX Operating System cannot access it when the remote system is off line. This problem is complicated by the fact that, under these circumstances, if a user with superuser (system group) authority is not logged on when the server goes off line, it will not be possible to unmount remote files or directories or to shut down the client node until the server is accessible again. In addition, if no user is logged on, no one will be able to log on until the server is accessible again.

To solve this problem, you must provide some way for a non-system group user to unmount an inaccessible **/etc/passwd** file or for the system to monitor that file and to unmount it when it becomes inaccessible.

To allow a non-system group user to unmount an inaccessible password file, compile the program listed in Figure 3-4 on page 3-69 (a copy of this sample program can be found in the **/usr/lpp/ds/samples** directory). Use the following commands to make this program available to users:

```
# cd /usr/lpp/ds/samples
# cc umpasswd.c -o umpasswd
# chown root umpasswd ; chgrp system umpasswd
# chmod u+s umpasswd
# cp umpasswd /etc
```

The `chmod u+s` command makes this a set-user-ID program (the **cc** command has already made it executable by all users). This gives a user the necessary authority to issue the **uvmount** system call in the program. (Note that the program only unmounts the password file if it cannot open it. Otherwise, it exits without taking any action.)

To have the system run this program periodically, schedule it to be run by the **cron** command. Then even if no user is logged on, the password file will eventually be unmounted once it becomes inaccessible.

```
/*  umpasswd.c */

#include <stdio.h>
#include <fcntl.h>

extern int errno;

main()
{
   int fildes;
   int rc;


   errno = 0;
   fildes = open("/etc/passwd", O_RDONLY);

   if (fildes == -1) {
      printf("/etc/passwd file cannot be opened, ");
      printf("(errno %d)\n", errno );

      errno = 0;
      rc = uvmount("/etc/passwd");
      if (rc == -1) {
         printf("unmount of /etc/passwd failed, ");
         printf("errno %d\n", errno );
      } else {
         printf("/etc/passwd umounted \n");
      }
   }
}
```

Figure 3-4. The umpasswd.c Sample Program. This program unmounts an inaccessible **/etc/passwd** file.

Managing the Single System Image Environment

A multi-node network can be said to have a ***single system image*** when the following conditions are true:

To Provide a Single System Image

1. Users have the same identity on each node in the network:

   - The same login name
   - The same UID
   - The same GID
   - The same concurrent group list.

2. Users see the same file tree at each node.

3. No matter which node runs the **users** command, a new user (or group) is added to (or deleted from) all nodes.

4. The system initialization files issue all mounts necessary to achieve the single-system-image environment.

More Detailed Information

1. Users have the same identity on each node in the network.

   To achieve this, you must have all nodes share the same copy of the following system files:

   - **/etc/passwd**
   - **/etc/opasswd**
   - **/etc/group**
   - **/etc/ogroup**
   - **/usr/adm/user.cfile**

   In order for the **passwd** and **users** commands to run in this environment, all five of these files must reside on the same node and be mounted at each client node in the network, using file-on-file mounts (the **/etc** directory cannot be mounted remotely).

Another candidate for a file-on-file mount in this environment is the **/etc/motd** file. (Sharing this file allows you to place a special message in the local copy of **/etc/motd** so that a user can be warned when the server is inaccessible.)

2. Users see the same file tree at each node.

   Each node mounts the same files and directories from all the servers in the network. The simplest single-system-image environment to customize has one server node and multiple client nodes.

3. No matter which node runs the **users** command, a new user (or group) is added to (or deleted from) all nodes.

   To accomplish this goal, each node must build the same **/u** file tree. There are two ways to do this:

   Method One

   The **/etc/filesystem** file at each node contains a stanza for each home directory at all the other nodes. The **/u** directory at each node contains empty stub directories for each remote home directory. At system start up, each remote home directory is mounted on the corresponding stub directory in the local **/u** directory. When a local system administrator adds a new user, the network administrator adds a new stanza to each **/etc/filesystems** file and makes a new stub directory at each remote node.

   Method Two

   The complete **/u** file tree is built at the server, and then each client node mounts that **/u** directory as an ***inherited*** mount. This approach is somewhat more complicated to set up initially, but easier to administer once it is set up. With this arrangement, all new home directories are created automatically at the server. Alternatively, if you create new home directories at client nodes, only the server must modify its **/etc/filesystems** file for the rest of the network to pick up this new mount.

To set this up, use the following steps:

a. At the client node, alter the stanza in **/etc/filesystems** file that defines the default mount point of the **/dev/hd1** device (**/u**) as follows:

   Original Stanza:

```
/u:
        dev       = /dev/hd1
        vol       = "/u"
        mount     = true
        check     = true
        free      = true
```

   New **/dev/hd1** stanza:

```
/mnt:
        dev       = /dev/hd1
        vol       = "/mnt"
        mount     = true
        check     = true
        free      = true
```

   New **/u** stanza:

```
/u:
        dev        = /u
        nodename   = Server
        mount      = inherit,false
        type       = string
```

   The reason that the **/dev/hd1** device must be mounted in an alternative location is to prevent its being covered up by the first mount in the series of mounts that will be issued as a result of inheriting **/u** from the server.

   Once the entire **/u** directory is mounted from the server, running the **users** command at any node in the network creates the new home directory at the server. Thus it is automatically inherited by each client node the next time the client reissues this mount.

If you want the new home directories created at the client node that runs the **users** command, alter the **/usr/adm/user.cfile** as follows:

Replace the line:

```
udir    /u/
```

with the line:

```
udir    /mnt/
```

The **users** command now creates home directories in the **/mnt** directory rather than in the **/u** directory.

Note: Since this is a shared file, this change applies to home directories created at the server.

b. At the server, create a stub directory in **/u** for each home directory at the client nodes and add stanzas to **/etc/filesystems** to mount the remote home directories (from the remote **/mnt** directory) onto those stub directories.

4. The system initialization files issue all mounts necessary to achieve the single-system-image environment. See "Automatic Mount Procedures" on page 3-74, especially "Single System Image Mounts" on page 3-80 and "Inherited Mounts" on page 3-82, for a complete discussion of the available options.

Automatic Mount Procedures

Note: Copies of the sample shell programs used in the following discussion of automatic mount procedures can be found in the **/usr/lpp/ds/samples** directory.

There are a number of automatic mount procedures, depending, in part, on whether or not remote mounts are inherited.

Mounts That Are Not Inherited

If your remote mounts are not inherited, automatic mount procedures need only be implemented at the node issuing the remote mount.

There are at least two way to reschedule mounts:

- Have a shell program use the **at** command to rerun itself.
- Run a background shell program that does not end until its mounts are all successful or until it reaches some specified limit.

Rescheduling Mounts: the at Command

The sample shell program `retry_mount` shown in Figure 3-5 on page 3-75 illustrates one way to use the **at** command to reschedule mounts. This shell program issues a mount, tests its return value, and reschedules itself if the mount is unsuccessful. This process continues until the mount succeeds.

```
# @(#)retry_mount
#
# SHELL SCRIPT NAME:  RETRY_MOUNT
# USAGE:  retry_mount  type  interval

msgpath=${msgpath=/usr/adm/mnt.msg}

if test $# -ne 2
then
    echo "Usage:  retry_mount  type  interval"
    exit
fi
/etc/mount -t $1 >>${msgpath} 2>&1
if test $? -eq 0
then
    exit
else
    echo "retry_mount $1 $2" | at now + $2 minute >>${msgpath} 2>&1
fi
```

Figure 3-5. The retry_mount shell program. Reschedules itself.

The retry_mount program should be run from the **/etc/rc.ds** initialization file if any remote mounts fail. For example, see the following command list from the **rc.ds** file shown in Figure 3-6 on page 3-76:

```
mount -t remote >/usr/adm/mnt.msg 2>&1 || retry_mount remote 5
```

The commands in this list are separated by two || (vertical bars). This causes the second program (retry_mount) to run only if the first program (mount) fails. retry_mount also uses ***conditional substitution*** to assign a value to the user-defined variable msgpath:

```
msgpath=${msgpath=/usr/adm/mnt.msg}
```

This line assigns the value /usr/adm/mnt.msg to the variable msgpath if that variable has not already been set on the command line. Conditional substitution can be useful if normally you expect to use a particular value, but want to be able to change that value if necessary. To reset this variable, enter a command such as the following:

```
msgpath=/dev/null retry_mount remote 5
```

This command line assigns all output from the mount commands to /dev/null.

See the **sh** command in *AIX Operating System Commands Reference* for a description of available command-list separators and terminators and for a discussion of user-defined variables and conditional substitution.

```
# @(#)rc.ds     5.3 87/02/05 11:05:49

echo Translate tables are being loaded ...
dsxlate

echo IPC keys are being loaded ...
dsipc

echo Starting SNA ...
start sna

echo Starting attachment ...
start /attachment/EDEFAULT&

echo Starting kernel processes ...
dsstate -p100 -k -ce -se -sa -aa

echo Mounting remote directories ...
mount -t remote >/usr/adm/mnt.msg 2>&1 || retry_mount remote 5
```

Figure 3-6. An rc.ds file. Automatic start up (at command)

Using a Shell Program that Sleeps

Another automatic mount procedure runs a background shell program that sleeps for a specified interval and then reissues the required mounts until they are all successful. The do_mounts program in Figure 3-7 is an example of such a program. The only exit from this program appears in the command list:

```
mount -t $1 >>${msgpath} 2>&1 && exit
```

The && separator causes exit to run only if the mount is successful.

```
# @(#) do_mounts -- Perform mounts of a certain type until successful
#
# usage:  do_mounts mount-type delay
#

msgpath=${msgpath=/usr/adm/mnt.msg}

while true
do
  mount -t $1 >>${msgpath} 2>&1 && exit
  sleep `expr 60 \* $2`
done
```

Figure 3-7. The do_mounts shell program

On the other hand, you may want to limit the number of times your system retries a mount. You may not want to tie up system resources indefinitely. Perhaps some nodes provide only limited file services or are not always on line. In this case, you could use a shell program such as the one in Figure 3-8 on page 3-78 to limit mount attempts. This version of the domounts shell program accepts an optional third command line argument that specifies the number of times that the program is to attempt the mount. If you do not specify a third argument, the program continues to loop until the mount is successful. The following line sets the default limit (0):

```
limit=${3-0}
```

The program, using conditional substitution to assign a value to a variable named `limit`, checks to see if a third argument is set on the command line ($3). If it is, `domounts` assigns that value to `limit`. If it is not set, `domounts` assigns the value 0 to `limit`.

```
# @(#)domounts -- Perform mounts of a certain type
#
#  usage: domounts mount-type delay1 [ trys ]
#

# Assign defaults
msgpath=${msgpath=/usr/adm/mnt.msg}
limit=${3-0}
i=1

while [ ${limit} -eq 0 -o ${i} -le ${limit} ]
do
        /etc/mount -t $1 >>${msgpath} 2>&1 && exit
        sleep `expr 60 \* $2`
        i=`expr ${i} + 1`
done
```

Figure 3-8. The domounts shell program

Have the **rc.ds** file run `do_mounts` ***in the background*** so that **rc.ds** can complete system startup. See Figure 3-9 on page 3-79.

```
# @(#)rc.ds     5.3 87/02/05 11:05:49
.
.
.
echo Starting kernel processes ...
dsstate -p100 -k -ce -se -sa -aa

# clear the message file
>/usr/adm/mnt.msg

echo Mounting remote directories ...
domounts primary 2 &
domounts secondary 5 10 &
```

Figure 3-9. An rc.ds file. Automatic start up (process that sleeps)

Single System Image Mounts

The single system image automatic mount procedure needs to handle both the making of backup copies of the relevant shared files as well as the mounting of the shared file and directories. See the `ssimounts` shell program in Figure 3-10 on page 3-81. This shell program issues remote mounts in two stages. First it mounts the master remote files—**/etc/passwd**, **/etc/opasswd**, **/etc/group**, and **/etc/ogroup**—over temporary files and makes local copies so that each system has a recent copy of the master files. Then it mounts all files and directories from the server node. This shell program is run by **/etc/rc.ds**, as shown in Figure 3-11 on page 3-82.

The four files to be copied are mounted twice; first over a temporary file, from which they can be copied and then unmounted, then over the local copy just made. The stanzas in **/etc/filesystems** that define the temporary mounts of the four files include the attribute **type = stage1**. The remaining stanzas that define remote mounts contain the attribute **type = stage2**. This allows `ssimounts` to wait to perform the second set of mounts until after the first set of mounts are completed. Since all home directories, most data files, and many programs are not available until all these mounts have been completed, the procedure retries unsuccessful mounts as quickly as possible until they have all succeeded.

```
# @(#)ssimounts -- Perform single system image mounts
#
#  usage: ssimounts mount-type1 mount-type2 [ delay ]
#

# Assign defaults (delay is in seconds)
msgpath=${msgpath=/usr/adm/ds.msg}
delay=${3-0}

touch /etc/passwd.tmp /etc/opasswd.tmp /etc/group.tmp /etc/ogroup.tmp

until mount -t $1 >>${msgpath} 2>&1
do
        sleep ${delay}
done

cp /etc/passwd.tmp /etc/passwd
cp /etc/opasswd.tmp /etc/opasswd
cp /etc/group.tmp /etc/group
cp /etc/ogroup.tmp /etc/ogroup

umount allr  >>${msgpath} 2>&1

rm /etc/passwd.tmp /etc/opasswd.tmp /etc/group.tmp /etc/ogroup.tmp

until mount -t $2 >>${msgpath} 2>&1
do
        sleep ${delay}
done
```

Figure 3-10. The ssimounts shell file. Sets up a single-system-image environment

```
# @(#)rc.ds     5.4 87/04/13 09:29:36
.
.
.
echo Starting kernel processes ...
dsstate -p100 -k -ce -se -sa -aa 2>&1

echo Mounting remote directories ...
ssimounts stage1 stage2 &
```

Figure 3-11. An rc.ds file. Run at each client node

Inherited Mounts

To get all the remote mounts in an inherited file tree, the node issuing an inherited mount must not issue that mount until after the tree has been built at the Server. Following are two solutions to this problem, one implemented from the server node (the "Smart Server"), and one implemented from the client node (the "Smart Client").

Smart Server: To implement this solution, a server node uses the **dsstate** command to block incoming requests until the file tree to be inherited has been completed or until a specified interval has passed. Then it reenables incoming server requests and, if necessary, continues reissuing the mounts. See Figure 3-12 on page 3-83.

By blocking incoming requests, the server node prevents other nodes from mounting a partially finished file tree. This solution is suited to environments where there is only one server node building file trees to inherit (and thus blocking incoming requests).

To enable this approach, the server node runs an **rc.ds** initialization file similiar to Figure 3-13 on page 3-84, while each client node runs an **rc.ds** file like the one in Figure 3-9 on page 3-79.

```
#  @(#)repeatmounts -- Attempt to build file tree to be inherited
#
#  usage: repeatmounts mount-type delay1 tries delay2
#

msgpath=${msgpath=/usr/adm/mnt.msg}

# loop n times controlled by third parameter
i=1
while [ i -le $3 ]
do
  mount -t $1 >>${msgpath} 2>&1 && { dsstate -sa ; exit ; }
  sleep `expr 60 \* $2`
  i=`expr ${i} + 1`
done

# Reached limit on attempts, allow server requests
dsstate -sa >/dev/null 2>&1

# Keep trying to mount, if necessary
while true
do
  mount -t $1 >>${msgpath} 2>&1 && exit
  sleep `expr 60 \* $4`
done
```

Figure 3-12. The repeatmounts shell program. Builds an inherited file tree

Note: Be sure that the **/etc/filesystems** file stanzas of all inherited mounts contain the attribute **mount=inherited**.

```
# @(#)rc.ds     5.3 87/02/05 11:05:49
.
.
.
# Disable server requests (change -sa to -sb)
# Enable all other requests (-aa)
echo Starting kernel processes ...
dsstate -p100 -k -ce -se -sb
dsstate -aa

echo Mounting remote directories ...
repeatmounts filetree 1 10 4 &
do_mounts remote 5 &
```

Figure 3-13. An rc.ds file. Automatic startup of inherited mounts controlled by the Server

Smart Client: To implement this solution, the server node keeps a log of the **type** attribute of each successful mount. This can be done by enhancing the `domounts` shell program as illustrated in Figure 3-8 on page 3-78. This enhanced shell program is illustrated by `recordmounts` shown in Figure 3-14.

```
# @(#)recordmounts -- Perform and track mounts of a certain type
#
#  usage: recordmounts mount-type delay1 [ trys ]
#

# Assign defaults
msgpath=${msgpath=/usr/adm/mnt.msg}
typepath=${typepath=/usr/adm/mnt.types}
limit=${3-0}
i=1

while [ ${limit} -eq 0 -o ${i} -le ${limit} ]
do
        /etc/mount -t $1 >>${msgpath} 2>&1 \
                && { echo $1 >>${typepath} ; exit; }
        sleep `expr 60 \* $2`
        i=`expr ${i} + 1`
done
```

Figure 3-14. The recordmounts shell program

After each successful mount, `recordmounts` writes the mount type in a file. The client node checks this log file, and does not attempt an inherited mount until this file contains a specified mount type, as illustrated by the `clientmounts` shell file shown in Figure 3-15 on page 3-86.

```
# @(#)clientmounts -- Wait for a complete file tree before mounting it
#
# usage: clientmounts  srvr-nid  type  delay  stub-path

# Assign defaults
msgpath=${msgpath=/usr/adm/mnt.msg}
typepath=${typepath=/usr/adm/mnt.types}

# mount file containing types mounted at server
mkdir /tmp/$$
touch /tmp/$$/mnt.types
while true
do
  mount -n $1 ${typepath} /tmp/$$/mnt.types >> ${msgpath} 2>&1 && break
  sleep `expr 60 \* $3`
done

# keep checking for entry in file indicating server mounts are done
until fgrep $2 /tmp/$$/mnt.types >/dev/null
do
  sleep `expr 60 \* $3`
done

# do inherited mount
mount -i $4

# clean up
umount /tmp/$$/mnt.types >>${msgpath} 2>&1
rm -r /tmp/$$
```

Figure 3-15. The clientmounts shell program. Wait for a complete file tree before mounting it

The `clientmounts` shell program uses its process ID ($$) to create a temporary directory and null file on which to mount the server's log file. It keeps scanning this file until it contains a line matching the mount type that `clientmounts` is looking for. Only then does it issue its inherited mounts.

The shell program shown in Figure 3-16 on page 3-87 manages the building of inherited file trees at the server, insuring that server requests are not reenabled until all the specified file trees have been completed. The **rc.ds** file shown in Figure 3-17 on page 3-88 runs at the Server; the file shown in Figure 3-18 on page 3-88 runs at the client node.

```
# @(#)buildtrees -- coordinate mounts
#
# usage: buildtrees mount-type1 mount-type2 ...&
#

# Assign defaults  (delays are in minutes)

typepath=${typepath=/usr/adm/mnt.types}
delay1=${delay1=1}
tries=${tries=10}
delay2=${delay2=5}

# empty out record of mounts by type
>${typepath}

for t
do
        recordmounts ${t} ${delay1} ${tries} &
done

wait
dsstate -sa >/dev/null 2>&1

for t
do
        fgrep ${t} ${mspath} >/dev/null 2>&1 || \
                recordmounts ${t} ${delay2} &
done
```

Figure 3-16. The buildtrees shell program. Coordinates mounts

```
# @(#)rc.ds     5.3 87/02/05 11:05:49
.
.
.
# Disable server requests (change -sa to -sb)
# Enable all other requests (-aa)
echo Starting kernel processes ...
dsstate -p100 -k -ce -se -sb
dsstate -aa

# Retry mounts if not successful
echo Mounting remote directories ...
buildtrees filetree1 filetree2 &
```

Figure 3-17. An rc.ds file. Server node logs successful mounts

```
# @(#)rc.ds     5.3 87/02/05 11:05:49
.
.
.
echo Starting kernel processes ...
dsstate -p100 -k -ce -se -sa -aa

echo Mounting remote directories ...
clientmounts node_a filetree1 2 /u &
clientmounts node_a filetree2 5 /library &
domounts remote 5 10 &
```

Figure 3-18. An rc.ds file. Client waits for Server to log successful mounts

Customizing a Server Node

Take the following steps to customize each RT PC that allows remote access to local files or programs:

To Plan a Server Node

1. Identify network requirements.
2. List the node ID (NID) of each RT PC that has remote access to local files and programs.
3. List the names of all network users and groups that have remote access to local files and programs. Include both the user or group ID and the node ID of the originating system.
4. Decide how to manage the translation of incoming and outgoing user and group IDs.
5. Build the necessary node and ID tables.
6. Check the permissions of local files and directories to insure that they provide or restrict network access as you intend.
7. Give client nodes the information they need to access local files and programs.

More Detailed Information

1. Identify network requirements. Consider the following:
   - To which files and directories do network users need access?
   - What kind of access is needed—search or execute permission, read permission, or write permission?
   - Which users need access—all users (local and remote), all network users, selected network and/or local users?

- Which groups need access—all groups (local and remote), all network groups, selected network and/or local groups?

- Who should be denied access?

2. List the node ID (NID) of each RT PC with remote access to local files and programs. To learn the NIDs, run the following command at each RT PC on the network:

   ```
   uname -m
   ```

   If you do not have access to all the RT PCs on the network, you will need to request this information from the local system administrator. If your network is part of a Baseband (Ethernet) LAN and you have installed TCP/IP, you may be able to use remote login to collect this information.

   As a general rule, you should also assign a 1- to 14-character nickname to each node. Associating a nickname with a NID can make it easier to build the tables or use the commands (such as **mount**) that require you to specify a NID.

   A node has only one valid NID, but it can have as many nicknames as it has client systems. Each system that accesses a node can assign it a different nickname. For example, Node A has NID `20810CBF`. Node B assigns the nickname `Tom` to Node A, while Node C assigns the nickname `Server`. The profiles at each system translate the locally defined nickname to NID `20810CBF`, which uniquely identifies Node A on the network.

3. List the names of all network users and groups with remote access to local files and programs. This list should include both the user or group ID and the node ID (or nickname) of the originating system.

4. Decide how you want to manage the AIX Operating System's translation of incoming and outgoing user and group IDs (see "Managing ID Translation" on page 3-58).

5. Build the necessary tables. Run the **ndtable** command to add a node profile for each node on the network to the Distributed

Network Node Table. Run the **ugtable** or **dsldxprof** command to add translation entries to the Network Users/Groups Table.

> **Note:** If creating additional local user or group IDs is part of your plan to manage network IDs, add them to the **/etc/passwd** or **/etc/group** file before building the Network Users/Groups Table. (See the **users** command in *AIX Operating System Commands Reference.*)
>
> While you can create either table first, if you create the Network Users/Groups Table before you create the Distributed Network Node Table, you get a message with each entry warning you that the node is not known.

6. Check the permissions settings of local files and directories to insure that they provide or restrict network access as you intend.

   If you do not map network user or group ID 0 to local ID 0 (and as a general rule you should not), be sure the permissions of directories that are available for remote access are set to `search by others` (allowing other nodes to mount the directory) and to `read by others` (allowing users at other nodes to list the contents of the directory with commands like **li** or **ls**).

   Conversely, you can set the search permissions on a directory so that a remote user cannot mount or read that directory, but keep in mind that this will restrict local access as well. You can, however, use concurrent group membership at the local node and careful mapping of remote IDs to deny access to remote users while allowing it to local users, if such security is important.

7. Give client nodes the information they need to access your files and programs. Include the full path names of available resources, the path names that should be mounted as inherited mounts, and a list of network IDs.

Customizing a Client Node

Take the following steps to customize each RT PC that accesses remote files or programs:

Planning a Client Node

1. Identify network requirements.
2. List the node IDs of each RT PC that has files or programs that you plan to use.
3. If local users or groups have been assigned network IDs, list each user or group name and its corresponding network ID.
4. Decide where you want to mount remote directories or files and create empty directories or files to mount over, if necessary.
5. Build the necessary node and ID tables.
6. Add stanzas to the **/etc/filesystems** file to describe remote mounts.
7. Modify the **/etc/rc.ds** file to enable automatic or graceful startup, if desired, and to block all incoming requests (`dsstate -sb`), if you do not want other nodes to have access your file systems.

Scenario One: A Program Development Environment

Joe manages a programming department within a large company. His department provides programming support to others in the company. Joe's department employs four programmers, a technical writer, and an administrative assistant. The group has five RT PCs, as shown in Figure 3-19 on page 3-94. These RT PCs are connected via a Baseband (Ethernet) LAN and configured for TCP/IP, SNA Services, and Distributed Services.

Joe, Gary (the lead programmer), and Don (administrative assistant) share one RT PC, which is customized with a lot of disk space. It is also connected to both a 3812 Pageprinter and an IBM 5201 Quietwriter printer. Users at RT PCs on the LAN can access either printer by using TCP/IP's remote printing facility.

This shared system also provides services and resources to the other members of the network, and operates as a ***server*** in the network. It provides the following services and resources:

- File and process services to the other RT PCs on the LAN
- Disk space for:
  - A central code library
  - A documentation library
  - System mail
  - News
- Extended Services programs for programmers use, including:
  - **INed**
  - **vi**
  - SCCS (source code control system).
- Locally developed tools for:
  - Project tracking
  - Documentation development
  - Additional programming support.

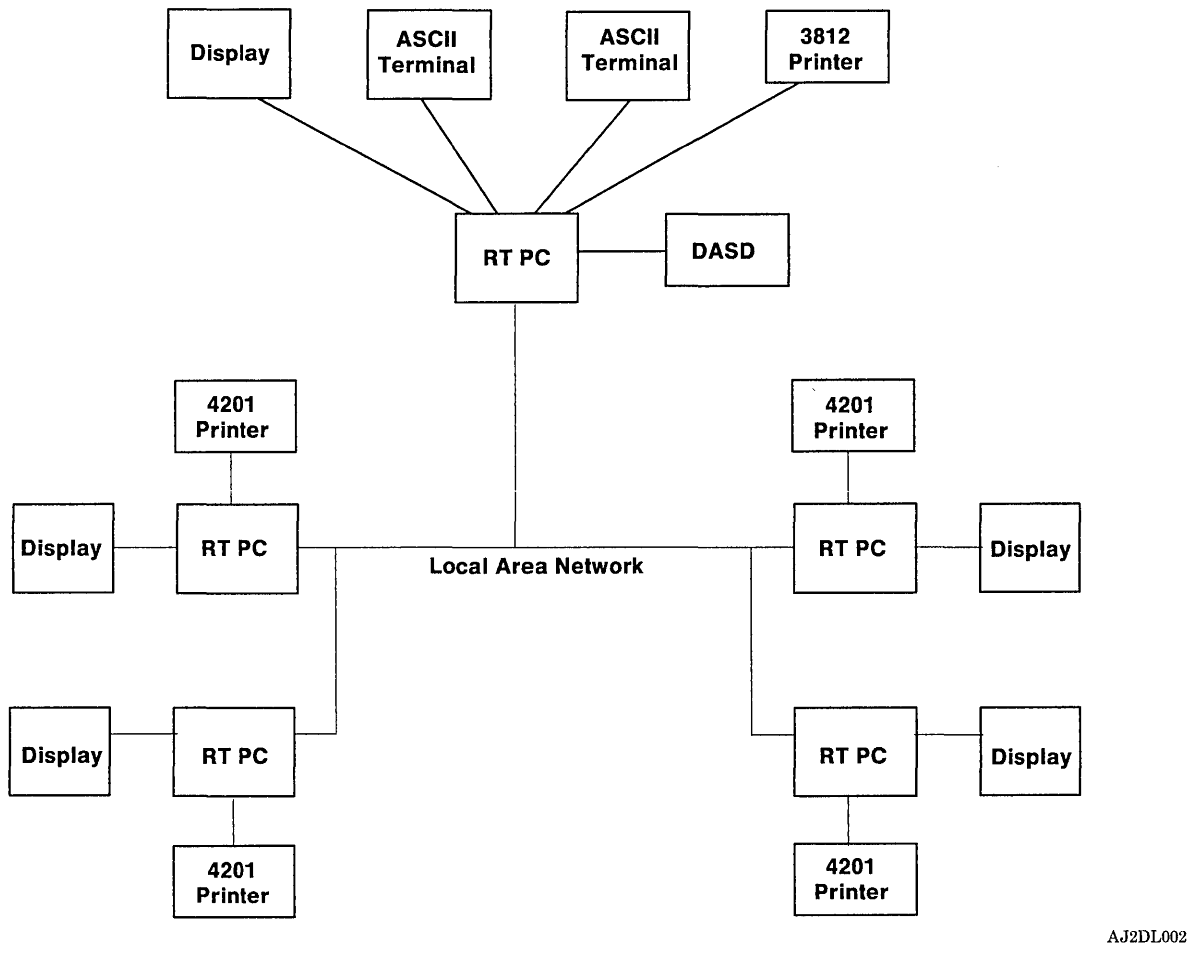


Figure 3-19. A Programming Environment

Other members of the department can access these programs as needed. This RT PC is on line at all times during working hours.

The three programmers—Ellen, Hal, and Liz—and the technical writer, Todd, each work at one of the four remaining RT PCs on the network. They develop their portions of the code or documentation at their own system, and store each version in the common code library as schedules warrant. They have not installed **SCCS**, but access the commands installed on the server system when they need them. They can also use most of the programming tools that they need from the server system without having them installed at their own system. These four RT PCs are not always on during a working day. Their network connections are not always active. Ellen, Hal, Liz, and Todd must customize their own systems for network access, and mount and unmount files and directories as needed.

Customizing the Server

Gary creates the code and documentation libraries and mounts them on **/library1** in his root file system. (His login ID owns this directory.) He decides that he will create a new group on his system, called **proj1**, to own that directory. Later, after he has created that group with the **users** command, he must change the group associated with that directory to ensure that it has the proper permissions to allow access by the proj1 group:

```
chgrp proj1 /library1
li -ld /library1
drwxrwxr-x   2 gary     proj1        112 May 21 10:10 /library1
```

Joe assigns him the additional task of customizing his RT PC as the network file server.

Identifying Requirements: Gary begins by identifying the following network requirements:

1. The network should require a minimum amount of administrative overhead to maintain and to expand. Additional nodes and/or network users may be added in the future.

2. All network users need read and execute access to the following system directories:

Directory	Contents
/usr/bin	Contains the **SCCS** programs, the **INed** and **vi** editors, and other locally developed support tools. He notes that it has the following permissions and IDs associated with it: `drwxr-xr-x    bin    bin` Gary also notes that **/usr/bin** contains set-user-ID and set-group-ID programs that belong to **root** and to **system** group.
/usr/lib/help	Contains the information files used by the **SCCS help** command. It has the following permissions and IDs associated with it: `drwxr-xr-x    bin    bin`
/usr/news	Contains the files that are read by the **news** command. It has the following permissions and IDs associated with it: `drwxrwxrwx    bin    bin`
/usr/mail	Contains the mailbox files accessed by the **mail** command. It has the following permissions and IDs associated with it: `drwxrwxr-x    bin    mail` Gary also notes that the **mail** command runs as a set-group-ID command, owned by the **mail** group (GID 6).

3. Gary, Ellen, Hal, Liz, and Todd need read and write access to **library1**. This is the mount point for the file system that contains the shared library.

   Gary decides to use group ownership to control access to the shared library, network IDs to create flexible ID translation tables, and wild card translations to handle all other incoming

IDs (except for GID 6—**mail**—which will be mapped to itself). He will leave it up to each node to select which local IDs to map to these network IDs.

4. Network users will not have superuser authority, but incoming system UIDs and GIDs (such as **root** and **system**) will be allowed into the node as UID or GID 100.

Building the Node Table: Once Gary has identified his requirements, he makes a list of the NIDs of the other RT PCs and of the user and group IDs that need access to his RT PC, using information provided by Ellen, Hal, Liz, and Todd. He assigns the user's name as the nickname for each node to produce the following list of nodes:

Nickname	NID
hal	108133EB
liz	108133F5
ellen	2081064A
todd	20810CBF

With this information, Gary is ready to build his Distributed Network Node Table. He uses the **ndtable** command to add each NID and its corresponding nickname, letting the system provide the other choices by default. This produces the following node table:

Remote Nickname	Remote Node ID	Node Security	Data Link Type	Connection Profile	Attachment Profile
hal	108133EB	None	Ethernet	108133EB	108133EB
liz	108133F5	None	Ethernet	108133F5	108133F5
ellen	2081064A	None	Ethernet	2081064A	2081064A
todd	20810CBF	None	Ethernet	20810CBF	20810CBF
-	208131AA	None	Ethernet	CDEFAULTCFD	CDEFAULT

Building the Users/Groups Table: Next he talks to each of the other four people on the network to get the following list of information about network users and groups:

Name	U/G	ID	Node
ellen	U	201	ellen
liz	U	201	liz
todd	U	201	todd
hal	U	201	hal
staff	G	1	ellen, liz, todd, hal
system	G	0	ellen, liz, todd, hal
mail	G	6	ellen, liz, todd, hal

To allow these users to access the server system, he runs the **users** command to add four new user IDs and to create one new group, as follows:

Name	U/G	ID	Comment
ellen	U	203	
liz	U	204	
hal	U	205	
todd	U	206	
proj1	G	200	members are ellen, liz, hal and todd

The new group, `proj1`, contains the IDs of all users who need access to **library1**.

He decides to assign the following network IDs to the new user and group IDs:

Name	U/G	ID	Network ID
ellen	U	203	10000
liz	U	204	10001
hal	U	205	10002
todd	U	206	10003
proj1	G	200	20000

The outbound translation tables at each node must map the local user and group IDs to their proper network IDs. If users change (or add) nodes, or if new users are added who should access these IDs, the outbound translations can be changed to include the new information.

Next he plans to map all requests from GID 6 (**mail**) to local GID 6, so that the **mail** commands running at each node have write access to the **/usr/mail** directory. All remaining IDs will be mapped to UID 100 (**guest**) and GID 100 (**usr**) to ensure that mount requests from **root** (**rc.ds**) are allowed.

Next he runs the **ugtable** command to enter this user and group information into the system, to produce the following Network Users/Groups Table:

```
Usr/Grp          Local         Network ID          Originating Node
 Name     U/G     ID     Outbound      Inbound     Name/Nickname

 ellen     U      203      10000        10000      *
 liz       U      204      10001        10001      *
 hal       U      205      10002        10002      *
 todd      U      206      10003        10003      *
 guest     U      100          -            *      *
 mail      G        6          -            6      *
 projl     G      200      20000        20000      *
 usr       G      100          -        90000      *
```

Customizing the Client

To connect to the server, each client (Ellen, Hal, Liz, and Todd) must analyze their expected needs and decide how to configure their system to achieve those needs. The following paragraphs summarize the important areas that they must consider when customizing their systems.

Network IDs: Each client must decide which local IDs should be mapped to the assigned network IDs. One client may map several local IDs to one network ID for access to shared library information. Others may require unique IDs for each ID on a system. If more network IDs are needed, they must ask Gary to create more network IDs.

File Tree: Each client must create mount points for the remote directories **library1** and **/usr/lib/help**. The mount points already exist for **/usr/bin**, **/usr/mail** and **/usr/news**.

Access: Each client must decide whether to block incoming requests or to allow selected network users to have access to the system. Incoming requests can be blocked either by running **dsstate** or by not having any inbound ID translation specified in the Users/Groups Table.

Start Up: Each client must decide whether to have:

- *Manual start up* — This method requires that each user enter the **mount** command for each file or directory to be mounted. The command can use information from **/etc/filesystems** or from the command line parameters to perform the mount.
- *Automatic start up* — This method requires that each of the mount points be defined by a stanza in **/etc/filesystems**. All **mount** commands are placed in **/etc/rc.ds** so that they are executed automatically when the system starts. However, if the server is not active when the client system starts, the mounts must be performed manually after the server starts.
- *Graceful start up* — This method requires that each of the mount points be defined by a stanza in **/etc/filesystems**. The user creates a shell procedure that tries to mount the required

files or directories, but retries them later if they are not successful on the first try. The user then puts the name of that procedure in **/etc/rc.ds** so that the procedure is executed automatically when the system starts. An example shell procedure is **retry_mount** in the **/usr/lpp/ds/samples** directory.

- *Some combination of the three* — This method allows you to automatically mount some files or directories through **rc.ds**, and to manually mount others that are not used often.

Set-user-id programs: Each client must decide what to do about the set-user- and set-group-ID programs in **/usr/bin** or any other mounted directory. To use any of these programs, copy them to a local directory that is searched before **/usr/bin** (or other mounted directory) in the **PATH**. For example:

```
find /usr/bin \( -perm -4000 -o -perm -2000 \) -exec mv {} /bin \;
```

is one solution.

When deciding which programs to copy from another system, consider the following points:

- Program licensing agreements may restrict moving programs from one system to another.
- Prerequisite programs and data files may need to be present on the local system for the copied program to work properly.
- If the program to be copied is part of an installed program on the other system and the installation for that program modified the kernel on the other system, the entire program must be installed on the local system to be able to use the desired program.

Programs Needed When the Server is Down: If the server system is not operating, any programs on that system will not be available for use at any of the client systems. Each client must decide which programs that it uses from the server system are important to the operation and use of the local system. The client should ensure that those important programs are installed on the local system so that they are always available.

Example: Ellen's Configuration

Ellen defines the network on her system using the **ndtable** command to produce the following node table:

Remote Nickname	Remote Node ID	Node Security	Data Link Type	Connection Profile	Attachment Profile
Hal	108133EB	None	Ethernet	108133EB	108133EB
Liz	108133F5	None	Ethernet	108133F5	108133F5
Ellen	2081064A	None	Ethernet	2081064A	2081064A
Todd	20810CBF	None	Ethernet	20810CBF	20810CBF
Gary	208131AA	None	Ethernet	208131AA	208131AA
Ellen	2081064A	None	Ethernet	CDEFAULTCFD	CDEFAULT

Ellen investigates how she and others will use the network. The following paragraphs describe how she considers each of the important areas when deciding how to configure her system in the network.

Network IDs: Ellen decides to restrict access to her local machine so that others on the network can only access her machine through the user ID, `guest`. She still wants to allow the mail system to work properly, however. In addition, she wants to access the server under her user ID (`ellen`) and the library group ID (`projl`), regardless of what user ID (`root` or `ellen`) or group ID (`system`, `staff` or `projl`) she may be using on her own system. After reviewing the IDs that Gary assigned at the server node, Ellen creates the following Network Users/Groups Table on her machine using the **ugtable** command:

Usr/Grp Name	U/G	Local ID	Network ID Outbound	Network ID Inbound	Originating Node Name/Nickname
ellen	U	200	10000	-	-
guest	U	100	90000	*	*
mail	G	6	6	6	*
root	U	0	10000	-	-
staff	G	1	20000	-	-
system	G	0	20000	-	-
usr	G	100	90000	*	*
projl	G	8	20000	-	-

File Tree: She creates a mount point for the **library1** directory in her **/u** directory, and a mount point for **/usr/lib/help** in her **/usr/lib** directory:

```
$ mkdir /u/library1
$ mkdir /usr/lib/help
```

Access: She limits external access to her machine through the limited ID translations that she provides in her User/Groups Table.

Start Up: To ensure access to the **library1** directory and to other system services, she decides to mount all directories during start up. To ensure that the files will be ready as soon as the server is available, she decides on a graceful start up approach. To implement this approach, she performs the following steps:

1. She adds the following stanzas to her **/etc/filesystems** file:

```
/u/library1:
        nodename = Gary
        dev = /library1
        type = prog_lib

/usr/lib/help:
        nodename = Gary
        dev = /usr/lib/help
        type = prog_lib

/usr/bin:
        nodename = Gary
        dev = /usr/bin
        type = prog_lib

/usr/mail:
        nodename = Gary
        dev = /usr/mail
        type = prog_lib

/usr/news:
        nodename = Gary
        dev = /usr/news
        type = prog_lib
```

The type value of prog_lib allows her to select these mounts as a group to be mounted.

2. After looking at the sample shell script, **retry_mount** in the **/usr/lpp/ds/samples** directory, she decides that it will serve her purpose without changes. She copies that file into the **/etc** directory and ensures that it has the correct permissions:

```
$ cp /usr/lpp/ds/samples/retry_mount /etc
$ li -l /etc/retry_mount
-r-xr-xr-x   1 ellen   system    370  May 21 15:42 /etc/retry_mount
$ chmod 754 /etc/retry_mount
$ chown root /etc/retry_mount
$ li -l /etc/retry_mount
-rwxr-xr--   1 root    system    370  May 21 15:42 /etc/retry_mount
```

3. She adds the following lines at the end of **/etc/rc.ds** to ensure that the remote directories of type prog_lib (as specified in **/etc/filesystems**) are mounted at start up:

```
echo Mounting remote directories ...
/etc/retry_mount prog_lib 3 &
```

This entry instructs the system to mount the remote directories when the system starts up. If the system cannot mount the directories at that time, it tries every three minutes until it is successful. Because this process runs in the background, she may experience a delay between the time that she can log on to her system and the time that the remote directories are mounted.

Set-user-id Programs: Ellen runs the following command on Gary's system to determine what set-user-ID and set-group-ID programs are available:

```
$ find /usr/bin \( -perm -4000 -o -perm -2000 \) -print
/usr/bin/at
/usr/bin/connect
/usr/bin/crontab
/usr/bin/format
/usr/bin/ipcrm
/usr/bin/ipcs
```

```
/usr/bin/dumpfmt
/usr/bin/errpt
/usr/bin/trace
/usr/bin/trcrpt
/usr/bin/timex
/usr/bin/netmail
/usr/bin/netstat
/usr/bin/ping
/usr/bin/tcom
```

She compares this list with what she already has installed in her **/usr/bin** directory. She finds that any programs that she may need are already on her system. She copies those programs from her local **/usr/bin** directory to her local **/bin** directory. Because her **Path** variable designates that **/bin** is searched before **/usr/bin**, the system executes the local copy of the program before it finds the remote copy.

Programs Needed When the Server is Down: Ellen decides that the only program she needs in case the server system is not operating is an editor program. She installs the **vi** editor from the Extended Operating System diskettes onto her system to allow for that circumstance.

Scenario Two: A Text Processing Environment

Bill owns a company that provides documentation and text processing services. His company has three RT PCs as shown in Figure 3-20 on page 3-108.

Bill employs four technical writers and a production assistant. The four technical writers share two RT PC Model 10s and Bill and the production assistant share an RT PC Model 25 that has been customized with 400M bytes of IBM 9332 DASD. All three RT PCs have 4M bytes of memory, are connected via a Baseband (Ethernet) LAN, and have installed word processing programs, SNA Services, and Distributed Services.

Bill plans to share the disk space installed on his RT PC Model 25 with all RT PCs on the network. The home directories of all users will reside on this shared disk. In addition, all users can access the Extended Services, Multi-User Services, and INed programs installed at this node.

Customizing the Server

Bill serves as both network administrator and system administrator for the three systems on the LAN. He has just installed the software on the three systems, but hasn't added any users yet.

Identifying Requirements: Bill wants to create a ***single system*** environment among the nodes in the network. To achieve this goal, he must satisfy the following requirements:

1. The three nodes on the LAN appear to the logged-in user to be part of a single system:

   - Users have the same identity regardless of where they log in:

     - The same login name
     - The same UID
     - The same GID
     - The same concurrent group list.

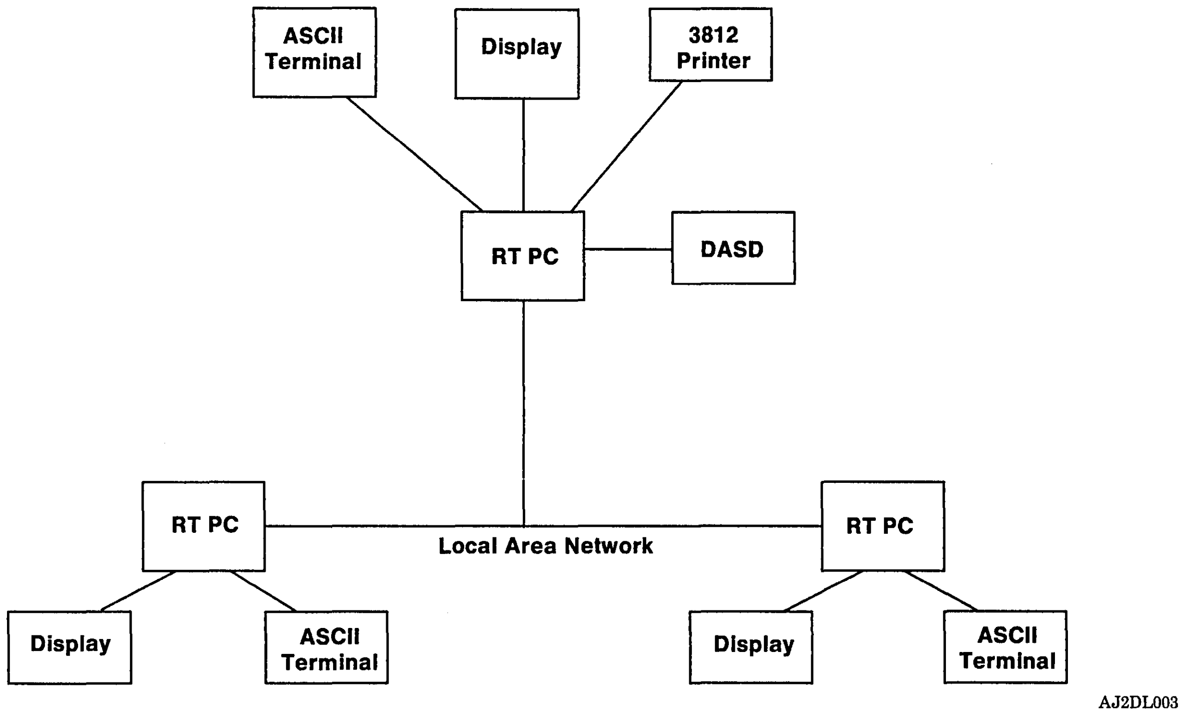


Figure 3-20. The Text Processing Environment

In addition, each local ID translates to the corresponding ID at each node in the LAN.

- Users see the same file tree, regardless of where they log in.

  All home directories reside at the server, and each client mounts the server's **/u** directory to have access to these home directories. Since both clients mount the same files and directories from the server, each has the same **/etc/filesystems** file.

- When users are added to any node, they are added to all nodes.

No matter which node runs the **users** command, it creates the home directory on the server and adds the new user to the server's passwd file.

To give all users the same identity, Bill plans to have all nodes share the following server files:

- **/etc/passwd**
- **/etc/opasswd**
- **/etc/group**
- **/etc/ogroup**
- **/etc/motd**
- **/usr/adm/user.cfile**

Since the home directories of all users will also reside at the server, Bill needs to run the **users** command only at the server, adding all network users (and groups) to the server's **/etc/passwd** (and **/etc/group**) file and creating all home directories.

2. The system initialization files issue all mounts necessary to achieve the single-system-image environment:

   - This automatic mount procedure enables each node to continue to retry remote mounts until all succeed.
   - This automatic mount procedure also makes a copy of the master **passwd** and **group** files at each node so that users can log in when the master node is not available.

3. The two client nodes rely heavily on the server for program and data resources. The following directories will be shared by all systems:

   - **/u**
   - **/lib**
   - **/usr/bin**
   - **/usr/guest**
   - **/usr/include**
   - **/usr/lib**
   - **/usr/mail**
   - **/usr/news**

Because the client systems are so dependent on the server, Bill plans to shut down the entire LAN whenever server resources are unavailable for an extended period.

Building the Node Table: Bill runs the **uname -m** command at each RT PC and makes a list of NIDs. Even though he does not plan to issue mounts from the server, he still decides to assign each node a nick name, including his own node, and to use the same nick names in all the node tables. He ends up with the following list of nodes:

Nickname	NID
Client1	108133EB
Client2	108133F5
Server	20810CBF

With this information, Bill is ready to build his Distributed Network Node Table. He uses the **ndtable** command to add each NID and its corresponding nickname (including an entry for his own node), letting the system provide the other choices by default. This produces the following node table:

Remote Nickname	Remote Node ID	Node Security	Data Link Type	Connection Profile	Attachment Profile
Client1	108133EB	None	Ethernet	108133EB	108133EB
Client2	108133F5	None	Ethernet	108133F5	108133F5
Server	20810CBF	None	Ethernet	20810CBF	20810CBF
Server	20810CBF	None	Ethernet	CDEFAULTCFD	CDEFAULT

He gives his own node a nickname because he plans, once the LAN is customized, to manage the network from the server, using remote mount queries (for example, `mount -n Client1`) and remote customization commands. By giving his own node a nick name (and using the same nick names at each node), he will see the same set of names, regardless of the mount table he is looking at.

Building the Users/Groups Table: Since each user (group) translates to the same ID everywhere on the LAN, each node needs to have the same Network Users/Groups Table. Instead of using the **ugtable** command to build the same table at each node (each table would have to be built separately), Bill decides to create an ASCII file that includes all the necessary translation entries so that he can use the **dsldxprof** command to load that file at each node. In addition, since each node uses the same **/etc/passwd** and **/etc/group** files, he decides to use a shell program to build his translation entries for him, using the **/etc/passwd** and **/etc/group** files as ID source files. The shell program that he uses is shown in Figure B-13 on page B-24.

Before he runs this shell program, he runs the **users** command to create user IDs and home directories for each of his employees. Once he has added his system users, he runs the `make.idlist` shell program. This program produces a file names `id.list`. He then runs the following command to build the server's Network Users/Groups Table:

```
# dsldxprof -f id.list
```

This produces the Network Users/Groups Table shown in Figure 3-21 on page 3-112:

Customizing the Clients: Bill has four tasks to complete at each client. He must:

1. Build the Distributed Network Node Table.

   He runs the **ndtable** command at each node, using the same NIDs and nicknames that he used at the server.

2. Build the Network Users/Groups Table.

   He copies the `id.list` file that he created at the server, and runs the same **dsldxprof** command at each client:

   ```
   # dsldxprof -f id.list
   ```

3. Implement a procedure that allows the client to function if the server goes off line.

Usr/Grp		Local	Network ID		Originating Node
Name	U/G	ID	Outbound	Inbound	Name/Nickname
system	G	0	0	0	*
root	U	0	0	0	*
staff	G	1	1	1	*
daemon	U	1	1	1	*
usr	G	100	100	*	*
guest	U	100	100	*	*
bin	G	2	2	2	*
bin	U	2	2	2	*
ctw	U	200	200	200	*
bill	U	201	201	201	*
mark	U	202	202	202	*
mel	U	203	203	203	*
geo	U	204	204	204	*
maureen	U	205	205	205	*
sys	G	3	3	3	*
sys	U	3	3	3	*
adm	G	4	4	4	*
adm	U	4	4	4	*
uucp	U	5	5	5	*
mail	G	6	6	6	*
tftpd	U	6	6	6	*

Figure 3-21. A Sample Network Users/Groups Table

Since the clients in a single-system environment are very dependent on the server for system resources, Bill uses all the procedures outlined in the discussion of managing client dependencies on page 3-66.

4. Implement an automatic startup procedure.

Bill decides to use the `ssimounts` shell program as his automatic startup program (see Figure 3-10 on page 3-81). He copies this program to the **/etc** directory as follows:

```
# cp /usr/lpp/ds/samples/ssimounts /etc
```

He then adds the following line to the end of **/etc/rc.ds**:

```
/etc/ssimounts stage1 stage2 &
```

He also adds stanzas to the **/etc/filesystems** file that identify all the mounts that ssimounts invokes. The first four stanzas define the initial set of (temporary) mounts that enable ssimounts to make backup copies of key files:

```
/etc/passwd.tmp:                        /etc/group.tmp:
        nodename = Server                       nodename = Server
        dev = /etc/passwd                       dev = /etc/group
        type = stage1                           type = stage1
/etc/opasswd.tmp:                       /etc/ogroup.tmp:
        nodename = Server                       nodename = Server
        dev = /etc/opasswd                      dev = /etc/ogroup
        type = stage1                           type = stage1
```

These files are unmounted after each one is copied to the local **/etc/passed**, **/etc/opasswd**, **/etc/group**, and **/etc/ogroup** files. (See Figure 3-10 on page 3-81.)

He also changes the stanza that defines the default mount point for the **/dev/hd1** device (**/u**) as follows.

Original stanza:

```
/u:
        dev     = /dev/hd1
        vol     = "/u"
        mount   = true
        check   = true
        free    = true
```

New **/dev/hd1** stanza:

```
/mnt:
        dev     = /dev/hd1
        vol     = "/mnt"
        mount   = true
        check   = true
        free    = true
```

The remaining stanzas define the second stage of mounts:

```
/etc/passwd:
        nodename = Server
        dev = /etc/passwd
        type = stage2

/etc/opasswd:
        nodename = Server
        dev = /etc/opasswd
        type = stage2

/etc/group:
        nodename = Server
        dev = /etc/group
        type = stage2

/etc/ogroup:
        nodename = Server
        dev = /etc/ogroup
        type = stage2

/usr/adm/user.cfile:
        nodename = Server
        dev = /usr/adm/user.cfile
        type = stage2

/etc/motd:
        nodename = Server
        dev = /etc/motd
        type = stage2

/u:
        nodename = Server
        dev = /u
        type = stage2
```

```
/lib:
        nodename = Server
        dev = /lib
        type = stage2

/usr/bin:
        nodename = Server
        dev = /usr/bin
        type = stage2

/usr/guest:
        nodename = Server
        dev = /usr/guest
        type = stage2

/usr/include:
        nodename = Server
        dev = /usr/include
        type = stage2

/usr/lib:
        nodename = Server
        dev = /usr/lib
        type = stage2

/usr/mail:
        nodename = Server
        dev = /usr/mail
        type = stage2

/usr/news:
        nodename = Server
        dev = /usr/news
        type = stage2
```

Handling System Errors

Whenever some part of your AIX system does not function properly, the system has an ***error condition***. Some error conditions are minor as when, for example, you make a typing mistake and enter a command that the system does not recognize. Other error conditions can be so severe that they cause your system to halt. The first part of this section summarizes what you should do if your system halts unexpectedly. The remainder of this section describes the AIX facilities for collecting, analyzing, and reporting system errors:

- Error logging services
- The **dump** command
- The **trace** command.

Recovering from Unexpected System Halts

Under normal conditions, before you turn off the power to the system, you use the **shutdown** command to bring all system processes to an orderly halt. Among other things, the **shutdown** command assures that all files are updated and stops any running system processes. In the event of a power failure, or if someone turns off the power to the system without running **shutdown**, your file system may have inconsistencies. You may be able to restore your system to normal operation by simply supplying power again. As part of the normal initialization procedure, the system runs the **fsck** program. If the file system sustained only minor damage, the form of **fsck** that runs automatically may be adequate to repair the damage. If not, a system message will indicate what action to take next.

The / (root) file system can become too full to use. For protection, the system moves **/etc/passwd** to **/etc/opasswd** and **/etc/group** to **/etc/ogroup**. In this state, the system either does not accept a valid password or prompts you for a password when one should not be required. In either case, you cannot use the system. To correct the problem, first start the maintenance system and verify that the names of **passwd** and **group** have been changed. Then either delete a substantial number of files from the / file system or back up the / file system, create a larger minidisk for it, and restore it. Next, move **/etc/opasswd** and **/etc/ogroup** back to their original names. Finally, start the system, obtain superuser authority, and run the **df** command to verify that the / file system is large enough (usage should be about 90 percent or less).

If a system halt is not caused by power failure or a full / file system, you should see *Problem Determination Guide* to determine its cause.

Error Logging, Analysis, and Reporting

The following components provide the AIX error logging services:

event log
Two files, **/usr/adm/ras/errfile.0** and **/usr/adm/ras/errfile.1**, of a predetermined size, which contain entries about system errors and other system events. When one file is full, the system begins to log events in the second file. When the second file is full, the system begins to overwrite data in the first file with new event entries.

error analysis facility
A program that provides information about the probable cause of errors.

errdemon
The ***error daemon process***, which collects error records from a buffer in memory, calls the error analysis facility, and writes error records, together with analysis information, to the event log. When it writes certain types of error records to the event log, the error daemon sends an alert message to the display.

error device driver
The device driver, **/dev/err**, used by **errdemon** to write error records to the event log.

errpt

A command that produces a report of logged errors from the data stored in the event log. **errpt** can produce summary or detailed reports, and you can specify what type of error records the report includes or the time span that the report covers. Generally, this is the only part of the error logging system that you work with directly.

When you start the AIX system, the error logging services start automatically. When you stop your system, the error logging services stop automatically.

To generate a report of entries in the event log, including errors, enter a command of the form:

errpt *flags filenames*

where *filenames* specifies the event log files. The **errpt** command accepts the following flags:

-s*date*	Ignores all records logged earlier than *date*. *date* must be in the form *mmddhhmmyy.*
-e*date*	Ignores all records logged later than *date.*
-a	Produces a detailed report on a single record.
-n*nodename*	Includes only error entries from *nodename* in the report.
-v*vmid*	Includes only error entries from this system name (*vmid*) in the report.
-d*list*	Limits the report to the types of error records specified in *list*. For the values that can appear in *list*, see **errpt** in *AIX Operating System Commands Reference.*

For more information about using **errpt** to analyze error data in the event log, see *Problem Determination Guide.*

There are two ways to limit the information in the error log report to that which you find useful:

- Periodically remove the error log files (you must have superuser authority):
  1. Enter `errstop` to stop the error daemon.
  2. Enter `del /usr/adm/ras/errfile.0`
  3. Enter `del /usr/adm/ras/errfile.1`
  4. Enter `/usr/lib/errdemon` to restart the error daemon.
- Narrow the error log report by selecting the **errpt** options to limit the error report generator to the type of information you find useful.

Memory Dump Services

The RT PC system has three facilities for recording its state at the time of a failure:

- Virtual Resource Manager dump
- AIX Operating System (kernel) dump
- Full image dump.

The data collected by any of these **dump** facilities is intended to help you or the person servicing your system determine the cause of the failure.

The Virtual Resource Manager Dump

If the system detects a failure, it starts the Virtual Resource Manager dump automatically. If the system, or part of the system, appears to be hung up (but a dump has not been started automatically), you can start the Virtual Resource Manager dump by pressing **Ctrl-(Left)Alt-numpad8**.

The Virtual Resource Manager dump program records its data on a diskette. You can use any formatted high-capacity diskette as a ***dump diskette***. However, since the system may not be able to format a diskette at the time you need one to hold dump data, you always should keep a dump diskette ready to use—store a dump diskette in the sleeve designed for that purpose in the back of *Problem Determination Guide.*

Besides the **dump** program, the AIX system provides a dump ***formatter***. The formatter, **dumpfmt**, processes the data collected by **dump**, producing information in a readable form. For more information about Virtual Resource Manager dump facilities, see *AIX Operating System Commands Reference* and *Problem Determination Guide.*

The AIX Operating System (kernel) Dump

If the system detects an operating system failure, it may start an operating system dump automatically. If the system, or part of the system, appears to be hung up (but a dump has not been started automatically), you can start an operating system dump by pressing **DUMP (Ctrl-(Left)Alt-End)**.

The operating system dump program writes its output to the dump minidisk on the fixed disk. To be used in problem determination, the data produced by this program must be copied to diskettes. See *Problem Determination Guide* for information about copying an operating system dump to diskettes.

Note: The **crash** command is a utility program designed to examine operating system dumps. If you are a very experienced computer user, you may find the **crash** command useful. See **crash** in *AIX Operating System Commands Reference.*

The Full Image Dump

A full image dump writes all of your system's real memory to diskettes (up to four). Generally, you should not make a full image dump unless you have been directed to do so by the person servicing your system. A full image dump takes longer to complete than a Virtual Resource Manager dump does, and you cannot use the dump formatter to inspect the data produced by a full image dump.

To be prepared to make a full image dump, you should have enough formatted diskettes available for your system:

System Memory	Number of Diskettes
2M bytes	2
3M bytes	3
4M bytes	4

trace Services

The **trace** command monitors system events while a program runs. You can use **trace** data to analyze system performance and to detect problems with programs. The **trcrpt** command processes the data collected by **trace** into a useable form and writes it into a file.

The **trace** command monitors the system events specified in a ***trace profile***. The default profile is the **/etc/trcprofile** file. To change the events that **trace** monitors, you can either modify **/etc/profile** (with an editor) or create alternate **trace** profiles.

To use **trace**, enter `trace`, then start the program with the activity you want to trace. To stop **trace**, enter `trcstop`.

Using trace to Monitor Distributed Services Activities

You can use the AIX Operating System trace facility to help determine whether the system is operating properly, or to indicate the nature of a problem that you may be experiencing on the network. The trace facility consists of the following components:

trace	The **trace** command starts a background process that logs system events in a log file.
/etc/trcfmt	This system text file defines the format of the events logged for each program that uses the trace facility.
/etc/trcprofile	A system file that you edit to select the system events to be traced.
trcstop	The **trcstop** command stops the background logging process.
trcrpt	The **trcrpt** command formats and displays the contents of the log file.

Each of the commands is described in *AIX Operating System Commands Reference.* The file **/etc/trcfmt** is useful only to programmers designing error reports. "Selecting Trace Events for Distributed Services" on page 3-122 describes the contents of **/etc/trcprofile** and how to use it to select Distributed Services events. Refer to *AIX Operating System Programming Tools and Interfaces* for a description of the structure of the trace facility and how to use it with an application program.

Selecting Trace Events for Distributed Services

To select the events that you want to trace, you must edit the file **/etc/trcprofile**. This file contains a list of all event classes that can be traced. Each event class is on a separate line in the file. If the line begins with an asterisk (*), events in that class are not logged; if the line does not begin with an asterisk, events in that class are logged. For Distributed Services, the following event classes appear in the file:

```
167     remote IPC             [DS_IPC]
166     rpc DS miscellaneous   [DS_MISC]
165     rpc DS locking         [DS_LK]
164     rpc DS open/close      [DS_OC]
163     rpc DS read/write      [DS_RW]
162     rpc server sending     [DS_SS]
161     rpc client receiving   [DS_CR]
160     rpc client sending     [DS_CS]
```

Each of these values selects a set of internal trace points. These values are defined in the following table. See "Remote Operations" on page 3-130 for information about these internal operations.

Number	Hook ID	Description
167	DS_IPC	Trace calls to remote IPC queues (msgget, msgsnd, msgrcv).
166	DS_MISC	Trace miscellaneous remote operations (null, mount, mntctl, msgctl, link, remove, getattr, access, setattr, statfs, mkdir, rmdir, rename, mknod, loadtbl, fsync, sgetattr).
165	DS_LK	Trace remote file locking operations (flock).
164	DS_OC	Trace remote file open and close operations (lookup, open, close, create, chg_sync, advise).
163	DS_RW	Trace remote file read and write operations (read, write, reada_clt, sync_ct, fclear, ftrunc).

Number	Hook ID	Description
162	DS_CS	The client node has sent a remote request to the server node. This event is logged in the trace report on the client node.
161	DS_SS	The server node has sent a response to a request from the client node. This event is logged in the trace report on the server node.
160	DS_CR	The client node has received a response from the server node. This event is logged in the trace report on the client node.

Tracing a Remote Mount Example

The following example shows how to use the trace facility to determine if the network is operating properly. It uses two systems that are connected by an Ethernet local area network and are running Distributed Services. These systems are:

Hera Machine 2061064A (Hera) is the client in this example. The client starts the remote mount operation with the command:

```
mount -n Zeus /etc /mnt
```

Zeus Machine 208131AA (Zeus) is the server in this example. The server receives the request for the mount.

1. Before starting the **trace** program, edit the file **/etc/trcprofile** on each system to remove the asterisks from column 1 of the following lines (near the bottom of the file):

   ```
   166    rpc DS miscellaneous  [DS_MISC]
   162    rpc server sending    [DS_SS]
   161    rpc client receiving  [DS_CR]
   160    rpc client sending    [DS_CS]
   ```

   Removing the asterisks from column 1 enables tracing for the four event classes (for this example, all other lines in the file begin with an asterisk).

2. Having enabled tracing for the selected functions, start the trace program on each system with the following command:

   ```
   $ trace
   ```

3. Run the **mount** command on Hera to mount the **/etc** directory from Zeus on the local **/mnt** directory.

   ```
   $ mount -n Zeus /etc /mnt
   ```

4. When the mount is complete, turn off **trace**. Enter the following command at each system:

```
$ trcstop
```

5. Create a trace report at each node. At Hera:

   ```
   $ trcrpt >Hera.rpt
   ```

   At Zeus:

   ```
   $ trcrpt >Zeus.rpt
   ```

6. Use the **pg** command to examine the report files. The actual report may contain many other entries depending upon system activity at the time the trace was run. However, for only the remote mount, the trace report at Hera might look like that shown in Figure 3-22 on page 3-126; Figure 3-23 on page 3-127 shows the report at Zeus. Figure 3-24 on page 3-128 explains the fields in the trace report.

```
                                TRACE LOG REPORT

File: /usr/adm/ras/trcfile

Wed May 13 09:42:41 1987
System: D19  1.1.2           Node: Hera
Machine: 2081064A


TIME         SEQ   PID   IODN   IOCN TYPE     HOOK     DATA
09:43:42.50 0015 00215  FFFF   FFFF DS_CS    Client_s Nid=208131AA Seq#=371
        Uid=0 Gid=0 DS_Op=1
09:43:42.52 0016 00215  FFFF   FFFF DS_CR    Client_r Seq#=371 Errno=0


Summary of event counts.
rpc client sending [DS_CS]: 1
rpc client receiving [DS_CR]: 1


Total number of events: 2
```

Figure 3-22. Example Trace Report at Hera

```
                                 TRACE LOG REPORT

File: /usr/adm/ras/trcfile

Wed May 13 09:42:14 1987
System: D19  1.1.2           Node: Zeus
Machine: 208131AA


TIME         SEQ   PID   IODN  IOCN TYPE     HOOK     DATA
09:43:00.58 0007 30002  FFFF  FFFF DS_SS    Server_s Seq#=371
        Uid=0 Gid=0 Errno=0


Summary of event counts.
rpc server sending [DS_SS]: 1


Total number of events: 1
```

Figure 3-23. Example Trace Report at Zeus

Field	Definition
TIME	This field shows the time that the trace event occurred. Time is the local machine time where the event occurred. If the two machines are not exactly synchronized (as in the example), the times do not show the correct sequence of events between the two machines.
SEQ	This field contains a sequence number given to the entry in the trace log. This number is local and does not correlate across the two systems.
PID	This field contains the process ID of the process associated with the trace event. This number is the process ID on the machine that is running that process and does not correlate across the two systems.
IODN	This field is not used.
IOCN	This field is not used.
TYPE	This field defines the type of trace event that caused the entry. The entry in this field corresponds to the line in **/etc/trcprofile** from which you removed the asterisk to enable tracing that event.
HOOK	This field contains a label that identifies the trace point in the code that generated the trace entry.

Figure 3-24 (Part 1 of 2). Trace Report Field Definitions

Field	Definition	
DATA	This field contains miscellaneous data that changes for each type of entry in the trace report. Each program that supports tracing has a unique set of data that it writes. The format of these reports is defined in **/etc/trcfmt**. For this example, the data included has the following meanings:	
	Server_s	A server send operation caused the trace entry.
	Client_s	A client send operation caused the trace entry.
	Client_r	A client receive operation caused the trace entry.
	Seq#=	This field contains the sequence number for the operation that generated the entry in the trace report. In the example, the operation was the remote mount and the sequence number is 371. The sequence number correlates all trace entries on both systems for a given operation. That is, the number 371 is common to all trace entries (on both Hera and Zeus) associated with the remote mount operation.
	UID	This field contains the user ID of the process that created the trace entry. This ID is local to the system on which the trace report is generated.
	GID	This field contains the group ID of the process that created the trace entry. This ID is local to the system on which the trace report is generated.
	Errno=	This field contains the error number associated with the attempted operation. If the error number is 0, the operation was successful. Otherwise, refer to *AIX Operating System Technical Reference* (**errno.h**) for information about the indicated error.
	Nid=	This field contains the node ID of the remote system with which the operation was attempted.
	DS_Op=	This number indicates the remote operation that is associated with this entry in the trace report. The number is a displacement into the list of remote operations defined in the format definition for this entry. The format definition is found in **/etc/trcfmt**. Refer to "Remote Operations" on page 3-130 for a list of these remote operations. In Figure 3-22 on page 3-126, the value of 1 in this field indicates a dfs_mount operation.

Figure 3-24 (Part 2 of 2). Trace Report Field Definitions

Remote Operations

When Distributed Services performs operations on one system that were requested on another system, the local system (client) sends a request on the network for an operation to be performed on the remote system (server). These requests are called ***remote procedure calls***, or ***rpc***s. These operations are performed internal to the kernel and cannot be affected by the administrator or application programmer. To help understand the information contained in the trace report, the following table lists the remote operations, their DS_Op value (displacement into the list of rpc's in **/etc/trcfmt**) and a summary of the operation being requested.

DS_Op	RPC Name	Function Requested
0	dfs_null	Used to test the connection.
1	dfs_mount	Mount a remote file or directory.
2	dfs_lookup	Get status information about a remote file.
3	dfs_open	Open a remote file.
4	dfs_close	Close a remote file.
5	dfs_create	Create a remote file.
6	dfs_read	Read from a remote file.
7	dfs_write	Write to a remote file.
8	dfs_chsync	Change the sync mode of a remote file.
9	dfs_mntctl	Execute the **mntctl** system call on the remote system.
10	dfs_msgget	Execute the **msgget** system call on the remote system.
11	dfs_msgsnd	Execute the **msgsnd** system call on the remote system.
12	dfs_msgrcv	Execute the **msgrcv** system call on the remote system.
13	dfs_msgctl	Control and status for a remote message queue.
14	dfs_link	Create a **link** to a remote file.
15	dfs_remove	Remove a link to a remote file.
16	dfs_getattr	Get the attributes of a remote file.

DS_Op	RPC Name	Function Requested
17	dfs_access	Execute the **access** system call on the remote system.
18	dfs_setattr	Set or change attributes for a remote file or directory.
19	dfs_statfs	Execute the **ustat** system call on the remote system.
20	dfs_mkdir	Create a directory in the remote file system.
21	dfs_rmdir	Remove a remote directory.
22	dfs_rename	Rename a remote file or directory.
23	reada_clt	Remote file read-ahead.
24	rbsync_clt	Remote file write-behind/
25	dfs_advise	Control of the remote directory cache.
26	dfs_mknod	Execute the **mknod** system call on the remote system.
27	dfs_flock	Lock all or part of a remote file.
28	dfs_loadtbl	Execute the **loadtbl** system call on the remote system.
29	dfs_fclear	Execute the **fclear** system call on the remote system.
30	dfs_fsync	Write all data in the remote (server) buffer to disk.
31	dfs_ftruncate	Execute the **ftruncate** system call on a remote file.

Generating a New Kernel

Customizing is the process of adapting the AIX for a particular set of requirements. When you customize a system, you define for the kernel such things as:

- Devices attached to the system (for example, printers, display stations, and fixed disks)
- Limits on system resources (for example, the maximum number of processes that can run at the same time or the maximum number of file systems that can be used at the same time).

The system is initially customized when it is installed; much of the customizing happens automatically during the installation process described in *Installing and Customizing the AIX Operating System.* You can customize some aspects of the system while it is in use (for example, with the **devices** command described in *Installing and Customizing the AIX Operating System*).

However, some system modifications require you to generate a new kernel. For example, if you modify ***system parameters*** or add new class of device drivers to your system, you must generate a new kernel. This section describes the procedure for generating a new kernel and provides general information about system parameters. It is beyond the scope of this section to discuss the potential advantages and disadvantages of modifying specific system parameters on your system.

Note: Generating a new kernel is not a routine procedure. You may never need to perform it.

Using the make Command

Two system description files, **/etc/master** and **/etc/system**, define the characteristics of your system:

/etc/master	Contains system parameters and the stanzas that describe device drivers included in the system (whether they are in use or not). You can think of this file as a model.
/etc/system	Contains entries for devices that make up the current system.

Depending upon the changes you want to make to your system, you may need to modify one or both of these files. For information about the types of changes you can make to the system description files, see "System Parameters" on page 3-135. Except for modifying the system description files, the process of generating a new kernel is automated.

To Generate a New Kernel

1. Use an editor to modify **/etc/master** or **/etc/system** as necessary.
2. Enter `cd /usr/sys`
3. Enter `make` and wait until the **make** command completes.
4. Test the new kernel. Verify changes to the new kernel before moving to the next step.
5. Enter `mv /unix` */unix.old* (where *unix.old* is a unique name for the old kernel).
6. Enter `mv unix.std /unix`

After you modify the system description file or files, change directories to **/usr/sys**. **/usr/sys** contains the **Makefile** file used by the **make** command to generate the new kernel. The **make** command, using the information in **Makefile**, generates the new kernel in two steps:

1. Uses the **/etc/config** command to translate **/etc/master** and **/etc/system** into a C language program (**conf.c**).

2. Compiles the C program (with the **cc** compiler) and combines the compiled program with standard parts of the operating system to produce the new kernel, a file named **unix.std**.

Even if you make only a minor change to a system parameter, you should first back up your file systems (see "Backing up Files and File Systems" on page 2-58) and then test the new kernel. To test the new kernel, start the standalone shell (as is explained under "Running the Maintenance System" on page 2-6) and then:

1. Save the old kernel under a different name, for example:

   ```
   mv /unix /unix.old
   ```

2. Move the new kernel to **/unix**, for example:

   ```
   mv /usr/sys/unix.std /unix
   ```

3. Shutdown the system and then restart it.

Note: Every version of the system should have a unique name. As far as is practical (given the amount of storage available on your system), you should keep old versions of the kernel until they become obsolete.

Once the new kernel is loaded, try read, write, and other operations that should indicate whether the kernel works properly. It is beyond the scope of this guide to discuss the testing and refinement of new versions of the kernel except for one general guideline: **When you generate a new kernel, thoroughly test every change you make and the general operation of the kernel.**

If the new kernel does not work, you must use the AIX Operating System Installation/Maintenance diskette to remove the new kernel and change the name of the original kernel back to **/unix**:

1. Start the standalone shell.

2. Mount the fixed disk:

   ```
   mount /dev/hd0 /mnt
   ```

3. Rename the faulty kernel:

   ```
   /mnt/bin/mv /mnt/unix /mnt/unix.bad
   ```

4. Rename the original kernel:

   ```
   /mnt/bin/mv /mnt/unix.old /mnt/unix
   ```

5. Unmount the fixed disk:

   ```
   umount /dev/hd0
   ```

6. Restart the system.

System Parameters

To a large extent, system parameters determine the system's capacity and performance. This section discusses several of the parameters that are most commonly reset to correct system resource problems. To modify any of these parameters, you first make your changes to the **/etc/master** file and then generate a new kernel (as described under "Using the make Command" on page 3-132). The values for the following parameters on your system are listed in the file **/etc/master**.

kbuffers Almost all disk I/O goes through a group of buffers. To avoid extra I/O operations, the system uses the buffers to maintain copies of the disk blocks that are being used most frequently. As the size of the buffer group decreases, the amount of disk I/O required increases, reducing the effectiveness of the buffering system. At the same time, an excessively large set of buffers may not improve the effectiveness of the buffering system, but will prevent the system from using the memory allocated to buffers for other purposes. The ***kbuffers*** parameter specifies the number of buffers used by a particular kernel.

Note: Each buffer is 2048 bytes. If you specify zero for the **kbuffers** parameter, the system uses the physical size of the installed memory to determine the number of buffers. If your system has less then 2 megabytes of memory installed, the system uses 75. If it has 2 megabytes or more of memory installed, the system uses 150. Generally, the best number to use is between 10% and 25% of the total system memory.

The number you choose may depend on how the system is being used. For example, if the system is used mostly for engineering graphics, you might choose fewer buffers; if the system is a multi-user system, you might choose more buffers.

procs The ***procs*** parameter specifies the maximum number of simultaneous processes the kernel supports. Once this limit is reached, no new processes can start. As long as the maximum number of processes is running, any attempt to start a new process causes the shell to display a message telling you to try again later.

maxprocs The ***maxprocs*** parameter specifies the maximum number of processes that can be run by any particular user. **maxprocs** controls the number of processes for each user in the same way that **procs** controls them for the kernel. The value for **maxprocs** on your system is listed in **/etc/master**.

mountab The ***mountab*** parameter specifies the maximum number of file systems that can be mounted at the same time. In general, the value of **mountab** should be one or two more than the number of file systems you expect to have mounted at the same time. Thus, if you normally mount four file systems, the value for **mountab** should be at least 5 or 6.

inodetab The system keeps copies of the i-nodes for all active files in a table in memory. The ***inodetab*** parameter sets the size of this table. An i-node is considered active if it is the i-node of an open file, if it is the current directory of any process, or if a file system is mounted on it. Once the table is full, no other files can be used, and almost all new system activity will be delayed until some processes complete, freeing up space in the i-node table.

On most systems, an allocation of five or six i-nodes for each process is adequate. If the value of **maxprocs** is 40, the value of **inodetab** usually should be about 250.

filetab The ***filetab*** parameter specifies the maximum number of files that can be open at the same time. If the limit of the file table is reached, attempts to open new files will fail until running processes close some of their files. The value of **filetab** should approximately equal the value of **inodetab**.

callouts Certain activities must be timed (that is, they require a predetermined interval between one action and another). Device drivers that use timed activities schedule them with the kernel's ***callout*** mechanism. The **callouts** parameter specifies how many entries are in a callouts table and thus, the maximum number of timed actions that can be scheduled (pending) concurrently. If the maximum number of callouts is reached, the system halts.

The callout mechanism is used most commonly by the display station device drivers. The value for **callouts** should be 25-100 percent more than the number of devices on the system.

texttab The AIX system makes it possible for all processes that are running the same program to share a single copy of the program (rather than loading a separate copy of the program into memory for each process). The ***texttab*** parameter determines the size of a text table. The text table contains one entry for each active shared text (program) segment in the system. If the text table fills up, no other text segments can be shared among processes until some processes complete and create some room in the text table.

The value of **texttab** is usually 30-50 percent of the number of processes on the system.

This section does not discuss a number of the system parameters. The **/etc/master** file contains all of the system parameters. System parameters also are documented in *AIX Operating System Technical Reference.*

Distributed Services Tuning and Problem Determination

In a distributed AIX environment, performance tuning and error handling are closely related. For example, in some situations, increasing the number of buffers in the device buffer pool may improve the server's performance in handling concurrent remote requests; an insufficient number of buffers can cause a connection to fail. This section describes several modifications that may improve performance, correct error conditions (such as connection failures, kernel panics, and hangs), or both.

Notes:

1. The default values for the parameters discussed here are adequate for typical network environments. Whether you will be able to improve performance by altering these values depends upon the unique characteristics and requirements of your Distributed Services installation.

2. This section does not discuss the SNA Services profile parameters that could have an effect on Distributed Services performance. For information about these profile parameters, see *SNA Services Guide and Reference.*

If you are trying to correct an error condition, you should check two things before you consider changing the values described in this section:

1. Make sure that the network hardware options are installed correctly.

2. Make sure that the remote node or nodes are running normally.

If Interface Program for use with TCP/IP is installed on your network, you can use the **ping** command to determine whether remote nodes are responding and, thus, whether the network hardware is working properly.

Increasing Kernel Buffers

The **kbuffers** parameter in the **/etc/master** file determines how many kernel buffers are reserved for disk I/O and the i-nodes of open files. Increasing the value of the **kbuffers** parameter can improve the speed of I/O operations, depending upon the amount of memory and disk space available. Beyond a certain point, however, increasing the value of **kbuffers** degrades performance.

The default value of **kbuffers** (**0**) reserves 100 buffers (2048 bytes each) for every M byte of memory installed. To change the value, first change the value of **kbuffers** in **/etc/master**, then generate a new kernel by following the procedure described in "Generating a New Kernel" on page 3-132.

Increasing Kernel Processes

The maximum number of concurrent, remote requests that a server can handle is limited by the maximum number of **kprocs** (internal kernel processes) that the server can run. Two conditions that may indicate an inadequate number of **kprocs** are:

- Slower performance in handling local requests
- Connection failures.

The **kprocs** parameter in **/etc/master** sets the maximum number of internal kernel processes. The default value of **kprocs** is 21. Generally, there should be two **kprocs** for each concurrent client request. Before you change the **kprocs** value, estimate the maximum number of remote requests that will run concurrently, multiply that estimate by two, and compare the result to the default (or current) value of **kprocs**.

To change the number of **kprocs** that can be run, do the following:

1. Change the value of **kprocs** in **/etc/master**.
2. Generate a new kernel (following the procedure described in "Generating a New Kernel" on page 3-132).
3. Run a **dsstate** command of the form:

```
dsstate value
```

where *value* is the value of **kprocs** in **/etc/master**; this determines the number of possible **kprocs** that can run. (See *AIX Operating System Commands Reference* for information about **dsstate**.)

Increasing Device Buffers

In a distributed AIX environment, increasing the number of device buffers may:

- Improve performance
- Increase the number of concurrent remote requests that a server can handle
- Eliminate connection failures.

To change the number of device buffers, change the values of one or more of the following network adapter parameters with the **devices** command:

nobodr Determines the number of buffers in the device ring. The default value of **nobodr** is 30.

When the number of requests to a server exceeds the capacity of the server's internal message queues, some message transmissions fail. The server continues to try to transmit those messages; consequently, its performance slows. If the server is unable to transmit the message successfully within a certain number of attempts, the transmission fails. Increasing the value of **nobodr** may restore or improve performance by reducing message retransmission.

norbosr Determines the number of receive buffers in the SLIH ring. The default value of **norbosr** is 30. The value of **norbosr** should be the same as that of the **nobodr** parameter.

nobibp Determines the number of buffers in the network device buffer pool. The default value of **nobibp** is 100. A larger value can enable a server to handle more concurrent remote requests and may eliminate some connection failures.

Use the following formula as a guide for changing the value of **nobibp**:

```
nobodr + n * (tw * 2) + 2 * n
```

where:

nobodr is the value of that device parameter.

n is the number of nodes in the network.

tw is the value of the SNA Services transmit window parameter. (For the value of tw, see the applicable SNA Services logical link profile.)

For example, if the server is on a four-node network, its **nobodr** value is 30, and its **tw** value is 16, the value for **nobibp** is 166:

```
30 + 4 * (16 * 2) + 2 * 4 = 166
```

Changing Device Descriptions

The values for the device description parameters should be set according to the configuration of your Distributed Services network. Connections fail if the values of these parameters are too small. To change one or both of the following network adapter parameters, use the **devices** command :

mnonid Sets the maximum number of node IDs that the server can handle. The default value of **mnonid** is 4. The value of **mnonid** should be equal to the number of clients on the network.

mnoal Sets the maximum number of gateways that the server can handle, including the base network. The default value of **mnoal** is 3. The value of **mnoal** should be equal to the number of network adapters installed in the server.

Tuning for Larger Networks and More Concurrent Activity

By modifying the values of the system parameters addressed in this section, you may be able to improve performance or correct error conditions on larger networks or on nodes that have unusually high concurrent file and process activity.

Note: Because each Distributed Services network is unique, there cannot be clear guidelines about what constitutes either a large network or unusually high concurrency.

The parameters covered in this section are in the **/etc/master** file. To change them, do the following:

1. Make the necessary changes in **/etc/master**.
2. Generate a new kernel (as described in "Generating a New Kernel" on page 3-132).
3. Increase the amount of paging space on the system (as described under "Increasing Paging Space" on page 3-144) to accomodate the new, larger kernel.

Parameters:

dsnkprocs	Sets the maximum number of kernel processes available for use by Distributed Services. The value of **dsnkprocs** should equal the number of concurrent connections. The default value of **dsnkprocs** is 20.
nncb	Sets the amount of memory allocated for Distributed Services translation tables. The value of **nncb** should equal the number of concurrent connections. If **nncb** is inadequate, the **dsxlate** command will fail. The default value of **nncb** is 25.
maxnode	Sets the maximum number of nodes that can be connected in the Distributed Services network. The default value of **maxnode** is 20.

filetab	Sets the maximum number of files (local and remote) that can be open simultaneously. The default value of **filetab** is 250. The values of **filetab** and **inodetab** should be approximately equal.
inodetab	Sets the maximum number of i-nodes that can be active simultaneously. The value of **inodetab** should equal the number of files (local and remote) that are open simultaneously. The default value of **inodetab** is 250. The values of **inodetab** and **filetab** should be approximately equal.
callouts	Sets the maximum number of timed events that can be scheduled concurrently. The value of **callouts** should be approximately: *maxnode* - *dsnkprocs* + 50

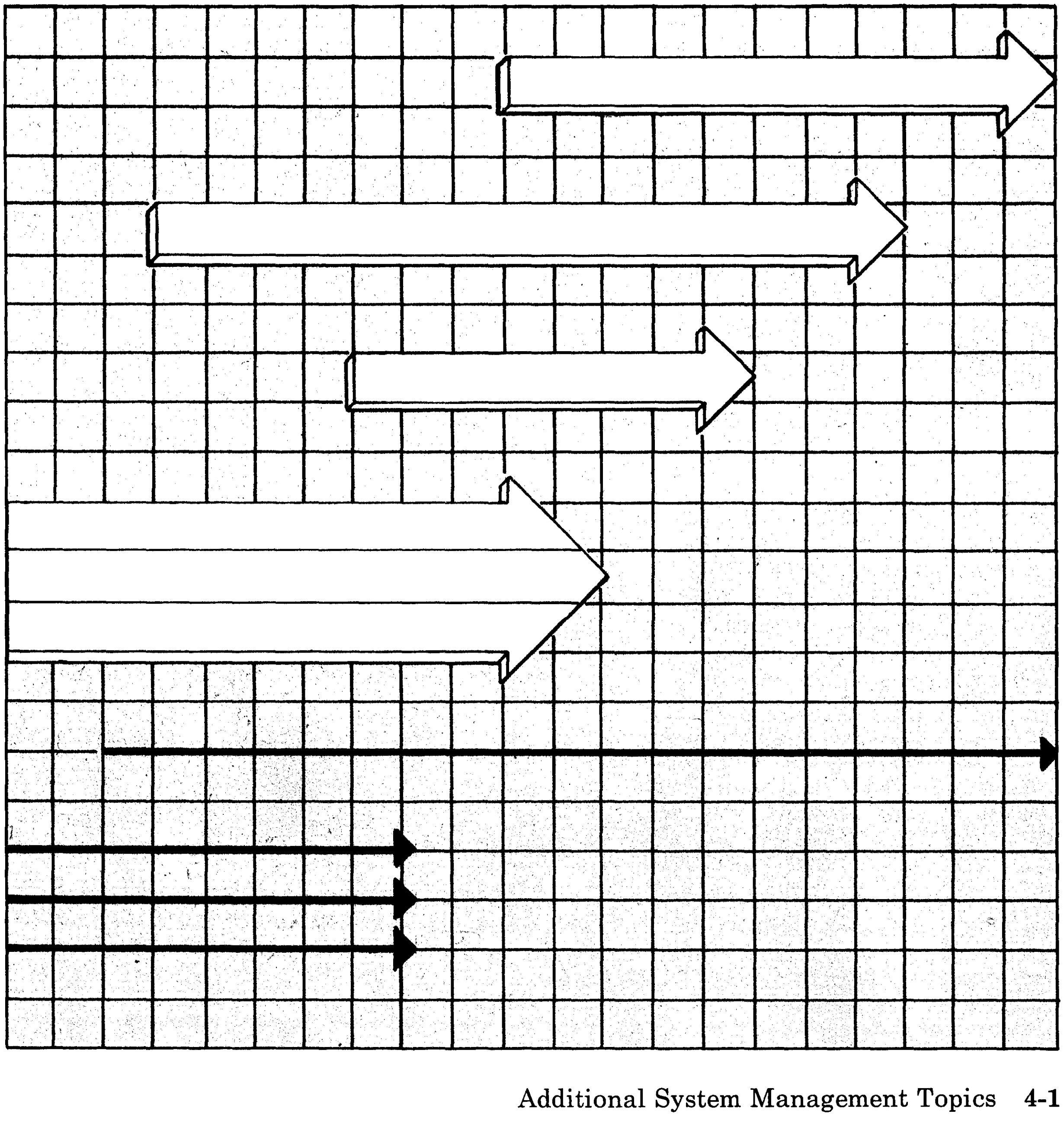
Increasing Paging Space: When you increase the values of the parameters covered in the previous section, you also increase the size of the AIX kernel (**/unix**). The larger kernel requires more paging space. Therefore, besides increasing the values of the parameters and generating a new kernel, you also must increase the size of the page space minidisk.

As a general guideline for how much to increase paging space, use the size difference between the new and old kernels. Use the **size** command to determine the size of each kernel. In the following example, the size difference between the new and old kernels is 69756 bytes:

```
size /unix
482272 + 67672 + 417752 = 967696
size /unix.old
443316 + 62136 + 392488 = 897940
```

To change the size of the page space minidisk, use the procedure described in *Installing and Customizing the AIX Operating System.* Since the VRM customizing program requires sizes specified in blocks, you must divide the size difference between the two kernels (bytes) by 512 (bytes per block). In the previous example, the size difference is 69756/512, or 136 blocks. To accommodate the new page space minidisk, you may need to put it on a different fixed disk (which you can do with the VRM customizing program) or rearrange the other minidisks (as is explained in “Handling the minidisk full Condition” on page 4-41).

Chapter 4. Additional System Management Topics



CONTENTS

About This Chapter

This section covers an assortment of topics that you may find useful in managing the AIX system. If your system supports several users, you also should acquaint yourself with the AIX facilities for accounting and monitoring system activity. Chapter 5, "Running System Accounting" on page 5-1 describes the AIX accounting facilities and how to use them. Chapter 6, "Using the System Activity Package" on page 6-1 describes the facilities for monitoring system activity.

Updating the System and Installing Local Programs

There are three ways in which you can alter the basic features of the RT PC system:

- Install additional licensed programs or components of licensed programs (using the **installp** command as is explained in *Installing and Customizing the AIX Operating System*).
- Make updates to one or more licensed programs or components of licensed programs that are already installed.
- Create your own programs.

This section contains guidelines for updating your system and installing your own programs.

Updating the System

An ***update*** is an improvement for some part of the system. An update may apply to one or more licensed programs or components of licensed programs. If IBM supplies updates for your AIX system, you will receive a diskette containing the update. You will use the **updatep** command to apply updates.

Updates for IBM provided software products on your AIX system are packaged together on the same diskette. You select the products for which you want to apply updates. After the updates are applied, you can selectively ***commit*** or ***reject*** the complete set of updates for each product based on the results of a test period. You must be a member of the **system** group to run the **updatep** command.

You start an update by running the **updatep** command, which performs four functions:

updatep -a Apply selected product updates from the diskette.

updatep -c	Commit selected product updates that have been applied.
updatep -r	Reject selected product updates that have been applied.
updatep -s	Present status about all currently applied product updates.

You must enter the appropriate action when you enter the **updatep** command.

A menu is presented if you choose one of the first three actions. If you choose **apply**, the menu contains every product on the diskette for which there is an earlier version on the system (that is, a version with a lower update level). If you choose either **commit** or **reject**, the menu contains all of the products on the system that have applied updates.

Select the appropriate products from the **apply**, **commit**, or **reject** menu. The action (**reject**, for example) is performed for all selected products.

For additional information on using **updatep**, see **updatep** in *AIX Operating System Commands Reference.*

Installing Applications and

Local Commands

If you create your own commands or programs (shell programs, for example), the following guidelines will make them convenient to use:

- **Create a directory for local commands**. If you create commands that are to be available to all system users, place them in a single directory, for example, `/u/local`. If you create commands that are to be available only to members of certain groups, create a different command directory (for example, **/u/local/system**) and set its permissions appropriately.

By keeping local commands together in one (or two) directories, you make it very simple to:

  - Remember the path name of the command.

  - Determine what local commands are on the system (by listing the contents of the local command directory).

- **Create a directory for private commands**. If you (or other system users) create commands for private use, keep them in a single subdirectory of the **$HOME** directory (for example, **/u/tom/bin**). This type of organization has the same advantages for an individual user that it does, on the larger scale, for all users of the system—it is easy to remember where the commands are and thus, to distinguish the standard system from the parts of the system that you have added or created.

Communicating with System Users

If your system has more than one user, you may find it convenient to use the system to communicate about such things as changes to the system, work schedules, unexpected system shutdowns, or new information that affects that project you are working on. The services described in this section provide several convenient means for communication among users.

Communicating with Another User—The write Command

The **write** command sends a message to another user. Often, the **write** command is used to converse with another user (that is, each user alternately sends and receives a short message).

If you do not want to be interrupted by messages, enter `mesg n` to deny message permission. Enter `mesg y` to permit messages again.

The **write** command sends a sound signal to the display station of the person receiving the message, and then displays the following message on that display station:

```
Message from username (ttynn)
[date] . . .
```

After successful connection with the other user's display station, **write** sends two sound signals to your display station.

When you try to send a message to a user who is not logged in, **write** displays the message: `user not logged in`. If you send a message to a user who has refused message permission (with the **mesg** command), **write** displays the message: `write: permission denied`.

To Send a Message with write

1. Enter:

   `write` *username*

2. After the two sound signals, enter your message.

3. At the end of your message, press:

   END OF FILE.

After you send a **write** message, the other user can respond by sending a message back to you. You may find the following conventions for carrying out such an exchange of messages useful: When you first enter the **write** *username* command, wait for the other user to send a message back before you send any text. End each message with a signal such as `o` (over) to alert the other person to reply. Use `oo` (over and out) when you complete the exchange.

For more information about **write**, see **write** in *AIX Operating System Commands Reference.*

Sending a Message to all Logged-In Users—The wall Command

The **wall** (write all users) command sends a message to all logged-in users. If you realize that you will have to shut down the system unexpectedly, the **wall** command is a good way to tell everyone who logged in to bring their work to a stopping place and log out. You must have superuser authority to use the **wall** command.

To send a message with **wall**, enter `wall` *message*:

```
# wall System shutdown at 14:30
# _
```

The **wall** command sends the message, preceded by a heading, to every user logged in to the system, for example:

```
Broadcast Message from root
System shutdown at 14:30
```

Creating a Message of the Day

To communicate with all users who will log in on a given day (not just those currently logged in), you can create a ***message of the day***. To create a message of the day, edit the **/etc/motd** file. Simply enter the text of your message into **/etc/motd** and then save the edited file. After you create a message of the day, the system sends a copy of the message to each user when the user logs in.

Creating and Reading News Items–The news Command

Use the **news** command to keep system users informed of news about the system or other topics of general interest. There are two tasks involved in using the **news** command:

- Posting news items
- Informing system users about how **news** works.

To post a news item, create a file in the **/usr/news** directory. You can either use an editor to create a file in **/usr/news** or copy a file from another directory into **/usr/news**. For example, if you create a file named `schedule` in your current directory, you can use the **cp** (copy) command to enter `schedule` as a news item:

```
cp schedule /usr/news
```

To create a news item, you must have write permission for the **/usr/news** directory. You should periodically remove all the files that contain outdated news items.

The **news** command gives you access to the items stored in **/usr/news**. To avoid reporting old news, the **news** command stores a ***currency time*** each time a user reads news items. **news** considers only the items posted after this time to be current for that user.

You can use the **news** command alone, with the name of one or more news items, or with one of three flags, depending upon the type of information you need. The following list explains the type of information returned by each version of the **news** command:

news Displays all items posted since you last read the news. To display the news items one page at a time, pipe the output of **news** to the **pg** command:

```
news | pg
```

news -a Displays all news items, regardless of the currency time. The currency time does not change.

news -n Reports the names of current news items without displaying their contents. The currency time does not change.

news -s Reports the number of current news items without displaying their names or contents. The currency time does not change.

news *item* Displays the contents of a particular *item.*

Typically, users enter the `news -n` command first. If **news -n** shows items that may be of interest, then the user can use the `news` *item* command to display the contents of the individual news items.

The most convenient way to use **news -n** is to set it up to run automatically each time a user logs in, which you can do in one of two ways:

- Add `news -n` to each user's **$HOME/.profile** file.
- Add `news -n` to the system's **/etc/profile** file.

Sending and Reading Messages—The mail Command

The **mail** command allows you to:

- Send messages or files to specific users.
- Receive and process mail sent to you.

To Send Mail

1. Enter:

   `mail` *username*

   where *username* is a list of one or more user names.

2. Enter your message.
3. Press **END OF FILE**.

You can also use **mail** to send a file to other users with a command of the form:

`mail` *username <filename*

The **mail** command prefixes each message with the sender's name and the date and time of the message (its ***postmark***). **mail** then places messages in `/usr/mail/`*username* (for example, `/usr/mail/tom`. When users read their mail and then use the **s** subcommand to save the message, **mail** saves the file in the user's mailbox—that user's **$HOME/mbox** file (unless the user specifies a different file name).

To Read Mail

1. Enter:

   ```
   mail
   ```

2. At the ? prompt following each message, press **Enter** to read the next message.

3. (Optional) To display the previous message, enter:

   - (hyphen)

4. (Optional) To delete the current message and display the next message, enter:

   ```
   d
   ```

5. (Optional) To save the message, enter:

   `s` *filename*

   If *filename* is not specified, **mail** saves the message in **/$HOME/mbox**.

6. (Optional) To forward a message to another user, enter:

   `m` *username*

7. To leave any remaining mail in the **/usr/mail/***username* file and end the **mail** program:

   Enter `q` or press **END OF FILE**.

For information about other **mail** flags and subcommands, see **mail** in *AIX Operating System Commands Reference.*

Identifying Logged-In Users—The who Command

Use the **who** command to identify all users currently logged-in to the system. Besides showing the user name of each logged in user, **who** also reports which display station each user is using and the date and time that each user logged in. In the following example, **who** reports that there are three users logged in to the system:

```
$ who
sam       console       Apr   4  09:19
pat       tty0          Apr   4  13:31
mark      tty2          Apr   4  15:04
$ _
```

For information about other ways to use the **who** command, see **who** in *AIX Operating System Commands Reference.*

Setting the System Date

The RT PC system has an internal, battery-powered clock. You set the time and date when you install the system and the clock maintains the time and date whether power to the system is on or off. Should you need to reset the time or date (for example, to synchronize system time with standard time or because of a battery failure) you can do so with the **date** command. You also can use the **date** command to find out what the current system date and time are.

To set the date and time, enter a command of the form:

date *mmddhhmm.ssyy*

where:

> *mm* designates the month.
> *dd* designates the day.
> *hh* designates the hour, using a 24-hour clock (for example, 7 p.m. is represented as 19).
> *mm* designates the minutes.
> *.ss* designates the seconds.
> *yy* designates the year.

Each time or date designation must be two digits long (for example, the date designation for July is 07). For values that do not need to be changed, type spaces (for example, if the system's current month designation is correct, but you need to change the setting for day, type two spaces and then the new date).

Warning: Do not set the date or time while other users are logged in to the system.

You must have superuser authority to set the time and date.

In the following example, the **date** command sets the date and time to April 18, 3:15.32 (32 seconds after 3:15) p.m., 1985:

```
# date 04181515.3285
#
```

To display the system's current date and time values, simply enter: `date`.

For more information about the **date** command, see **date** in *AIX Operating System Commands Reference.*

Understanding System Security

The key to effective security is understanding how the security features work and then using them conscientiously. If more than one person uses your system, it is important for everyone to understand how security works and the importance of observing certain security guidelines.

The principal AIX security features are described in other sections of this book. This section summarizes important security features and indicates where you can find more detailed information.

Passwords

Passwords help protect your system against unauthorized access. The AIX system encrypts passwords, stores them in a file, and then compares the password supplied when a user tries to log in with the encrypted version. If the two match, the user gains access to the system.

While it is not mandatory to use passwords on your system, it is strongly advised, even if you are the system's only user. Without password protection, all data on your system is available to anyone who knows how to turn on the power switch and enter a valid user name (generally, user names are easy to obtain or guess). Even if your system does not contain sensitive data, an unauthorized user cause problems by altering the contents of files or turning off the system without running **shutdown**.

Note: If user **root** does not have a password, password protection is not enforced. Anyone who knows a valid user name can log in without entering a password and use the AIX system.

For more information about passwords, the password file, and the **passwd** command, see the following:

- *Using the AIX Operating System*
- "Managing User Accounts" on page 2-16
- **passwd** in *AIX Operating System Commands Reference.*

File Protections

Every file and directory in the AIX system has a group of permissions associated with it. The permissions define who can use the file in what way. In other words, the permissions grant certain users access to the files and directories and protect the files and directories from other users.

For an explanation of how the file and directory permissions work, see *Using the AIX Operating System.* While permissions afford an effective way to control access to data stored in the system, they are only as effective as users make them. Thus, it is important for all users of your system to understand how the permissions work, how they can be modified, and how their effectiveness can be undermined by a lax attitude toward system security.

Invalid Login Attempts

Invalid login attempts due to an incorrect login name or password are recorded in the **/etc/.ilog** file. The contents of this file can be examined only by the user **root** or **su**, or a member of the system group. Each time you log in as **root** or **su** and there is an entry in **/etc/.ilog**, the system displays a message advising you to check the contents of **/etc/.ilog**. Use the **who** command to look at this file.

Running Commands at Pre-set Times

The **cron** command resembles a clock that can run shell commands at pre-set dates and times. You do not enter the **cron** command; rather, **cron** is included in the **/etc/rc** file and starts during the system initialization process. Specify what commands are to run and when they are to run, with either the **at** command or the **crontab** command:

at	Use **at** when a command is to be run only once.
crontab	Use **crontab** when a command is to be run regularly.

Using the at Command

The **at** command takes its list of commands from standard input. Standard output and standard error from the commands are mailed to the user who ran the **at** command unless they are redirected.

To Use the at Command

1. Enter:

   `at` *time date +increment*

2. Enter the commands to be run (either one command per line, or separated by ; [semicolons]).

3. Press **END OF FILE.**

Following is an explanation of each **at** command parameter and how to set it:

time Required parameter. Specify *time* in one, two, or four digits. **at** interprets one- and two-digit numbers as hours, four-digit numbers as an hour and minutes (for example, **10** means 10 o'clock, **1043** means 10:43). Though not required, you can separate the hours and minutes with a colon (**10:43**).

To indicate whether the time is a.m. or p.m., do one of the following:

- Append an `am` or `pm` suffix to *time* (for example, `10:43pm`).
- Use a 24-hour clock (for example, 10:43 p.m. is `22:43`). If you do not supply an `am` or `pm` suffix, **at** interprets the time based on a 24-hour clock.

You also can use the following special names in the **at** *time* specification:

noon
midnight
now
next

date Optional parameter. Specify *date* in one of the following ways:

- *monthname daynumber,yearnumber*. For *monthname*, use a three-letter abbreviation (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, or Dec). For *daynumber*, use a one- or two-digit number. The *yearnumber* is an optional two-digit number preceded by a comma.
- *dayname*. Specify the name of a day of the week, either fully spelled out or abbreviated to three letters.

The **at** command also recognizes the following special days:

today
tomorrow

If you do not specify *date*, **at** interprets the time to be:

`today` if *time* is later than the current time
`tomorrow` if *time* is earlier than the current time.

+*increment* Optional parameter. The +*increment* parameter, which is sometimes a more convenient way to specify time or date, is a number followed by one of these special words:

minute or **minutes**
hour or **hours**
day or **days**
week or **weeks**
month or **months**
year or **years**.

For example, to run a command at this same time tomorrow (rather than specifying the exact time) you could set up an **at** job like the one in the following example:

```
$ at now + 1 day
skulker
END OF FILE
$ _
```

Enter `at -l` to list the **at** jobs you have scheduled. Note that `at -l` lists **at** jobs by the numbers **at** assigns them. To see what commands a job contains, look at the file `/usr/spool/cron/atjobs/`*jobnumber*.

To cancel a scheduled command, use the **-r** flag:

`at -r` *jobnumber*

Using the crontab Command

The **crontab** command copies a specified file into the directory that contains all **crontab** files. The **cron** command (started by **/etc/rc** at system initialization) runs commands according to instructions in **crontab** files.

The general format of the **crontab** command is:

`crontab` *filename*

where *filename* is the name (or path name) of a file that contains information **cron** uses to schedule commands.

A **crontab** file can contain more than one entry (one entry corresponds to one command or command group). Each entry is on a separate line and consists of the following six fields, separated by spaces:

minute hour day/month month/year day/week command

Each field except *command* can contain:

- A number in the specified range

  Minute (0-59)
  Hour (0-23)
  Day of the month (1-31)
  Month of the year (1-12)
  Day of the week (0-6 for Sunday-Saturday)

- Two numbers separated by a - (hyphen) to indicate a range of values (for example, `4-15`)

- A list of numbers separated by commas, which specifies each number in the list (for example, `4, 8, 17`)

- An * (asterisk), which specifies all possible values.

cron runs the command at the specified time. If you include a % (percent sign) in the sixth field, **cron** takes everything before the % as the command and everything after it as input for the command.

Following is a sample **crontab** file, named `happy`, which you can create with an editor:

```
0 16 10-31 12 5 wall%HAPPY HOLIDAYS!
```

This **crontab** file writes a message to all logged-in users (**wall**) at 4:00 p.m. (0 minutes, 16 hours), from the 10th through the 31st (10-31) of December (12), on Fridays (5). The **wall** command takes the text following the `%` as its standard input.

Once you create `happy`, enter it into the **crontab** directory with the command: `crontab happy`.

You can modify the action of **crontab** with the following flags:

-l	Lists the contents of your **crontab** file.
-r	Removes your **crontab** file from the **crontab** directory.

Note: If you are logged in as the user **root**, you cannot run **crontab** on a file named **/usr/spool/cron/crontabs/su** unless **su** is the first entry in **/etc/passwd**. Similarly, if you are logged in as the user **su**, you cannot run **crontab** on a file named **/usr/spool/cron/crontabs/root** unless **root** is the first entry in **/etc/passwd.**

Monitoring Files and Directories that Get Larger Automatically

Even though the fixed-disk on the RT PC system holds a large amount of data, it can become crowded, even full. Certain files on the system can grow automatically, using disk space to store data that are no longer useful. Following is a list of files that you may or may not have on your system. If you do have any of these files, you should check them periodically and remove all useless data.

- Accounting files:

  /usr/adm/wtmp

  /usr/adm/pacct

  /usr/adm/acct/nite/*

- Other files:

  /usr/lib/cron/log

  /usr/spool

  $HOME/mbox

Also check the **/tmp** directory for files that you can delete. Ordinarily processes should remove their temporary files from **/tmp**, but they may not always do so.

You can use the following shell procedure to locate files in specified directories (in this case, `/usr/adm`, `/usr/lib`, and `/usr/spool`) that are larger than a certain number of blocks (in this case, 50):

```
for i in /usr/adm /usr/lib /usr/spool
do
    find $i -size +50 -exec ls -l {} \;
done
```

Finding Files and Directories

The **find** command searches the file system for all file names that match a specified ***expression*** (characteristic or group of characteristics). For example, the following **find** command lists the complete path name of all files in the file system with the file name **.profile**:

```
find / -name .profile -print
```

The / starts the **find** command at the root directory; the search continues through all subdirectories of /. The `-name` parameter specifies the name that **find** is to match. The `-print` parameter causes **find** to display the path name of each file that matches the `-name` expression.

The `-name` expression is just one of several expressions that you can use with the **find** command. For more information about **find** and its expressions, see **find** in *AIX Operating System Commands Reference.*

You can use the **find** command together with the **skulker** and **xargs** commands to periodically remove unwanted files from the file system. For more information, see **skulker** and **xargs** in *AIX Operating System Commands Reference.*

Managing Display Station Features

This section covers two topics:

- How to automatically set the characteristics of any display station on your system
- How to use the special features of the main display station.

Setting Display Station Characteristics Automatically

A display station has standard characteristics, such as the length of the lines displayed on the screen, the number of lines per screen, and whether the special command line editing functions are available. However, you can use the **stty** command to change certain display station characteristics. (For more information about **stty**, see *Using the AIX Operating System* and **stty** in *AIX Operating System Commands Reference.*)

If you want the display station characteristics set in a particular (nonstandard) way every time you log in, you may find it convenient to have the system run **stty** automatically. Depending upon your requirements, you can do this in one of two ways:

- Place the **stty** command in the **/etc/profile** file.

  Each time you start the system, the system automatically runs the commands in **/etc/profile**. If you include **stty** in **/etc/profile**, every display station on the system starts with the same characteristics.

- Place the **stty** command in your user profile file, **$HOME/.profile**.

  Each time you log in, the system automatically runs the commands in **$HOME/.profile** (the individual user's profile). Commands in **$HOME/.profile** supersede those in **/etc/profile**. Therefore, if you include the **stty** command in your user profile, the display station always has the same characteristics when you first log in.

For more information about these two profiles, see "/etc/profile and $HOME/.profile" on page 2-45.

Managing Special Features of the Main Display Station

The **/dev/hft** device driver provides the special features of the main display station. (**hft** stands for high function terminal.) Another device driver, **/dev/console**, also can run the main display station (or ***console***), but it does not provide the special features).

Note: You should open another virtual terminal with `open sh` to run commands and programs; thus, leaving **/dev/console** free to receive error messages. To ensure that error messages do not scroll off the screen, enter `stty page length 22`.

"Managing the Virtual Terminal Feature" explains how to manage virtual terminals, a special feature you can use directly. "Additional Main Display Station Features" on page 4-28 lists some other special features provided by the **/dev/hft** device driver.

Managing the Virtual Terminal Feature

As explained in *Using the AIX Operating System*, a virtual terminal is one of several equivalents of a display station, one or more of which can be available at the main display station concurrently. The **actman** (activity manager) command monitors virtual terminal activity.

To Enable actman

1. Enter:

   ```
   users
   ```

2. Use the **c** (change) subcommand to place `/bin/actman` in the `program` field for the appropriate user.

3. End **users**.

The **actman** command creates the initial shell and then monitors the number of open virtual terminals until all are closed. If you try to end the initial shell before all virtual terminals are closed, **actman** restarts the initial shell.

If you do not have virtual terminal capabilities (that is, you are not logged in at the main display station), the system replaces **actman** with the initial shell program.

Additional Main Display Station Features

In addition to virtual terminal capability, the device driver for the main RT PC display station provides several other special features:

- Sound control
- Keyboard control
- Locator (mouse) control
- Access to an extended character set
- Access to multiple fonts (character styles)

You generally cannot use the extended character set and multiple fonts directly. That is, there is no command or key to turn them on and off. Rather, these features are available, through system calls, to programs that require them. For more technical information about the **hft** device driver and the system calls associated with it, see *AIX Operating System Technical Reference.*

Controlling Sound: The **sound** command controls the volume of the sound output (the console bell and the keyboard click). You can set these two sound specifications independently of each other. There are two sets of flags for the **sound** command. The following flags control the volume of all sound output:

-h	Sets the volume to high.
-l	Sets the volume to low.

-m	Sets the volume to medium.
-o	Turns the volume off.

The second set of flags whether the click is produced:

-c	Turns clicking on.
-q	Turns clicking off.

The **sound** command in the following example sets the sound volume to low and turns the click function off:

```
$ sound -lq
_
```

Controlling the Keyboard: The **keyboard** command controls the ***delay rate*** and ***repetition rate*** of the keyboard. The delay rate is the interval from when you press a key to when it begins to repeat. The repetition rate is the number of times the key repeats per second. These rates are set at system startup to 500 milliseconds and 14 characters per second, respectively.

The **keyboard** has two flags:

-d*rate*	Sets the delay rate to the specified value. *rate* can be **300**, **400**, **500**, or **600**.
-r*rate*	Sets the rate of repetition to the specified value. This *rate* can be an integer from **2** to **40**.

The following **keyboard** command sets the delay to 300 milliseconds and the repetition rate to 40 characters per second:

```
keyboard -d300 -r40
```

Controlling the Locator (Mouse): The **locator** command controls the rate at which the system checks the position of the locator (mouse). You can specify any of the following rates: **10**, **20**, **40**, **60**, **80**, or **100** times per second. At system startup, the locator rate is set at **60**.

The following **locator** command sets the rate at **40** times per second:

```
locator -r40
```

Changing the Physical Display

With the **display** command, you can change the physical display assigned to the current virtual terminal and the virtual terminal characteristics or assign a default display to be used when you open a virtual terminal. You can request only displays that are actually installed on your system.

The flags to the **display** command are:

-b	Changes the background color on the display.
-c	Changes current virtual terminal display.
-d	Sets default display to be used when a virtual terminal is opened.
-f	Changes the foreground color on the display.
-m	Changes the DMA pinned page size.
-p	Change the color palette to be used.
-t	Changes the font.

The **display** command with the **-c** and **-d** flags accept the following parameters:

pcmono	PC Monochrome Adapter and Display

egamono	Enhanced Graphics Adapter and PC Monochrome Display
egacol	Enhanced Graphics Adapter and Display
advmono	Advanced Monochrome Graphics Adapter and Display
advcol	Advanced Color Graphics Adapter and Display
extmono	Extended Monochrome Graphics Adapter and Display
megapel	Megapel Display Adapter and IBM 5081 Display Model 16 or 19.

To see a menu of available options, enter **display** without flags or parameters. For example, when you enter `display -t`, you see a menu that contains a list of the available fonts. Choose one that is suitable for the work you will be doing. Not all fonts are available on all displays, nor do they all work with all programs.

The following **display** command changes the current virtual terminal display to an attached PC Monochrome Adapter and PC Monochrome Display:

```
display -c pcmono
```

You can use the **-c** and **-d** flags at the same time. The following **display** command changes the current virtual terminal display to the PC Monochrome Adapter and Display and makes the Enhanced Graphics Adapter the default display:

```
display -c pcmono -d egamono
```

For more information, see **display** in *AIX Operating System Commands Reference.*

TERM Values for Different Displays, Adapters, and Terminals

Change the value of `TERM` in the **.profile** file, and add `export TERM` to the **.profile** file.

Use the following values for `TERM`:

Display/Terminal	**Adapter**	**Value**
IBM Personal Computer Display	IBM Monochrome Display and Printer Adapter	ibm5151
IBM Personal Computer Display	IBM PC Enhanced Graphics Adapter	ibm5154
IBM Personal Computer Enhanced Color Display	IBM PC Enhanced Graphics Adapter	ibm5154
IBM RT PC Advanced Monochrome Graphics Display	IBM RT PC Advanced Monochrome Graphics Display Adapter	ibm6153
IBM Advanced Color Graphics Display	IBM Advanced Color Graphics Display Adapter	ibm6154
IBM Extended Monochrome Graphics Display	IBM Extended Monochrome Graphics Display Adapter	ibm6155
IBM 3161 ASCII Display Station	n/a	ibm3161
IBM 3163 ASCII Display Station	n/a	ibm3161
DEC VT100[1] (terminal)	n/a	vt100
DEC VT220[1] (terminal)	n/a	vt220
IBM 3161 ASCII Display Station with cartridge[2]	n/a	ibm3161-C
IBM 3162 ASCII Display Station	n/a	ibm3161

Figure 4-1 (Part 1 of 2). TERM Values for Different Displays, Adapters, and Terminals

Display/Terminal	Adapter	Value
IBM 3162 ASCII Display Station with cartridge[2]	n/a	ibm3162
IBM 5081 Display	Megapel Display Adapter	ibm3162

Figure 4-1 (Part 2 of 2). TERM Values for Different Displays, Adapters, and Terminals

1 DEC, VT100, and VT220 are registered trademarks of Digital Equipment Corporation.

2 International character support

Managing Printers

There are two main management tasks associated with printers on the RT PC system:

- Adding new printers to the system.

  For information about adding a printer that is supported by the RT PC system, consult the documentation that comes with the printer and the section of *Installing and Customizing the AIX Operating System* that discusses the **devices** command.

  For information about adding a printer that is not supported, consult the documentation that comes with the printer.

- Altering the way printers work (the subject of this section).

When you print a file, the system sends a stream of codes to the printer. Some of the codes cause the printer to print particular characters (for example, a-z, A-Z, and 0-9). Other codes cause the printer to print characters or files in a particular way (for example, underscoring certain characters or making every page 60 lines long).

If you want to send different character codes to the printer (for example, to change the word `that` to the word `this`), you simply edit your file—you do not have to understand that codes are involved. However, altering the way a printer works is somewhat more complicated—you must understand what happens when you print a file, what options you have for sending control information to the printer, and what printer characteristics you can control.

The Printing Process

You use the **print** command to send a file to a printer. Often you may do nothing more than enter a command of the form `print` *filename* and wait for the printer to complete the job.

A file does not go directly to the printer, however. First, the **qdaemon** command places the print request in a queue. The print request stays in the queue until a printer becomes available, at which point **qdaemon** runs the **piobe** (printer input/output backend) command. (For more information on queues and the **qdaemon** command, see "Parts of the Queueing System" on page 3-20.)

The **piobe** command processes the file and sends it, along with control information, to the printer. Thus, the printer receives a data stream composed of the contents of the file and the control information supplied by **piobe**.

Controlling the Printing Process

You can add printer control information to the printer data stream in the following ways:

- Include printer control codes in the file

  All printer control information that is unique to a file should be included in that file. For example, if you want to underscore the title of a book or print a paragraph in bold type, you must use an editor to insert codes into your file that cause the printer to start and stop the special treatment at the correct places. Appendix A, "Printer Control Codes" on page A-1 includes the RT PC printer control codes. Consult the documentation for your editor to learn how to use it to enter control codes into a file.

- Supply piobe flags with the print command

  The **piobe** command recognizes a number of flags that control printer operations such as:

  - Starting and stopping condensed, emphasized, double-wide, and double-strike printing
  - Printing in specified colors
  - Setting the left, right, top, and bottom margins
  - Setting the number of lines per vertical inch
  - Maintaining the horizontal position on the print line for a line feed or vertical tab control.

All of the **piobe** flags are listed and explained under **piobe** in *AIX Operating System Commands Reference.*

If you want to use particular **piobe** flags for a single print job, specify them along with the **print** command. For example, the **piobe** flag for setting form (page) length is `-fl=`*num*, where *num* is a number of lines. If the standard **piobe** setting is `-fl=66` and you want the file `printtest` to be printed with 45 lines per page, you can enter the command:

```
print -fl=45 printtest
```

The flag on the command line overrides the standard **piobe** setting for this one job. However, the standard **piobe** form length setting remains 66.

- Change the standard piobe settings

  To modify some printer characteristic for all print jobs, modify the standard **piobe** flags; **piobe** flags are contained in the **/etc/qconfig** file. (The **piobe** flags are listed under **piobe** in *AIX Operating System Commands Reference.*)

  For example, if you want the standard page length for all print jobs to be 45 lines, you can use an editor to modify the **/etc/qconfig** file, setting the value of **-fl** to `45`. Regardless of the page length set in **/etc/qconfig**, you always can specify a different page length for a particular print job by supplying a different value for **-fl** when you enter the **print** command.

Maintaining System Performance

System performance can be a highly specialized and complex subject, further complicated by the fact that a tactic that improves the performance of one system might actually impair the performance of a similar system used for different work. A discussion of improving system performance is beyond the scope of this book. However, there are several things you can do to maintain the performance level of your system:

- Keep directory files small.
- Reorganize file systems.
- Reconstruct minidisks.

This section covers each of these system maintenance tasks. (For a discussion of system configuration parameters that affect performance, see "Generating a New Kernel" on page 3-132.)

Keeping Directory Files Small

Directories that are larger than 10 ***logical*** blocks are very inefficient. A 10-block directory can contain over 1200 entries. (Each logical block can contain 2048 bytes of data, and each entry requires 16 bytes.) Thus, there usually is little reason to have larger directories. However, large directories (directories that occupy more than 10 blocks) can occur when, for example:

- A user has a large number of closely-related small files.
- A user does not understand how to use a directory structure to organize files.
- A directory (such as **/usr/news**) is shared by several users.
- A command or program writes its output to a new file (or files) each time it runs.

Your first tactic for keeping directory files small should be to inform all users that their directories should contain no more than a certain number of directories (a number well below the 1200 that could be possible). At the same time, you should make certain that all users understand how to use directory structures to organize files. (For information on how to use directory structures to organize files, see *Using the AIX Operating System.*)

Your second tactic should be to periodically search all file systems for directories larger than 10 blocks. If any large directories exist, the following **find** command displays a list of them:

```
find / -type d -size +10 -print
```

Simply removing files from a directory does not make that directory smaller; rather, it leaves the directory entries for the removed files vacant and ready to be assigned to other files.

To make a directory smaller, you must:

1. Make sure the directory has an acceptable number of entries. (For convenience, you might establish a 100-file directory size guideline.)

   If you need to remove files from the directory, you can use either the **rm** command (to remove files from the system) or the **mv** command (to move files to another directory).

2. Create a compacted directory or directory structure (one that does not contain the vacant entries).

   - To compact a single directory:

     a. Create a new directory with the **mkdir** command.

     b. Move (**mv**) or copy (**cp**) the files from the old directory into the new directory.

     c. Remove all files from the old directory (**rm**) and then remove the old directory (**rmdir**).

   - To compact a complete file system, use the **dcopy** command to copy the entire old file system to a new file system. For information about **dcopy**, see **dcopy** in *AIX Operating System Commands Reference.*

Reorganizing File Systems

When you create a file system, its free list is organized for maximum efficiency. However, as you create, delete, link, unlink, copy, and move files on the file system, the free list becomes less and less organized. At the same time, the physical location of data on the disk becomes less organized. The less organized the free list and data are, the longer average time it takes for the system to access files and directories in the file system. Reorganizing the free list and data of a file system often can improve system response time.

Reorganizing Only the Free List

Two flags cause the **fsck** command to reorganize free lists: **-s** and **-S**.

The **-s** flag causes **fsck** to reorganize the free list regardless of the condition of the file system. The **-S** flag causes **fsck** to reorganize the free list if **fsck** finds no inconsistencies in the file system.

If you only want to reorganize the free list, **fsck -S** is the quickest way to do so. You should run **fsck -S** regularly, perhaps once per week or twice per month, depending upon how heavily your system is used.

Note: Always run **fsck** on unmounted file system.

For more information about the **fsck** command, see "Checking and Repairing File Systems—The fsck Command" on page 3-8 and **fsck** in *AIX Operating System Commands Reference.*

Reorganizing Data and the Free List

There are two ways to reorganize both the data and the free list of a file system:

- **backup**, **mkfs**, and **restore**

  1. Back up the complete file system (with the **backup** command).

  2. Make a new file system on the minidisk that held the file system (with the **mkfs** command).

  3. Restore the backed up file system to the new file system (with the **restore** command).

  This procedure for reorganizing free lists is most appropriate when done in conjunction with routine level 0 system backups. For more information on **backup**, backup levels, and **restore**, see "Backing up Files and File Systems" on page 2-58. For more information on **mkfs**, see "Creating and Mounting File Systems" on page 2-50. Also see **backup**, **restore**, and **mkfs** in *AIX Operating System Commands Reference.*

- **dcopy**

  Besides compacting directories (as is discussed under "Keeping Directory Files Small" on page 4-37), the **dcopy** command also reorganizes a file system's free list. The main purpose of **dcopy** is to reorganize the file system for faster access times. It is also useful for copying an existing file system into a new file system (for example, when you move a file system from one minidisk to another, perhaps to balance the load among several fixed-disks). In such cases, the effect that **dcopy** has on directories and free lists is of secondary importance.

  However, if you want to use **dcopy** to reorganize the free list and compact directories, but do not want to move the file system, first use **dcopy** to copy the file system to an available minidisk, and then use either **dcopy** or **dd** to copy the file system back to its original minidisk. (For more information on the **dd** command, see **dd** in *AIX Operating System Commands Reference.*)

Handling the minidisk full Condition

Your system consists of a certain number of separate file systems, each of which resides on its own minidisk. At the very least, your system has four minidisks for the operating system, one for the VRM, and one for page space. The exact number of minidisks on your system depends upon the options you have installed and the number of additional minidisks you have created.

The size of each minidisk is set when when the minidisk is created. A minidisk that is too small for its purpose can significantly reduce your system's performance. A minidisk can be too small either because it was not made large enough initially (the problem shows up immediately) or because it has become full with use (the problem may appear gradually). If you suspect that one or more minidisks are becoming too full, enter the **df** command to determine, for each mounted file system:

- Total number of disk blocks
- Number of disk blocks free
- Number of disk blocks used.

Note: The **df** command (without arguments) only reports information about file systems with stanzas in **/etc/filesystems** that contain the line `df=true`.

In some cases, you may be able to correct a minidisk size problem by removing files from the file system. Often, however, you will need to reconstruct one or more minidisks, increasing their size as necessary. The remainder of this section describes how to recover from a `minidisk full` condition, whether it results from the installation of a program or from free space being used up over time.

Note: If you are not familiar with the **minidisks** command, please see the information on creating minidisks in *IBM RT PC Installing* *and Customizing the AIX Operating System* before you continue with this section.

To Recover from a minidisk full Condition

1. Use the **minidisks** command to determine the amount and location of existing free space.
2. Determine the amount of free space required.
3. Determine whether sufficient free space exists to meet your requirements.
4. Follow an appropriate recovery procedure.

If you are installing a licensed program, the documentation supplied with the program should state how many blocks the program requires and which file systems (minidisks) must contain those blocks. Thus, you know how much free space such a program requires. However, if a minidisk has simply filled up over time, it is not so easy to determine how much of the available free space you will allocate to it; you will have to make this decision based on your overall understanding of how your system is used.

If there is not sufficient free space to meet your requirements, you have two alternatives when faced with the minidisk full condition:

- Remove a sufficient number of existing files to create free space.
- Add additional storage to your system (that is, install another fixed disk).
- Replace an existing fixed disk with a larger one.

If there is sufficient free space available to meet your requirements, you can use one of the following four procedures to recover from a minidisk full condition:

- Expand a subdirectory into a file system.

- Create and mount a new licensed program file system.
- Expand a file system.
- Reconstruct existing minidisks.

Expanding a Subdirectory into a File System

This procedure involves creating a new minidisk, moving all data from an existing subdirectory into its own file system on the new minidisk, and then mounting the new minidisk onto the original file system subdirectory.

Note: For this procedure to be successful, the subdirectory to be expanded must have no mounted subdirectories and there must be sufficient contiguous free space for a copy of the complete subdirectory plus the new data.

Following are the steps for expanding a subdirectory into a file system:

1. Use the **add** subcommand of **minidisks** to create a new minidisk large enough to hold the old subdirectory plus the new requirements. For the mount directory, specify the original subdirectory name.
2. Mount the new minidisk on a temporary directory (for example, **mount /dev/hd***n* **/tmp/lpp**).
3. Copy the original subdirectory to the new file system. (To copy everything under the current subdirectory to the new file system, you can use the command: **find . -print | cpio -pdvm** *dirname*.)
4. Compare the original subdirectory to the new one (use the **dircmp** command).
5. Unmount the new file system from its temporary directory (for example, **umount /tmp/lpp**).
6. Use **fsck** to check the consistency of the new file system.

7. Delete files from the original subdirectory. (Enter **pwd** to verify that the original subdirectory is your current directory and then enter **rm -r *** to delete all files under that subdirectory.)

8. Use the **cd** command to move to the parent directory.

9. Mount the new file system onto the original subdirectory.

Creating and Mounting a New licensed program File System

This procedure involves creating a minidisk large enough for the new requirements, mounting that minidisk onto a new subdirectory of an existing file system, and installing the new files onto the new file system.

Note: For this procedure to be successful, there must be enough contiguous free space on the fixed disk to meet the new requirements.

Following are the steps for creating and mounting a new licensed program file system:

1. Use the **add** subcommand of **minidisks** to create a new minidisk large enough to meet the requirements of the new licensed program and specify the minidisk mount directory (for example, **/usr/lpp/***lppname*).

2. Create the minidisk mount directory (for example, **mkdir /usr/lpp/***lppname*).

3. Mount the new file system onto the minidisk mount directory (for example, **mount /usr/lpp/***lppname*).

Note: If you create a separate minidisk for each licensed program, you could eventually reach the maximum number of minidisks (28) and perhaps experience problems due to fragmentation of disk space. Therefore, it is good practice to merge minidisks periodically, if possible.

Expanding a File System

This procedure involves transferring all data from a mounted file system to a file system on a new, larger minidisk. There must be enough contiguous free space to hold the old file system and the new data. Following are the steps for expanding a file system:

1. Start the maintenance system from the AIX Operating System Installation/Maintenance diskette. (For information on starting and using the maintenance system, see "Running the Maintenance System" on page 2-6.)

2. Select `Use Maintenance Commands`.

3. Select `Backup commands`.

4. Select `Backup a file system` and back up the file system that you are expanding.

   Note: Do not use the `Backup a minidisk image` option.

5. Select `Delete a fixed disk minidisk` and write down the `Minidisk IODN` for the minidisk to be deleted—this information is required to complete the procedure. Delete the minidisk.

6. Select `Create a fixed disk minidisk`:

   a. Select `Specify IODN` and supply the IODN that you noted in the previous step.

   b. Select `No preference` to cause the minidisk to be created in the first disk area large enough to accommodate it. (Do not use the `beginning, middle, end` allocation suboptions.)

   c. Specify the number of blocks necessary to meet your new requirements.

7. Select `Make a file system` to create a file system on the new minidisk. Select the same IODN that the original file system had, the one you noted above.

8. Select `Restore commands`.

   a. Select `Restore a file system`

   b. Restore the old file system to the new minidisk, using the same IODN.

9. Select `Check a file system` and check the structure of the new file system.

10. Remove the AIX Operating System Installation/Maintenance diskette and start the system from the fixed disk.

Rearranging Existing Minidisks

With this procedure, you can physically rearrange the minidisks on the fixed disk(s). Use this procedure if:

- There is enough total free space to meet your requirements and

- You cannot use any of the previous procedures because the free space is not contiguous.

Following are the steps for rearranging existing minidisks:

1. Start the maintenance system from the AIX Operating System Installation/Maintenance diskette. (For information on starting and using the maintenance system, see "Running the Maintenance System" on page 2-6.)

2. Select `Use Maintenance Commands`.

3. Select `Backup commands`.

4. Select `Backup a file system` and back up the file systems that you are expanding. (If you want to arrange minidisks so that all of the free space is in one area, backup all minidisks between the first and last areas of free space on a particular disk.)

   Note: Use the `Backup a minidisk image` option only for minidisks that are not AIX file systems.

5. Select `Delete a fixed disk minidisk` and write down the `Minidisk IODN` and number of blocks for each minidisk to be deleted—this information is required to complete the procedure. Delete the minidisk.

6. Select `Create a fixed disk minidisk` to create a new minidisk to replace each of the minidisks that you deleted:

   a. Select `Specify IODN` and supply the IODN that you noted in the previous step.

   b. Select `No preference` to cause the minidisk to be created in the first disk area large enough to accommodate it. (Do not use the `beginning`, `middle`, `end` allocation suboptions.)

   c. Specify the number of blocks that you noted in the previous step.

   d. Repeat these steps for each of the new minidisks.

7. Select `Make a file system` and create a file system on each new minidisk that requires one (that is, the ones that originally had file systems). Use the same IODN that the original file system had, the one you noted above.

8. Select `Restore commands`.

   a. Select `Restore a file system`

   b. Restore the old file system to the new minidisk, using the same IODN.

   c. Repeat these steps for each of the new file systems.

   Note: To restore minidisks that do not have AIX file systems (the ones that were backed up with the `Backup a minidisk image` option), select the `Restore a minidisk image` option.

9. For all minidisks that contain AIX file systems, select `Check a file system` and check the structure of the new file system.

10. Remove the AIX Operating System Installation/Maintenance diskette and start the system from the fixed disk.

Logging in Automatically

Generally, it is best to use the standard login and password system to protect your system from unauthorized access. However, if your situation does not require this security, you can modify your system so that it logs you in automatically each time you start it. To make your system log you in automatically, create a file named `/etc/autolog` that contains a valid user name (that is, a user name that is in the password file). For information about users and the password file, see "Managing User Accounts" on page 2-16.

Note: You can use the automatic log in only for the main display stations (the console).

Introduction to International Character Support

The AIX operating system provides international character support in the form of an extended character set that includes accented characters and characters and symbols not used in the English language. A configurable table establishes the character set and the sorting order for the characters. Environment variables let you select alternate time, date, and currency formats, and the characters to use for currency and other symbols. Appropriate defaults can be selected by nationality.

The following terms are defined for use with international character support:

Code page	An ordered set of up to 256 characters.
Code point	A conceptual character; a 1-byte or 2-byte value that identifies a single character of a code page. AIX translates code points to support character sets for many languages, while remaining compatible with systems and devices that only support ASCII characters.
Collating order	The sequence in which characters and character strings are sorted.
Collation table	A table that defines the collating order.
Conversion subroutine	Provides an appropriate format and strings that the system uses to translate date, time, and currency information for a selected nationality.
Equivalence class	A set of characters that are considered equal for purposes of collation. For example, U.S. dictionary collation sequences place an uppercase character in the same equivalence class as its lowercase form, but ASCII collation sorts uppercase and lowercase alphabetical

	characters into different equivalence classes, because the sort is by the ordinal position of the character. Many collation sequences will not distinguish between accented and unaccented character forms for the purpose of collation, but some will.
Extended character	A character other than a 7-bit ASCII character. An extended character can be a 1-byte character with the high-order bit set or a 2-byte character.
Flattened character	An extended character translated to an 7-bit ASCII equivalent of similar appearance. For example, the flattened form of è is e.

Features

The international character support provides these features:

Extended characters
Extended characters can be used to support various languages and dialects. Extended characters are supported throughout the system in files, names of files and user, group, and queue names. A few programs do not support extended characters, such as the DOS services shell, **vi**, and **nroff**.

When a password is defined, the system determines that it is unique, and that its flattened ASCII equivalent is also unique.

Character translation
The **dd** command and various conversion subroutines translate between extended characters and ASCII escape strings that preserve unique character information.

Extended characters can also be converted to ASCII characters of a similar appearance.

Regular expression support
Regular expression support is provided for ASCII and extended characters. Metacharacters retain their traditional use, but now support extended characters.

Configurable Collation
A collating table defines an alphabetical (sort) order, designates character case and sets up equivalence classes. You can alter the collation order, case, and equivalence classes to support a language, dialect, site or user.

String formats
Utilities that output time and date information use a format selected from the environment. The **at** utility accepts input dates in the current format as well.

Numeric symbols
The currency symbol, decimal point character, and thousands-divider character can be set for the site. An example figure using the (U.S.) default is `$9,999.99`.

Configuration

The environment for international character support is established using local environment variables from the user profile. If a variable is not established as a local variable, an **NLFILE** variable is used. If a variable is not defined in the local environment or in **NLFILE**, default values are used. The **NLgetctab** function loads a collation table automatically at system initialization. To change environment settings from an application, **NLgetfile** subroutine runs the **NLgetctab** subroutine. To select an initialization environment suitable for your site, set the environment variable **NLLANG** to the an appropriate language string, set the NLFILE environment to a suitable environment file and the **NLCTAB** environment variable to the appropriate collation table file name.

A number of environment files are established in directory **/usr/lib/nls**; environment files have an **.en** suffix.

You can revise the collation table file using the **ctab** command, to change the character set, collation sequence, or equivalences values. Collation table files are found in **/usr/lib/nls** directory, and have a **.ctab** suffix. The **ctab** output files in the same directory have no suffix. For example, the output file corresponding to **french.ctab** is **french**.

Code Point Support

The ASCII character set is represented as code points of code page P0, which is a code page composed of 1-byte characters. The 7-bit ASCII characters are lower code page P0. Upper code page P0 is composed of 1-byte extended characters, with the high-order bit set high. Included in code page P0 are ASCII, accented, and other characters that support many languages.

The 1-byte character set is extended by using four values (0x1c, 0x1d, 0x1e, and 0x1f) as single-shift bytes that, together with the next byte in the data stream, select a character from code page P1 or P2. 2-byte characters are handled by an **NLchar** data type. Each **NLchar** occupies two bytes of storage. The high order bit of the **NLchar** is not used in representing a character. An extended character cannot be mistaken for an ASCII character because the high bit is set.

Bytes	**Ordinal (NLchar Value)**	**Ordinal Decimal**	**Code Page**
0xxxxxxx	000xxxxxxx	0-127	P0 low
1xxxxxxx	001xxxxxxx	128-255	P0 high
00011111 1xxxxxxx	010xxxxxxx	256-383	P1 low

Figure 4-2 (Part 1 of 2). Code Points

Bytes	Ordinal (NLchar Value)	Ordinal Decimal	Code Page
00011110 1xxxxxxx	011xxxxxxx	384-511	P1 high
00011101 1xxxxxxx	100xxxxxxx	512-639	P2 low
00011100 1xxxxxxx	101xxxxxxx	640-767	P2 high

Figure 4-2 (Part 2 of 2). Code Points

Any character in any code page can be entered from the keyboard. Techniques for entering characters that are not supported by the keyboard map are fully described in *Keyboard Description and Character Reference*, Appendix C.

Limits

Some character-handling commands do not support extended characters, such as the editing commands **edit**, **vi**, and **ex**. The enhanced edit mode does not work with multi-byte characters.

Warning: Do not use **edit**, **ex** or **vi** to examine or edit a non-ASCII file.

File and directory names have a size limit of 14 bytes. Because code points can represent either 1 or 2 bytes of storage, the number of significant code points in a file name or directory name can be limited to as few as 7. The kernel checks for a final byte consisting of a single-shift character and replaces that character with a null character.

Intersystems Compatibility

The AIX operating system is compatible with environments that do not support extended characters, providing work station, intersystem mail, networking, and program development support.

An ASCII-character synonym is automatically defined for a login name containing extended characters so that ASCII synonyms do not have to be explicitly maintained in the **/etc/passwd** file. The **adduser** utility checks both the login name containing extended characters and its ASCII equivalent against all existing full IDs, their ASCII equivalents, and synonyms before permitting the login name to be added to the system. Subroutines can convert extended characters to ASCII characters. This provides support for communications to ASCII systems that do not have international character support.

AIX can communicate with systems that use standard ASCII work stations. Site names must conform to the requirements of the host system communication protocol (**SNA**, **uucp**, or other), and must be composed of ASCII characters. Therefore, site names are limited to ASCII characters or some subset of ASCII characters. See the *AIX Operating System Communications Guide* for further information.

Environment

To select an environment for international character support, you must provide appropriate environment variables to control date and time strings and to select the collation table. The collation table must be set up appropriately for international character support. For most sites, seting up the environment requires creating an appropriate **NLFILE** value for the collation table file without changing the default collation order or equivalences, and selecting strings and values for time and date conversion. You should set an appropriate **TZ** value to establish time zone information and daylight savings time change points on appropriate Julian dates for the system.

Time and Date Strings

You can set the string variables for indicating the day of the week, month of the year, and other time strings, such as substitution strings for *yesterday*, *today*, and *noon*. You can set other environment variables as well, for example, the local time value (plus or minus GMT) and a 12-hour or 24-hour time display format.

You can set the system environment with the environment variables described on page 4-57 in the file **/etc/profile**. An environment for an individual can differ from the system environment, and is stored in the **$HOME/.profile** (from the password file) for the user. These variables can also be changed from the command line or by subroutines.

Environment variables for international character support are specified in the process environment using ordinary shell environment variables or in the text file with a path name that is specified by the shell environment variable **NLFILE**. Values specified in the process environment take precedence over values specified in **NLFILE**. If a given environment variable is not set either in the process environment or in **NLFILE**, or if a specified value is the null string, a default value (suitable for U.S. English) is used.

Note: If different international character support environments are in use on the same system, particularly different collation tables or time settings, confusion can result.

The form of environment variables is *variable=value(s)*. Environmental variables that establish the local environment vocabulary consist of a sequence of strings separated by colons. The actual language is identified by the value of **NLLANG**. Each string must be a translation of the U.S. English name or symbol used in the defaults, in exactly the same order. You can set the following environment variables for international character support:

NLLANG
Is a string that represents the international character support environment name.

NLFILE
Is the path name of a file containing other environment variable definitions for international character support. You cannot define **NLFILE** within a file that is identified by another **NLFILE** definition. There is no default path name.

TZ
Provides time zone strings, the difference in hours from Greenwich Mean Time (GMT), and times for the site to switch to and from daylight savings time. It has eight parameters in five fields in the form:

```
lclndst:bgn:end:chgwd:chghr:chgamt
```

The first field contains the following parameters:

lcl	is the standard local time zone abbreviation.
n	is the difference in local time from GMT in hours (a number from -12 to 12).
dst	is the abbreviation for the local daylight savings time zone, if any.

The remaining fields contain the following:

bgn	is the beginning day (Julian) of daylight savings time, if any.
end	is the ending day (Julian) of daylight savings time, if any.
chgwd	is the weekday of the change to daylight savings time, if any (a numeric value).
chghr	is the hour of the change to daylight savings time, if any (using a 24-hour clock).
chgamt	is the amount, in hours, of the change to daylight savings time.

An example setting is:

```
TZ=EST5.5EDT:119:315:7:2:1.5
```

NLCTAB
Provides the path name of the file containing tables that define the current collating sequence, as produced by **ctab**. The default is:

```
NLCTAB=/etc/nls/ctab/default
```

NLLANG
Provides the language label for the set of strings, collation table, and environment formats used. The default environment label for international character support is:

```
NLLANG=u.s.english
```

NLCURSYM
Provides the currency symbol and symbol placement relative to quantity. Placement is `f` or `F` if the symbol precedes the quantity, and `l` or `L` if the symbol follows the quantity. The default is:

```
NLCURSYM=:$:L:
```

NLNUMSEP
Provides numeric triad and decimal separator symbols. (The first of the two separators is the triad separator.) The defaults are:

```
NLNUMSEP=:,:.:
```

NLLDAY
Provides strings for the full ("long") names for the days of the week. The default values are:

```
NLLDAY=Sunday:Monday:Tuesday:Wednesday:Thursday\
:Friday:Saturday
```

NLLMONTH
Provides the long name strings for the months of the year. The default values are:

```
NLLMONTH=January:February:March:April:May:June:\
July:August:September:October:November:December
```

NLSDAY
Provides short name strings of the days of the week. Names should be the same length, and of 5 or fewer characters. The defaults are:

```
NLSDAY=Sun:Mon:Tue:Wed:Thu:Fri:Sat
```

NLSMONTH
Provides short name strings of the months of the year. Names should be the same length, and of 5 or fewer characters. The default values are:

```
NLSMONTH=Jan:Feb:Mar:Apr:May:Jun:Jul:Aug:Sep:Oct:Nov:Dec
```

NLTMISC
Provides miscellaneous strings needed for input and output of date and time specifications. The default values are:

```
NLTMISC=at:each:every:on:through:am:pm
```

NLTSTRS
Provides strings for relative or informal names used for input of date and time specifications to the **remind** and **at** commands. The default informal time string values are:

```
NLTSTRS=now:yesterday:tomorrow:noon:midnight:\
next:weekdays:weekend
```

NLTUNITS
Provides singular and plural forms of strings for all names of units of time, used for input of date specifications to the **at** command. The defaults are:

```
NLTUNITS=minute:minutes:hour:hours:day:days:\
week:weeks:month:months:year:years
```

NLTIME
Specifies the format of the time. The default time format string is:

```
NLTIME=hh:mm:ss
```

NLDATE
Specifies the short form of the date. The default is:

```
NLDATE=MM/DD/YY
```

NLLDATE
Specifies the long form of the date. The default is:

```
NLLDATE=mon DD, YYYY
```

Collation Table

You can use the **ctab** command to establish collating sequence and case conversion of characters. The input and output files are stored in the conventional directory **/usr/lib/nls**. Numerous files exist to support various user environments, and you can create new ones. (See "Configuration" on page 4-52.) File names should generally reflect their contents — for example, **uk** for British characters, symbols, and lexicographical collating sequence. An example collation table is provided for you in **/usr/lib/nls/example.ctab**.

Using **ctab**, you can use a defined table file for a language without changes, modify an existing table, or build a customized table.

For each character in a collating sequence (each **NLchar**), the table input file provides the following information:

- For alphabetical characters, the corresponding uppercase or lowercase version of the given character. (You can assign case to any character.)
- Collating sequence.
- The range of an equivalence class. All the characters in this class are counted in the character range, along with this character, whenever a character is named as an end point of a character range.
- You can assign a string of characters to a character equivalence class.

The following **ctab** input conventions are used for setting up a table file:

- Input starts from a standard template file containing the entire supported character set in the general order of an ISO collating sequence. Collating sequence follows the ordering of the per-character input lines. Except for ASCII characters and the characters in all supported languages, lines in the file are commented out.
- Escape sequences (in the style of the C language); are permitted in the input table if a backslash does not form part of a valid escape sequence, it strips the following character of any special meaning it could have had.
- One line of information is present for each character explicitly named.

Each character information line in a collation table file contains four data fields:

1. ***Subject Character***
   Identifies the character to appear in the collating order at that point. The treatment of the subject character is dependent on the type of character and its use. The subject character can be:

   - A (nonprinting) control character or a space character.

- An alphabetical character (having a case equivalent character) and (optional) equivalent characters. If a string of characters is given, they are collated as a single unit.

- A character to be translated using a defined translation string. The field contains a subject character followed by a ¦ (vertical bar) and a single or multiple-character string; this is a ***translation string*** that the subject character is translated to before collation. For example, to treat the character œ as equivalent to the string oe the line will contain: œ¦oe as its first field. However, the subject character cannot be used in the string of characters into which it is to translate. This is an illegal construction:

  ```
  o¦oe
  ```

2. ***case match***
   Identifies the matching case character for the character in field 1 if field 1 is an alphabetical character. If the field 1 character is lowercase, field 2 holds its uppercase equivalent, and visa-versa. If the character in field 1 has no equivalent, this is an empty (null) field.

3. ***type identifier***
   Identifies the type of character in field 1:

   - If the subject character is to be treated as an alphabetic character (even if field 2 is null) this field must identify the character as lowercase or uppercase. An **l** or **L** placed in the type identifier field selects a lowercase subject character (or characters). A **U** or **u** in this field selects an uppercase character (or characters).
   - If the subject character is a (nonprinting) control character, a value of **C** or **c** must be placed in this field.
   - If the subject character is a space (blank or tab) character, a value of **S** or **s** must be placed in this field.

4. ***equivalence class***
   Specifies the first character in the equivalence class of the subject character. Members of an equivalence class must be listed in consecutive lines. If this field is not specified, the group of consecutive subject characters having blank fourth

fields are placed in the same equivalence class (only) if they are based on the same roman alphabetic character.

A line beginning with the word `option` serves to change one or more of the default conditions or metacharacters built into **ctab**. This option line contains a set of *name-value* pairs; each pair is delimited by space or tab characters. Recognized *names* are:

eclass
Turn equivalence class function on or off globally. The *value* must be `on` or `off`; the default is **on**.

sep
Use the *value* as the separator character between fields on an input line; the default is **:** (colon). Tab or space characters may surround fields and separators.

trans
Use the *value* as an indicator of translation in subject character fields; the default is **1**.

repeat
Use the *value* as a repeat indicator meaning *same as previous character* in subject character fields; the default is ^ (circumflex). A repeat character cannot be used following a translation string because the character to be translated has no inherent collating value.

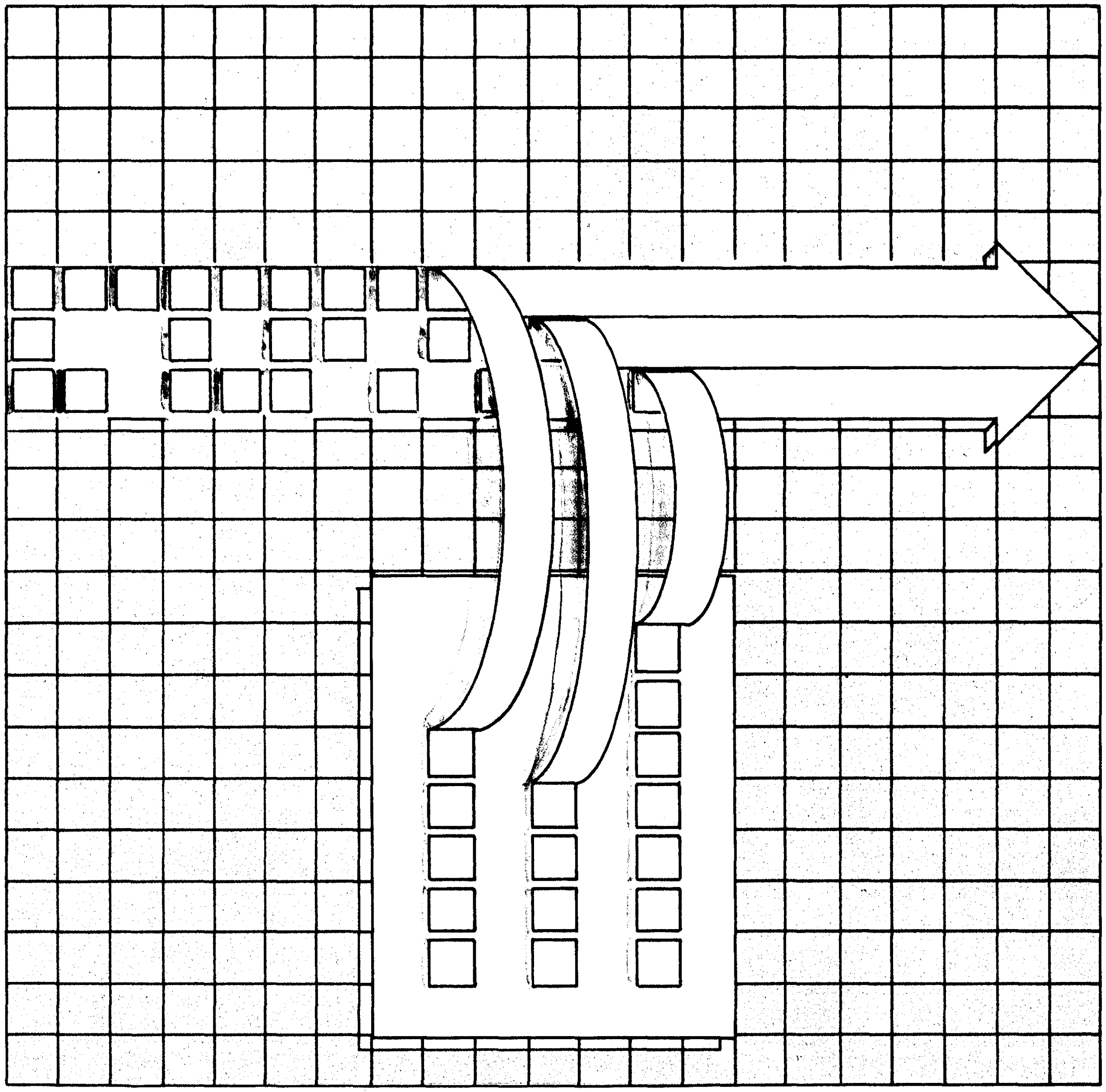
comment
Use the *value* as a comment character; the default is # (pound sign). That portion of a line to the right of a comment character is ignored.

Terminal Mapping

You can set the terminal maps for your system with the **stty** command. The **imap** parameter sets a terminal **.in** *filename* and the **omap** parameter sets a terminal **.out**. You must select the terminal **.in** and terminal **.out** names from the files in the **/etc/nls/termmap**.

You can establish terminal map assignments in the **/etc/ports** file. The **getty** utility uses this file at system initialization.

Chapter 5. Running System Accounting



CONTENTS

About This Chapter

This chapter describes the structure, setup, and management of the AIX accounting system, as well as the reports it generates.

This accounting system is designed to provide not only a way to bill for computer use but also a way to monitor various aspects of system operations.

Since you need system data combined and summarized by system users for billing purposes, and you need data combined and summarized by system resources for monitoring system operations, the accounting system collects detailed data on each transaction and provides tools for processing the data to produce different kinds of reports.

Note: The accounting system commands are part of the Multi-User Services licensed program. Before you can use the accounting system, the Multi-User Services licensed program must be installed on your system.

An Introduction to System Accounting

AIX system accounting provides for collecting and processing the following types of system data:

1. The amount of time each user spends logged into the system (***connect-time accounting***).
2. The use by each process of the processing unit, memory, and I/O resources (***process accounting***).
3. The amount of disk space occupied by each user's files.
4. The amount of printer use by each user.
5. Fees charged for materials and services.

Connect-Time Accounting

Data collection

Connect-time data are collected through cooperation between the **init** and **login** commands. When you log in, the **login** program writes a login record in the file **/etc/utmp**. This record includes your user name, the date and time of the login, and the login port. Commands such as **who** use this file to find out which users are logged in and onto which display stations. If the connect accounting file **/usr/adm/wtmp** exists, **login** adds a copy of this login record to it.

When your login program ends (normally when you log out), the **init** process records the end of the session by writing another record in **/usr/adm/wtmp**. Both the login and logout records have the form described in the header file **utmp.h**. Logout records differ from login records in that they have a blank user name.

The **acctwtmp** command also writes special entries in **/usr/adm/wtmp** having to do with system shutdown and startup.

Reports: The accounting commands concerned with keeping track of each user's share of system resources write standardized ***total accounting records***, whose format is given in the header file **acct.h**. All billable accounting data originates as, or can be converted to, either the total accounting record format or a readable and editable ASCII version thereof.

With the **acctmerg** command, you can convert between the ASCII and binary formats and merge the records from different sources into single records, one for each user.

You can use the **prtacct** command to display any total accounting file. You can also use **awk** scripts to produce site-specific reports from the ASCII representation of the total accounting records. (For a brief discussion of the **awk** command, see *AIX Operating System Commands Reference.* For a more detailed discussion, see *AIX Operating System Programming Tools and Interfaces.*)

The **acctcon1** command produces connect-time and line-use records (***session records***), from login and logout records. From these session records, you can produce reports that show each individual login session, that show a summary of login sessions for each port, or that show an overall summary of login sessions for the system.

You can use the **prctmp** command to display session records, with appropriate headings. The **acctcon2** command converts a collection of session records, sorted by user name, to total accounting records. You usually merge these records with the total accounting records produced by other parts of the accounting system to produce reports for each user.

You can also find out the last date on which each user logged in. The **lastlogin** command produces this report, normally as a part of the **runacct** procedure.

Process Accounting

Data collection

The AIX Operating System collects resource usage data for each process as it runs. The data include the user and group numbers under which the process runs, the first eight characters of the name of the command, the elapsed time and processor time used by the process, memory use, the number of characters transferred, and the number of disk blocks read or written on behalf of the process. The **accton** command causes the kernel to record this data in a specified file, usually **/usr/adm/pacct**. If called without parameters, **accton** turns off the recording of data.

Other commands relating to the recording of process accounting data are **startup**, **shutacct**, **dodisk**, **ckpacct**, and **turnacct**. These commands are simple shell files that build on the other commands discussed in this section.

Reports: In addition to providing billing information, process accounting data also provides a rich source of information that you can use to monitor the use of system resources.

The **acctcms** command summarizes resource use by command name. It provides information on how many times each command was run, how much processor time and memory it used, and how intensive that use was while the command was running (its ***hog factor***). It can also produce long-term statistics on system utilization. Thus, it provides answers to questions about total system usage, what commands are used frequently enough to be worth optimizing, and so on.

The **acctcom** command handles the same data as **acctcms**, but for a different purpose. Whereas **acctcms** summarizes the data by command, **acctcom** provides detailed information about each running of each process. With it, you can display all process accounting records or select records of particular interest. Selection criteria include the load imposed by the process, the time period during which the process ended, the name of the command, the user or group that ran the process, and the port from which the process was run.

The **acctprc1** and **acctprc2** commands produce billing information in two forms. **acctprc1** translates the user ID into a user name and writes ASCII records containing the chargeable items (***prime*** and ***nonprime CPU time***, ***mean memory size***, and ***I/O*** data). **acctprc2** transforms these records into total accounting records.

Disk-Usage Data

Most accounting information on AIX is collected continuously as the resources are consumed. Disk usage data is an exception. Rather than maintain a running record of disk space consumption, you use the **dodisk** command to collect this data periodically. **dodisk** examines file systems and summarizes space use by user name. You then use **acctmerg** to produce total accounting records from the output of **dodisk**.

The **dodisk** command charges a user name for links to files found under that user's login directory. It divides the charge for a file evenly among links to that file. The decision to charge for links rather than for files owned, is motivated by two considerations:

- The cost of a file should be borne by all who use it. The most reasonable way to determine who uses a file is to see who has links to it.
- A user can create a file, allow another user to link to it and then remove his own link to it. Once he has relinquished his access to the file, it seems unfair to charge him for it.

Printer-Usage Data

Collection of printer usage data is a cooperative effort between the **print** command that enqueues files to be printed and the queuing daemon that schedules the printing. The **print** command enqueues the user name and number of the user doing the printing, along with the name of the file to be printed and other user-specified flags. After the file has been printed, the **qdaemon** writes to a file, usually **/usr/adm/qacct**, an ASCII record containing the user name, user number, and number of pages printed. You can sort these records, convert them to total accounting records, and use **acctmerg** to combine them with data from other sources.

Fees for Services and Materials

You use the **chargefee** command to charge users for services, such as file restores or consulting, or for certain materials. **chargefee** writes an ASCII total accounting record in the file **/usr/adm/fee**.

Files and Directories

The directory **/usr/lib/acct** contains all of the C Language programs and shell procedures necessary to run AIX system accounting. The accounting system data files belong to user **adm** (currently user ID 4), and all active data files (**wtmp**, **pacct**, etc.), reside in this user's home directory, **/usr/adm**. All reports and summary files are stored in the subdirectories **nite**, **sum**, and **fiscal** (see Figure 5-1).

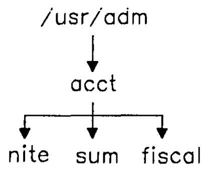


Figure 5-1. Accounting Directory Structure

The **nite** directory contains files that **runacct** reuses daily. The **sum** directory contains the cumulative summary files that **runacct** updates. Finally, the **fiscal** directory contains the summary files that **monacct** creates.

Setting Up the Accounting System

In order to automate the operation of this accounting system, you need to do several things.

To Set up System Accounting

1. Enter:`/usr/lib/acct/nulladm wtmp pacct`
2. Update the file **/usr/lib/acct/holidays**.
3. Turn on process accounting through the file **/etc/rc**.
4. In the file **/etc/filesystems**, identify file systems to include in disk accounting.
5. Specify the data file for printer usage data in the file **/etc/qconfig**.
6. Schedule daily accounting in the file **/usr/spool/cron/crontab/adm**.
7. Schedule monthly or fiscal summaries in the file **/usr/spool/cron/crontab/adm**.
8. Set the PATH shell variable in **/usr/adm/.profile** to include **/usr/lib/acct**.

Accounting Setup Procedure

1. Run the **nulladm** procedure as follows:

   a. Log in as user **root**.

   b. Enter:

   ```
   /usr/lib/acct/nulladm wtmp pacct
   ```

The **nulladm** procedure ensures that each file has the proper access permission code (664).

2. Update the file **/usr/lib/acct/holidays**, if necessary. This file contains the time table that the accounting system uses to distinguish prime time and nonprime time. Edit this table to reflect your holiday schedule for the year and to reflect the hours during each day that you want to designate as prime time.

   This file contains three types of entries:

   a. Comment lines:

      Comment lines may appear anywhere in the file as long as the first character in the line is an asterisk (*).

   b. Year Designation Line:

      This line must be the first data line (that is, the first line that is not a comment) and may appear only once. It consists of three fields of four digits each (leading blanks ignored). These three fields contain:

      1) The year

      2) The time (*hhmm*) at which prime time begins

      3) The time (*hhmm*) at which prime time ends.

      Use a 24-hour clock to specify the time. You can enter the hour of midnight as either 0000 or 2400.

      For example, to specify the year as 1984, prime time as beginning at 8:00 a.m. and nonprime time as beginning at 5:00 p.m., enter the following line:

      ```
      1984      0800      1700
      ```

   c. Company Holiday Lines:

These entries come after the year designation line. Each of these lines, composed of four fields, has the following general format:

```
Year-day  Month  Day  Description of Holiday
```

The Year-day field contains a number from 1 through 366 that indicates the day of the year on which the holiday falls (leading blanks ignored). The other three fields–Month, Day, and Description of Holiday– are treated as comments by other accounting programs. These fields identify to other users what holiday is designated by the number in the Year-Day field. Figure 5-2 shows a sample **holidays** file.

```
* Curr  Prime  Non-Prime
* Year  Start  Start
*
  1985  0830   1700
*
* Day of    Calendar    Company
* Year      Date        Holiday
*
    1       Jan 1       New Year's
    2       Jan 2       day after
   46       Feb 15      Wash. Birthday
  152       May 31      Memorial Day
  186       Jul 5       Indep. Day
  249       Sep 6       Labor Day
  332       Nov 28      Thanksgiving
  333       Nov 29      day after
  358       Dec 24      Christmas eve
```

Figure 5-2. Sample Holidays File

3. To turn on process accounting, add the following line to **/etc/rc**, if it is not already there (or delete the comment symbol if it is present):

   ```
   /bin/su - adm -c /usr/lib/acct/startup
   ```

   The **startup** shell procedure calls **acctwtmp** to record in **/usr/adm/wtmp** the time that accounting was turned on. It calls **turnacct on**, which cleans up the previous day's accounting files by calling the **remove** shell procedure.

The **remove** procedure deletes all **/usr/adm/acct/sum/wtmp***, **/usr/adm/acct/sum/pacct***, and **/usr/adm/acct/nite/lock*** files.

4. In **/etc/filesystems**, add the following line to each stanza that defines a file system you want to include in system accounting:

   ```
   account = true
   ```

5. Enable printer usage accounting by adding the following line to the queue stanza in the file **/etc/qconfig**:

   ```
   acctfile = /usr/adm/qacct
   ```

6. To automatically run daily accounting through **cron**, add the following lines to the file **/usr/spool/cron/crontab/adm**. (See the discussions of **cron** and **crontab** in *AIX Operating System Commands Reference* for details about submitting jobs to **cron**.) If these times do not fit the hours that your system is operational, adjust your entries accordingly.

   ```
   0 2 * * 4 /usr/lib/acct/dodisk
   5 * * * * /usr/lib/acct/ckpacct
   0 4 * * 1-6 /usr/lib/acct/runacct
                 2>/usr/adm/acct/nite/accterr
   ```

   The first entry schedules the running of **dodisk** for 2:00 a.m. (0 2), each Thursday (4). This second schedules **ckpacct** for 5 minutes past every hour (5 *) of every day (*). The third schedules **runacct** for 4:00 a.m. (0 4) every Monday through Saturday (1-6).

   The **dodisk** procedure calls **diskusg** and **acctdisk** to write disk usage records to the file **dacct**. It stores this file in **/usr/adm/acct/nite**.

   The **ckpacct** procedure monitors the size of **/usr/adm/pacct**. If the file is larger than 500 blocks, **ckpacct** calls **turnacct switch** to copy the current **pacct** file to **pacct***x*, where *x* is an integer that is increased each time **turnacct switch** is called. The advantage of having several smaller **pacct** files becomes apparent when you must restart **runacct** after a failure in processing these records.

The **runacct** procedure processes the active data files (including **wtmp**, **pacct**) to produce command summaries and usage summaries sorted by user name. For a detailed discussion of **runacct**, see "Running Daily Accounting—The runacct Command" on page 5-14.

7. To automatically perform monthly merging of accounting data, add the following line to the file **/usr/spool/cron/crontab/adm**:

   ```
   15 5 1 * * /usr/lib/acct monacct
   ```

   Be sure to schedule this procedure at a time (5:15 a.m. here), that allows **runacct** sufficient time to finish. This will, on the first day of each month, create monthly accounting files with the entire month's data. For a detailed discussion of **monacct** reports, see "Daily Command and Monthly Total Command Summaries" on page 5-23.

8. Set the PATH shell variable in **/usr/adm/.profile** as follows:

   ```
   PATH=/usr/lib/acct:/bin:/etc:/usr/bin::
   export PATH
   ```

   This simplifies the running of individual accounting commands when you are logged in as **adm**.

Running Daily Accounting—The runacct Command

The **runacct** command is the main daily accounting shell procedure. Normally initiated by **cron** during nonprime hours, **runacct** processes connect, fee, disk, and process accounting files. It also prepares daily and cumulative summary files for use by the **prdaily** command or for billing purposes.

Output Files

The following files produced by **runacct** are of particular interest. See Figure 5-3 on page 5-26 for a description of Accounting file formats.

/usr/adm/acct/nite/lineuse
This file contains usage statistics for each terminal line on the system. This report is especially useful for detecting bad lines. If the ratio between the number of logouts and logins exceeds about 3 to 1, there is a good possibility that a line is failing.

/usr/adm/acct/nite/daytacct
This is the total accounting file for the previous day.

/usr/adm/acct/sum/tacct
This file contains the accumulation of each day's **nite/daytacct** file and can be used for billing purposes. **monacct** restarts it each month or fiscal period.

/usr/adm/acct/sum/cms
This file contains the accumulation of each day's command summaries. **monacct** uses this binary version of the file and restarts it. The ASCII version is **nite/cms**.

/usr/adm/acct/sum/daycms
This file contains the daily command summary. An ASCII version is stored in **nite/daycms**.

/usr/adm/acct/sum/loginlog
This file contains a record of the last time each user ID was used.

/usr/adm/acct/sum/rprt*mmdd*
This file contains a copy of the daily report saved by **runacct**.

Operational States

The **runacct** procedure takes care to protect active data files. It checks frequently for operational errors. If it detects one, it writes a message to **/dev/console** (the command line in **crontab/adm** should redirect these messages to the file **/usr/adm/acct/nite/accterr**), mails messages to the users root and **adm**, removes locks, saves diagnostic files, and exits.

To make **runacct** easier to restart after an error, its operation is divided into clearly defined stages, or ***operational states***. It records its progress through these states by writing descriptive messages in the file **/usr/adm/acct/nite/active**. It also writes its current state to the file **/usr/adm/acct/nite/statefile**. The last operation in each state is to write the next state to **statefile**. After it completes each state, **runacct** reads **statefile** for the next state to process.

These states are processed as follows:

State	**Actions**
SETUP	This state calls **turnacct switch**. It moves the process accounting files **/usr/adm/pacct***?* to **/usr/adm/Spacct***?.mmdd*. It also calls **acctwtmp** to write the current time as the last record in **/usr/adm/wtmp** and then moves the file to **/usr/adm/acct/nite/wtmp**.*mmdd*.
WTMPFIX	This state calls **wtmpfix** to check **nite/wtmp** for accuracy. Some date changes cause problems with the **acctcon1** command, so **wtmpfix**

	attempts to adjust the time stamps in the **wtmp** file if a date change record appears.
CONNECT1	This state calls **acctcon1** to write session records from **wtmp** to the **ctmp** file (see Figure 5-3 on page 5-26). It also creates the **lineuse** file and the **reboots** file. Together, these three files contain all of the login and logout records found in the **wtmp** file.
CONNECT2	This state calls **acctcon2** to convert **ctmp** files to total accounting files. It sends these to **acctmerg** to produce the **ctacct**.*mmdd* file. This file contains all the connect accounting records (see Figure 5-3 on page 5-26).
PROCESS	This state calls **acctprc1** and **acctprc2** to convert the process accounting files **/usr/adm/Spacct***?.mmdd* into total accounting records in **ptacct***?.mmdd* (see Figure 5-3 on page 5-26). The **Spacct** and **ptacct** files are correlated by number so that if **runacct** fails in this state, you do not need to reprocess all **Spacct** files. **Note:** When restarting **runacct** in the PROCESS state, remove the last **ptacct** file as it will not be complete.
MERGE	This state calls **acctmerg** to merge all process accounting records with the connect accounting records, writing these to the **daytacct** file (see Figure 5-3 on page 5-26).
FEES	This state calls **acctmerg** to merge any records from the file **fee** into **daytacct**.
DISK	This state calls **acctmerg** to merge **dacct** with **daytacct**.

QUEUEACCT	This state sorts the queue (printer) usage records, converts them into total accounting records, and calls **acctmerg** to merge them with the other total accounting records in **daytacct**.
MERGETACCT	This state calls **acctmerg** to merge **daytacct** with **sum/tacct**, the cumulative total accounting file. Each day, **daytacct** is saved as **sum/tacct***mmdd*, so that **sum/tacct** can be recreated in the event it becomes damaged or lost.
CMS	This state merges the current day's command summary with the cumulative command summary file **sum/cms**, producing both ASCII and binary summary files (see Figure 5-3 on page 5-26).
USEREXIT	If the shell file **/usr/adm/siteacct** exists, this state calls it to perform site-dependant processing.
CLEANUP	This state cleans up temporary files, runs **prdaily** and saves its output in **sum/rprt***mmdd*, removes the locks, and exits.

Recovering From Failure

The **runacct** procedure can fail for a variety of reasons, most commonly because the system goes down, the file system **/usr** runs out of space, or the **wtmp** file has records with inconsistent date stamps. If **runacct** fails, do the following:

1. Check the file **/usr/adm/acct/nite/active***mmdd* first for error messages.

2. If both the **active** file and lock files exist in **acct/nite/**, check the file **accterr** (the file you redirected error messages to when **cron** called **runacct**).

3. Perform any actions needed to eliminate errors (see "Fixing Damaged Files" on page 5-19).

4. Restart **runacct** (see the following section, "Restarting runacct" on page 5-18).

Restarting runacct

If you call **runacct** with no command line parameters, it assumes that this is the first invocation of the day. You need to include the parameter *mmdd* if you are restarting **runacct**. This parameter specifies the month and day for which **runacct** is to rerun accounting. If you do not specify a *state*, **runacct** determines the entry point for processing by reading **statefile**. To override **statefile**, specify the desired *state* on the command line.

Note: When you perform the following tasks, you may need to use the full path name of **runacct** (**/usr/lib/acct/runacct**) rather than the simple command name.

1. To start **runacct**, enter:

   ```
   nohup runacct 2>/usr/adm/acct/nite/accterr &
   ```

   This entry causes **runacct** to ignore all **INTERRUPT** and **QUIT WITH DUMP** signals while it performs its processing in the background. It also redirects all standard error output (file descriptor 2) to the file **/usr/adm/acct/nite/accterr**.

2. To restart **runacct**:

   ```
   nohup runacct 0601 2>>/usr/adm/acct/nite/accterr &
   ```

   This restarts **runacct** for day of June 1 (0601). **runacct** reads the file **/usr/adm/acct/nite/statefile** to find out which state it should begin with. All standard error output is appended to the file **/usr/adm/acct/nite/accterr**.

3. To restart **runacct** at a specified state, in this case the MERGE state, enter the following:

   ```
   nohup runacct 0601 MERGE 2>>/usr/adm/acct/nite/accterr &
   ```

Fixing Damaged Files

Unfortunately, a file occasionally becomes damaged or lost. Certain files must be fixed in order to maintain the integrity of the accounting system.

Fixing wtmp Errors

The **wtmp** files seem to cause the most problems in the day-to-day operation of the accounting system. When the date is changed and the RT PC AIX Operating System is in multiuser mode, a set of date change records is written to **/usr/adm/wtmp**. The **wtmpfix** command is designed to adjust the time stamps in the **wtmp** records when a date change is encountered. However, some combinations of date changes and system restarts will slip through **wtmpfix** and cause **acctcon1** to fail. The following steps show how to patch up a **wtmp** file:

```
cd  /usr/adm/acct/nite
fwtmp  <wtmp.mmdd >wtmp.new
ed  wtmp.new
```

    (Delete damaged records or
    delete all records from beginning
    up to the date change)

```
fwtmp  -ic   <wtmp.new  >wtmp.mmdd
```

The **fwtmp** command without a flag converts a binary **utmp** file to an ASCII file that you can edit. **fwtmp** with the **-ic** flag converts the ASCII file back to the binary **utmp** format.

If the **wtmp** file is beyond repair, use the **nulladm** command to create an empty **wtmp** file. This prevents any charging of connect time.

Fixing tacct Errors

If you are using the accounting system to charge users for system resources, the integrity of **sum/tacct** is quite important. Occasionally, mysterious **tacct** records appear that contain negative numbers, duplicate user numbers, or a user number of 65,535. If this happens, first use the **prtacct** command to check the **sum/tacctprev** file. If it looks all right, the latest **sum/tacct**.*mmdd* should be patched up, then **sum/tacct** recreated. A simple patchup procedure would be:

```
cd  /usr/adm/acct/sum
acctmerg -v  <tacct.mmdd  >tacct.new
ed tacct.new
```

(Remove the bad records.
Write duplicate user number
records to another file.)

```
acctmerg -i  <tacct.new  >tacct.mmdd
acctmerg tacctprev  <tacct.mmdd >tacct
```

Remember that the **monacct** procedure removes all the **tacct**.*mmdd* files; therefore, **sum/tacct** can be recreated by merging these files together.

Accounting Reports

The **runacct** procedure produces five basic reports. They cover the areas of connect accounting, daily use of system resources by each user, command usage reported by daily and monthly totals, and a report of the last time users were logged in.

The following sections describe the reports and the meaning of their tabulated data.

Daily Report

In the first part of the **/usr/adm/acct/sum/rpt** *mmdd* report, a `from . . . to` banner identifies the period reported on. The beginning time for each report is the time the last accounting report was produced. The ending time is the time the current accounting report was produced. The banner is followed by a log of system restarts, shutdowns, power fail recoveries, and any other record written to **/usr/adm/wtmp** by **acctwtmp**.

The second part of the report breaks down line use. The `TOTAL DURATION` entry tells how long the system was in the multiuser state (that is, able to be accessed through terminal lines). The report columns are:

`LINE`	The terminal line or access port.
`MINUTES`	The total number of minutes that line was in use during the accounting period.
`PERCENT`	The total number of `MINUTES` the line was in use divided into the `TOTAL DURATION`.
`# SESS`	The number of times this port was accessed for a login session.
`# ON`	The same as `# SESS`.

`# OFF`	The total number of both logouts and interrupts that occur on a line.

During real time, you should monitor **/usr/adm/wtmp**. If it grows rapidly, run **acctcon1** to see which line is the noisiest. System performance is affected as the interrupt rate increases.

Daily Usage Report

This report, **/usr/adm/acct/nite/lineuse**, gives a breakdown of system resource use by user. Its data consists of:

`UID`	The user ID.
`LOGIN NAME`	The user name of the user. There can be more than one user name for a single user ID; this identifies which one is referred to.
`CPU (MINS)`	The amount of time that the user's process used the central processing unit. This category is broken down into `PRIME` and `NPRIME` (nonprime) use. The accounting system's definition of prime and nonprime time is taken from the **/usr/lib/acct/holidays** file.
`KCORE-MINS`	A cumulative measure of the amount of memory a process uses while running. The amount shown reflects kilobyte segments of memory used per minute. This measurement is also broken down into `PRIME` and `NPRIME` amounts.
`CONNECT (MINS)`	The amount of time that a user was logged into the system. This column is also subdivided into `PRIME` and `NPRIME` use.
`DISK BLOCKS`	The number of disk blocks used.

# OF PROCS	The number of processes that were invoked by the user. This is a good column to watch for large numbers indicating that a user may have a faulty shell procedure.
# OF SESS	The number of times the user logged onto the system.
DISK SAMPLES	The number of times the disk accounting was run to obtain the average number of DISK BLOCKS listed earlier.
FEE	The total charged against the user by the **chargefee** command.

Daily Command and Monthly Total Command Summaries

These two reports, **/usr/adm/acct/sum/cms** and **/usr/adm/acct/sum/daycms**, are virtually the same except that the Daily Command Summary only reports on the current accounting period while the Monthly Total Command Summary tells the story for the start of the fiscal period to the current date. In other words, the monthly report reflects the data accumulated since the last running of **monacct**.

These reports are sorted by TOTAL KCOREMIN, which is an arbitrary yardstick but often a good one for calculating drain on a system.

COMMAND NAME	The name of the command. Unfortunately, all shell procedures are lumped together under the name **sh** since only object modules are reported by the process accounting system. You should monitor the frequency of programs called **a.out** or **core** or any other name that does not seem quite right. **acctcom** is also a good tool to use to determine who ran a specific command and to determine if superuser authority was used.

NUMBER CMDS	The total number of invocations of this particular command.
TOTAL KCOREMIN	The total cumulative measurement of the amount of kilobyte segments of memory used by a process per minute of run time.
TOTAL CPU-MIN	The total processing time this program has accumulated.
TOTAL REAL-MIN	The total ***real-time*** (wall clock) minutes this program has accumulated. This total is the actual time that you must wait for a system prompt, as opposed to starting a process in the background.
MEAN SIZE-K	This is the mean of the TOTAL KCOREMIN over the number of invocations reflected by NUMBER CMDS.
HOG FACTOR	A relative measurement of the ratio of system availability to system utilization. It is computed by the formula: (total processor time) / (elapsed time) This gives a relative measure of the total available processor time consumed by the process during its execution.
CHARS TRNSFD	The total count of the number of characters transferred by the **read** and **write** system calls. (This total may be a negative number.)
BLOCKS READ	A total count of the physical block reads and writes that a process performed.

Last Login

This report, **/usr/adm/acct/sum/loginlog,** simply gives the date when a particular user name was last used. This could be a good source for finding likely candidates for the archives or for getting rid of unused logins and login directories.

Accounting File Formats

Figure 5-3 provides a description of the formats of the major accounting data and summary data files.

File Name	Record Format	Contents
wtmp	utmp.h	Login records: 1. user number 2. user name 3. device name 4. process ID 5. entry type 6. exit status 7. date / time
ctmp	ASCII	Connect records: 1. user number 2. user name 3. device name 4. prime connect time 5. nonprime connect time 6. session starting time 7. starting date
pacct* **Spacct***	acct.h	Process records: 1. beginning time 2. user time 3. system time 4. elapsed time 5. memory used 6. chars transferred 7. blocks read / written 8. command name

Figure 5-3 (Part 1 of 3). Accounting File Formats

File Name	Record Format	Contents
daytacct **sum/tacct**	tacct (acct.h)	Total accounting records: 1. user number 2. user name 3. total kcore minutes 4. total connect time 5. total disk usage 6. number of processes 7. number of login sessions 8. number of disk samples 9. fees for special services 10. total chars transferred 11. total blocks read / written 12. queuing system charges
ptacct	tacct	Process total accounting records: 1. user number 2. user name 3. total kcore minutes 4. number of processes 5. total chars processed 6. total blocks read / written
ctacct	tacct	Connect total accounting records: 1. user number 2. user name 3. total connect time

Figure 5-3 (Part 2 of 3). Accounting File Formats

File Name	Record Format	Contents
cms **daycms**	binary & ASCII	Command summary records: 1. command name 2. number of times called 3. total kcore minutes 4. total processor minutes 5. total read minutes 6. mean memory size 7. mean processor size 8. Hog factor

Figure 5-3 (Part 3 of 3). Accounting File Formats

Accounting System Files

Files in the /usr/adm Directory

diskdiag	Diagnostic output during the execution of disk accounting programs.
dtmp	Output from the **acctdusg** command.
fee	Output from the **chargefee** command, in ASCII **tacct** records.
pacct	Active process accounting file.
Spacct?.*MMDD*	Process accounting files for *MMDD* during the execution of **runacct**.

Files in the /usr/adm/acct/nite Directory

active	Used by **runacct** to record progress and print warning and error messages. The file **active***MMDD* is a copy of **active** made by **runacct** after it detects an error.
cms	ASCII total command summary used by the **prdaily** command.
ctacct.*MMDD*	Connect total accounting records.
ctmp	Connect session records.

daycms	ASCII daily command summary used by the **prdaily** command.
daytacct	Total accounting records for one day.
dacct	Disk total accounting records, created by **dodisk**.
accterr	Diagnostic output produced during the execution of **runacct**.
lastdate	The last day **runacct** executed, in **date +%m%d** format.
lock **lock1**	Used to control serial use of **runacct.**
lineuse	tty line usage report used by **prdaily**.
log	Diagnostic output from **acctcon1**.
log*mmdd*	Same as **log** after **runacct** detects an error.
reboots	Contains beginning and ending dates from **wtmp**, and a listing of system restarts.
statefile	Used to record the current state during execution of **runacct**.
tmpwtmp	wtmp file corrected by **wtmpfix**.
wtmperror	Contains **wtmpfix** error messages.
wtmperr*mmdd*	Same as **wtmperror** after **runacct** detects an error.
wtmp.*mmdd*	The previous day's **wtmp** file.

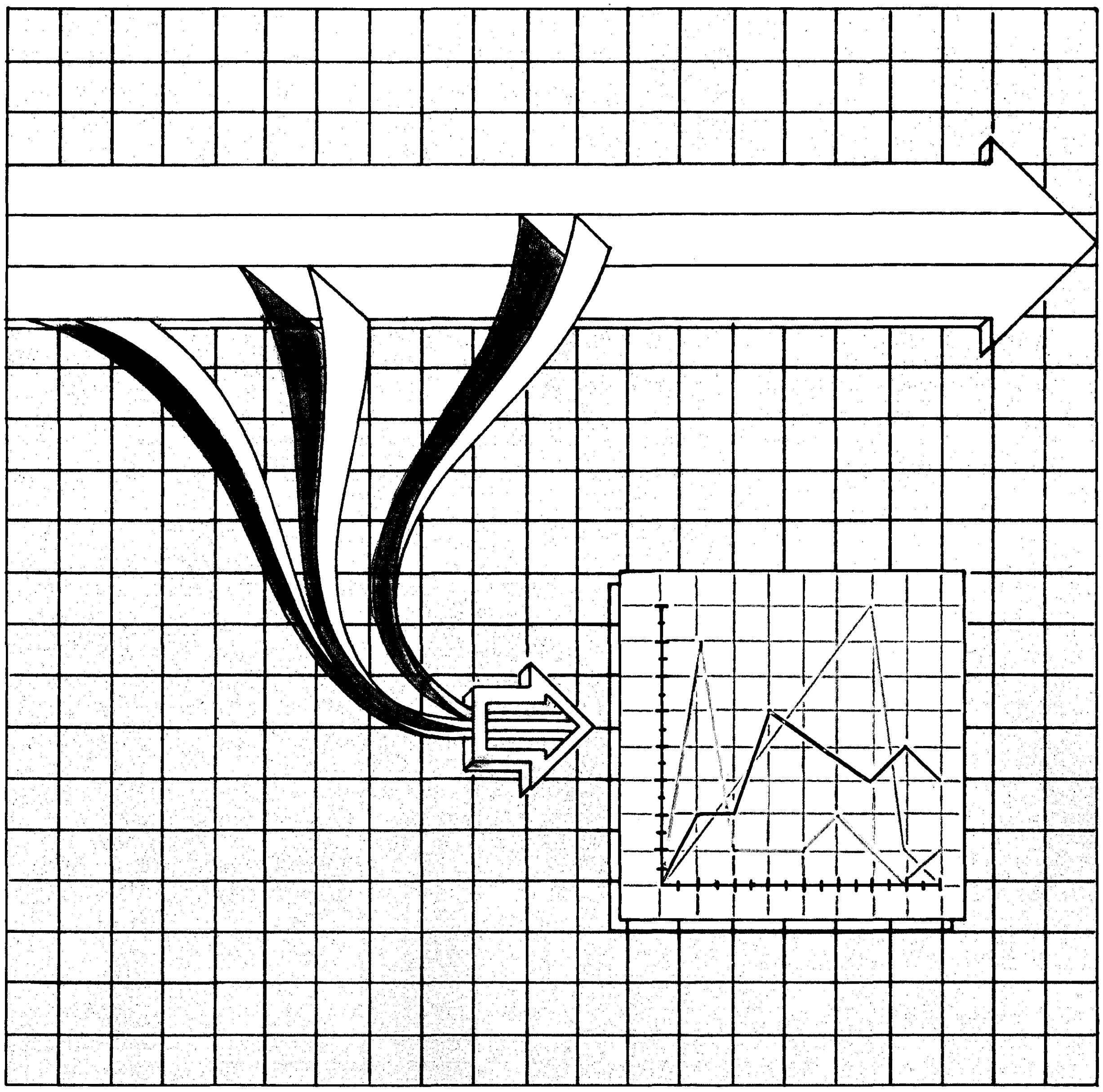
Files in the /usr/adm/acct/sum Directory

cms	Total command summary file for the current fiscal period, in binary format.
cmsprev	The command summary file without the latest update.
daycms	The command summary file for the previous day, in binary format.
lastlogin	The file created by **lastlogin**.
pacct.*mmdd*	Concatenated version of all **pacct** files for *mmdd.* This file is removed after system startup by the **remove** procedure.
rprt*mmdd*	The saved output of **prdaily**.
tacct	The cumulative total accounting file for the current fiscal period.
tacctprev	The same as **tacct** without the latest update.
tacct*mmdd*	The total accounting file for *mmdd.*
wtmp.*mmdd*	The saved copy of the **wtmp** file for *mmdd.* This file is removed after system startup by the **remove** procedure.

Files in the /usr/adm/acct/fiscal Directory

cms*?*	The total command summary file for fiscal period *?*, in binary format.
fiscrpt*?*	A report similar to that of **prdaily** for fiscal period *?*.
tacct*?*	The total accounting file for fiscal period *?*.

Chapter 6. Using the System Activity Package



CONTENTS

About This Chapter

This chapter describes the design and implementation of the AIX System Activity Package.

Note: The system activity package is part of the Multi-User Services licensed program. Before you can use the system activity package, the Multi-User Services licensed program must be installed on your system.

An Introduction to the System Activity Package

The AIX Operating System contains a number of counters that are incremented as various system actions occur. The system activity package reports system-wide measurements, including central processing unit utilization, disk and tape I/O activities, terminal device activity, buffer usage, system calls, system switching, file-access activity, queue activity, and message and semaphore activities.

The package provides three commands that generate various types of reports. Procedures that automatically generate daily reports are also included. The four functions of the activity package are:

The **sar** command	This command allows you to generate system activity reports in real-time and to save system activities in a file for later use.
The **sag** command	This command displays system activity in a graphical form.
The **timex** command	This command is a modified **time** command that times a process and also optionally reports concurrent system activity and process accounting activity.
Daily Reports	Procedures are provided for sampling and saving system activities in a data file periodically and for generating the daily report from the data file.

The system activity information reported by this package is derived from a set of system counters located in the AIX Operating System kernel. These system counters are described under "System Activity Counters" on page 6-5. "System Activity Commands" on page 6-9 describes the commands provided by this package. The procedure for producing daily reports is described under "System Activity Daily Reports" on page 6-12.

System Activity Counters

The AIX Operating System manages a number of counters that record various activities and provide the basis for the system activity reporting system. The data structure for most of these counters is defined in the **sysinfo** structure in **/usr/include/sys/sysinfo.h** (see "System Activity Data Structures and File Formats" on page 6-14). The system table overflow counters are kept in the **_syserr** structure. The device activity counters are extracted from the device status tables.

The following is a list of the system activity counters sampled by the system activity package:

CPU time counters
At each clock interrupt, the system increments one of four time counters (**cpu[]**), depending on the mode the CPU is in at the interrupt (idle, user, kernel, and wait for I/O completion).

lread and lwrite
The **lread** and **lwrite** counters contain the count of logical read and write requests issued by the system to block devices.

bread and bwrite
The **bread** and **bwrite** counters contain a count of the number of times data are transferred between the system buffers and the block devices. The ratio of block I/O to logical I/O is a common measure of the effectiveness of system buffering.

phread and phwrite
The **phread** and **phwrite** contains a count of the read and write requests issued by the system to raw devices.

pswitch and syscall

These counters are related to the management of multiprogramming. **syscall** is incremented every time a system call is invoked. The numbers of invocations of **read**, **write**, **fork**, and **exec** system calls are kept in counters **sysread**, **syswrite**, **sysfork**, and **sysexec**. **pswitch** counts the times the *switcher* was invoked, which occurs when:

1. A system call resulted in a road block.
2. An interrupt occurred resulting in awakening a higher priority process.
3. A one-second clock interrupt occurs.

iget, namei, and dirblk

These counters apply to file-access operations. **iget** and **namei**, in particular, are the names of AIX Operating System routines. The counters record the number of times the respective routines are called.

The **namei** routine performs file system path searches. It searches the various directory files to get the associated i-number of a file corresponding to a special file.

The **iget** routine locates the i-node entry of a file. It first searches the i-node table in memory. If the i-node entry is not in the table, the **iget** routine gets the i-node from the file system where the file resides and enters it in the i-node table in memory. **iget** returns a pointer to this entry.

The **namei** routine calls **iget**, but other file access routines also call **iget**. Therefore, counter **iget** is always greater than counter **namei**.

Counter **dirblk** records the number of directory block reads issued by the system. Note that the directory blocks read divided by the number of **namei** calls estimates the average path length of files.

runque and runocc

These counters record queue activities. They are implemented in the **clock.c** routine. The clock routine periodically examines the process table to see whether any processes are in memory and in ready state. If so, the system increments the counter **runocc** and adds the number of such processes to counter **runque**.

readch and writech

The **readch** and **writech** counters record the total number of bytes (characters) transferred by the **read** and **write** system calls.

Monitoring terminal device activities

There are six counters monitoring terminal device activities. **rcvint**, **xmtint**, and **mdmint** are counters measuring hardware interrupt occurrences for receiver, transmitter, and modem individually. **rawch**, **canch**, and **outch** count the number of characters in the raw queue, canonical queue, and output queue. Characters generated by devices operating in the "cooked" mode, such as terminals, are counted in both **rawch** and (as edited) in **canch**; but characters from raw devices, such as communication processors, are counted only in **rawch**.

msg and sema counters

These counters record message sending and receiving and semaphore operations.

Monitoring I/O activities

Four counters are kept for each disk or tape drive in the device status table. Counter `io_ops` is incremented when an I/O operation has occurred on the device. It includes block I/O, and physical I/O. `io_bcnt` counts the amount of data transferred between the device and memory in 512-byte units. `io_act` and `io_resp` measure the active time and response time for a device in time ticks summed over all I/O requests that have completed for each device. The device active time includes the device seeking, rotating, and data transferring times, while the response time of an I/O operation is the time the I/O request is queued to the device to the time when the I/O completes.

i-nodeovf, fileovf, textovf, and procovf

These counters are extracted from the `_syserr` structure. When an overflow occurs in any of the i-node, file, text, and process tables, the corresponding overflow counter is incremented.

System Activity Commands

The system activity package provides three commands for generating various system activity reports and one command for profiling disk activities. These tools facilitate observation of system activity during:

- A controlled standalone test of a large system
- An uncontrolled run of a program to observe the operating environment
- Normal production operation.

The **sar** and **sag** commands permit you to specify a sampling interval and number of intervals for examining system activity and then to display the observed level of activity in tabular or graphical form. The **timex** command reports the amount of system activity that occurred during the precise period of execution of a timed command.

The sar Command

The **sar** command can be used in the following two ways:

- When you specify the frequency parameters *interval* and *number*, **sar** invokes the data collection program **sadc** to sample the system activity counters in the operating system every *interval* seconds for *number* intervals and generates system activity reports in real-time. Generally, it is desirable to include the option to save the sampled data in a file for later examination. For the format of this file, see "System Activity Data Structures and File Formats" on page 6-14. In addition to the system counters, a time stamp is also included. It gives the time at which the sample was taken.
- If you do not supply frequency parameters, **sar** generates system activity reports for a specified time interval from an existing data file that was created by **sar** at an earlier time.

A convenient usage is to run **sar** as a background process, saving its samples in a temporary file but sending its standard output to **/dev/null**. Then an experiment is conducted after which the system activity is extracted from the temporary file.

See **sar** in *AIX Operating System Commands Reference* for a discussion of all flags and usage.

The sag Command

The **sag** command displays system activity data graphically. It relies on the data file produced by a prior run of **sar**, from which any column of data or the combination of columns of data can be plotted. A fairly simple but powerful command syntax allows the specification of cross plots or time plots. Data items are selected using the **sar** column header names.

See **sag** in *AIX Operating System Commands Reference* for a discussion of all flags and usage.

The timex Command

The **timex** command is an extension of the **time** command. Without flags, **timex** behaves like **time**. In addition to giving the time information, it can also display a system activity report and a process accounting report. For all the flags available, see *AIX Operating System Commands Reference.*

In the report, the **user** and **sys** times reported in the second and third lines apply to the measured process itself, including all its children. The remaining data, including **cpu user %** and **cpu sys %**, apply to the entire system.

While the normal use of **timex** will probably be to measure a single command, multiple commands can also be timed, either by combining them in an executable file and timing it or by entering:

```
timex sh -c "cmd1; cmd2;...;"
```

This establishes the necessary parent-child relationships to correctly extract the user and system consumed by *cmd1*, *cmd2*, . . . (and the shell).

System Activity Daily Reports

The previous section described the commands available to users to initiate activity observations. It is probably desirable for each installation to routinely monitor and record system activity in a standard way for historical analysis. This section describes the steps that you may follow to automatically produce a standard daily report of system activity.

Facilities

sadc	This command reads system counters from **/dev/kmem** and records then in a file. In addition to the file parameter, two frequency parameters are usually specified to indicate the sampling interval and number of samples to be taken. If frequency parameters are not given, it writes a dummy record in the file to indicate s system restart.
sa1	The shell procedure that invokes **sadc** to write system counters in the daily data file **/usr/adm/sa***dd*, where *dd* represents the day of the month. It may be invoked with sampling interval and iterations as parameters.
sa2	The shell procedure that invokes **sar** to generate daily report **/usr/adm/sa/sar***dd* from the daily data file **/usr/adm/sa/sa***dd*. It also removes the daily data files and report files after seven days. The starting and ending times and all report options of **sar** are applicable to **sa2**.

Suggested Operational Setup

You should use **cron** to control the normal data collection and report generation operations. For example, the sample entries in **/usr/spool/cron/crontabs/sys**:

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

would cause the data collection program **sadc** to be invoked every hour on the hour. Moreover, depending on the arguments presented, it writes data to the data file one to three times at every 20 minutes. Therefore, under the control of **cron**, the data file is written every 20 minutes between 8:00 and 18:00 on week days and hourly at other times.

Note that data samples are taken more frequently during prime time on week days to make them available for a finer and more detailed graphical display. You should invoke **sa1** hourly rather than invoking it once every day, as this ensures that if the system crashes, data collection will be resumed within an hour after the system is restarted.

Because system activity counters restart from zero when the system is restarted, a special record is written on the data file to reflect this situation. This process is accomplished by invoking **sadc** in **/etc/rc**, without frequency parameters, when going to multiuser state:

```
su adm -c "/usr/lib/sa/sadc /usr/adm/sa/sa` date +%d`"
```

cron also controls the invocation of **sar** to generate the daily report via shell procedure **sa2**. You may choose the time period the daily report is to cover and the groups of system activity to be reported. For instance, if:

```
0 20 * * 1-5 /usr/lib/sa/sa2 -s 8:00 -e 18:00 -i 3600 -uybd
```

is an entry in **/usr/spool/cron/crontabs/sys**, **cron** will execute the **sar** command to generate daily reports from the daily data file at 20:00 on week days. The daily report reports the processor utilization, terminal device activity, buffer usage, and device activity every hour from 8:00 to 18:00.

In case of a shortage of the disk space or for any other reason, these data files and report files can be removed by a user with superuser authority.

System Activity Data Structures and File Formats

This section contains the data structures and file formats used by the system activity package.

sysinfo.h

```
#ifndef _h_SYSINFO
#define _h_SYSINFO

struct sysinfo {
        time_t    cpu[4]
#define CPU_IDLE      0
#define CPU_USER      1
#define CPU_KERNEL        2
#define CPU_WAIT      3
        time_t    wait[3]
#define W_IO        0
#define W_PIO       2
        long      bread;
        long      bwrite;
        long      lread;
        long      lwrite;
        long      phread;
        long      phwrite;
        long      pswitch;
        long      syscall;
        long      sysread;
        long      syswrite;
        long      sysfork;
        long      sysexec;
        long      runque;
        long      runocc;
        long      iget;
        long      namei;
        long      dirblk;
        long      readch;
```

```
	long	writech;
	long	rcvint;
	long	xmtint;
	long	mdmint;
	long	rawch;
	long	canch;
	long	outch;
	long	msg;
	long	sema;
	long	ksched;
	long	koverf;
	long	kexit;
};

extern struct sysinfo sysinfo;

struct syswait {
	short	iowait;
	short	physio;
};

extern struct syswait syswait;

extern syserr {
	long	i-nodeovf;
	long	fileovf;
	long	textovf;
	long	procovf;
	long	sbi[5];
#define SBI_SILOC	0
#define SBI_CRDRDS	1
#define SBI_ALERT	2
#define SBI_FAULT	3
#define SBI_TIMEO	4
};

extern struct syserr syserr;
#endif
```

sar Data File Structure

The structure of the binary daily data file is:

```
struct sa {
     struct sysinfo si;
     int szi-node;   /* current entries of i-node table */
     int szifile;    /* current entries of file table */
     int szitext;    /* current entries of text table */
     int sziproc;    /* current entries of proc table */
     int mszi-node;       /* size of i-node table */
     int mszfile;    /* size of file table */
     int msztext;    /* size of text table */
     int mszproc;    /* size of proc table */
long i-nodeofv;      /* cum. overflows of i-node table */
     long fileofv;   /* cum. overflows of file table */
     long textofv;   /* cum. overflows of text table */
     long procofv;   /* cum. overflows of proc table */
     time_t ts;      /* time stamp, seconds */
     long devio[NDEVS][]   /* device info for up to NDEVS units */
#define IO_OPS   0         /* cum. I/O requests */
#define IO_BCNT  1    /* cum. blocks transferred */
#define IO_ACT   2         /* cum. drive busy time in ticks */
#define IO_RESP  3    /* cum. I/O resp time in ticks */
};
```

Appendix A. Printer Control Codes

Figure A-1 on page A-2 lists the printer control codes that you can use with RT PC printers. If the printer can perform the function by itself, the system passes the code directly to the printer. If the codes do not work on the printer installed on your system, the printer support system performs one of the following actions:

- Tries to emulate the control with functions that the installed printer does have.
- Removes the control from the output stream. The function is not performed.

The three code columns in the table show different representations of the same code, depending on how you enter the code into the data stream.

Control Name This column shows the name of the control character. In many cases the name is the same as the keyboard key that produces the required ASCII code for the control code.

Hex Code This column show the hexadecimal representation of the control code.

ASCII Code This column shows the decimal representation of the control code.

Category	Function Performed	Control Name	Hex Code	ASCII Code
Control:	Provides a null value. Used as a list terminator.	NUL	00	0
	Sounds the buzzer. [2]	BEL	07	7
	Prints the next character as a printable character. The next character is a control with an ASCII value of less than 32.	ESC ^	1B5E	27 94
	Prints more than one character with an ASCII value that is below 32.	ESC \ *m n c*	1B5C *m n c*	27 92 *m n c*
	Clears the printer memory of all data waiting to be printed following the last received line feed. [1]	CAN	18	24
	Sets resolution for raster image print (*n* indicates a string of control bytes).[1]	ESC [O *n*	1B5B4F *n*	27 91 79 *n*
	Performs a printer power on reset.[1]	ESC [K	1B5B4B 0100 0	27 91 75 1 0 0
Positioning the Printhead:	Sets back space.	BS	08	8
	Sets horizontal tab.	HT	09	9
	Sets horizontal tabs (*n* is a list of one or more tab positions).	ESC D *n* NUL	1B44*n*00	27 68 *n* 0
	Sets tab stops to power-on settings.	ESC R	1B52	27 82
	Sets line feed.	LF	0A	10
	Sets reverse line feed.	ESC]	1B5D	27 93
	Starts automatic line feed.	ESC 5 1	1B35 1	27 53 1

Figure A-1 (Part 1 of 7). Printer Control Codes

Category	Function Performed	Control Name	Hex Code	ASCII Code
	Stops automatic line feed.	ESC 5 0	1B35 0	27 53 0
	Provides a carriage return (no line feed).	CR	0D	13
	Provides a vertical tab.	VT	0B	11
	Sets vertical tabs (*n* is a list of tab positions).	ESC B *n* NUL	1B42 *n* NUL	27 66 *n* NUL
Paper Control:	Provides a form feed.	FF	0C	12
	Sets top of forms. [2]	ESC 4	1B34	27 52
	Ignores End of Forms. [2]	ESC 8	1B38	27 56
	Respects End of Forms. [2]	ESC 9	1B39	27 57
	Sets skip perforation [2] (*n* is lines to skip).	ESC N *n*	1B4E *n*	27 78 *n*
	Stops skip perforation [2]	ESC O	1B4F	27 79
Formatting the Page Image:	Uses 12 characters-per-inch printing.	ESC :	1B3A	27 58
	Sets 1/8″ Line spacing.	ESC 0	1B30	27 48
	Starts *n*/72″ Line spacing.	ESC 2	1B32	27 50
	Sets *n*/72″ Line spacing.	ESC A *n*	1B41 *n*	27 59 *n*
	Sets Page Length. [2] (*n* is lines per page).	ESC C *n*	1B43 *n*	27 67 *n*
	Sets Page Length [2] (*n* is inches per page).	ESC C 0 *n*	1B43 0 *n*	27 67 0 *n*

Figure A-1 (Part 2 of 7). Printer Control Codes

Category	Function Performed	Control Name	Hex Code	ASCII Code
	Sets left and right margins (*m* and *n* are column numbers).	ESC X *m n*	1B58 *m n*	27 88 *m n*
	Sets top and bottom margins (*m* and *n* are length of control; *t1 t0* are high/low-order bytes of top margin; *b1 b0* are high/low-order bytes of bottom margin).	ESC [S *m n t1 t0 b1 b0*	1B5B53 *m n t1 t0 b1 b0*	27 91 83 *m n t1 t0 b1 b0*
	Starts automatic line justification.	ESC M 1	1B4D 1	27 77 1
	Stops automatic line justification.	ESC M 0	1B4D 0	27 77 0
	Starts proportional spacing.	ESC P 1	1B50 1	27 80 1
	Stops proportional spacing.	ESC P 0	1B50 0	27 80 0
	Sets print resolution for draft quality font and ESC K image data (*n* indicates the resolution value).[1]	ESC [0 *n*	1B5B30 0100 *n*	27 91 48 1 0 *n*
	Sets presentation surface color (*n* indicates the available colors).[1]	ESC [L *n*	1B5B4C 0200 00 *n*	27 91 76 2 0 *n*
Controlling the Ribbon:	Sets color band 1 (yellow).	ESC y	1B79	27 121
	Sets color band 2 (magenta).	ESC m	1B6D	27 109
	Sets color band 3 (cyan).	ESC c	1B63	27 99
	Sets color band 4 (black).	ESC b	1B62	27 98
	Sets automatic ribbon band shift.	ESC a	1B61	27 97

Figure A-1 (Part 3 of 7). Printer Control Codes

Category	Function Performed	Control Name	Hex Code	ASCII Code
	Sets the background color for Text, ESC K, or ESC [0 printing (*n* indicates the available colors).[1]	ESC [N *n*	1B5B4E 0100 *n*	27 91 78 1 0 *n*
	Sets the foreground color for Text, ESC K, or ESC [0 printing (*n* indicates the available colors).[1]	ESC [M *n*	1B5B4D 0100 *n*	27 91 77 1 0 *n*
	Swaps the foreground and background printing color (*n*= 0 indicates normal printing; *n*= 1 indicates reversing background and foreground colors).[1]	ESC []	1B5B5D 0100 *n*	27 91 93 1 0 *n*
Selecting Print Mode:	Starts double wide printing.	SO	0E	14
	Stops double wide printing.	DC4	14	20
	Starts double wide continuous printing.	ESC W 1	1B57 1	27 87 1
	Stops double wide continuous printing.	ESC W 0	1B57 0	27 87 0
	Starts compressed printing.	SI	0F	15
	Stops compressed printing.	DC2	12	18
	Starts underline printing.	ESC -1	1B2D 1	27 45 1
	Stops underline printing.	ESC -0	1B2D 0	27 45 0
	Starts emphasized printing.	ESC E	1B45	27 69
	Stops emphasized printing.	ESC F	1B46	27 70
	Starts double strike printing.	ESC G	1B47	27 71
	Stops double strike printing.	ESC H	1B48	27 72

Figure A-1 (Part 4 of 7). Printer Control Codes

Category	Function Performed	Control Name	Hex Code	ASCII Code
	Starts superscript printing.	ESC S 0	1B53 0	27 83 0
	Starts subscript printing.	ESC S 1	1B53 1	27 83 1
	Stops superscript or subscript printing.	ESC T	1B54	27 84
	Starts color underline, bypassing white space (*n* defines the available color).[1]	ESC [B 1 *n*	1B5B42 0200 1 *n*	27 91 66 2 0 1 *n*
	Starts continuous color underline (*n* defines the available color).[1]	ESC [B 0 *n*	1B5B42 0200 0 *n*	27 91 66 2 0 0 *n*
	Stops underline.	ESC [E	1B5B450000	27 91 69 0 0
Selecting the Character Set:	Uses PC character set 2.	ESC 6	1B36	27 54
	Uses PC character set 1.	ESC 7	1B37	27 55
	Selects font (*n* specifies the font; varies with printer type).	ESC I *n*	1B49 *n*	27 73 *n*
	Sets graphic set ID (*c* selects graphic set 0, 1 or 2).	ESC [T 1 0 *c*	1B5B54 1 0 *c*	27 91 84 1 0 *c*
Using Bit Image Graphics: [3]	Sets bit graphics normal (*n* is a string of control bytes).	ESC K *n*	1B4B *n*	27 75 *n*
	Sets graphics dual-half speed (*n* is a string of control bytes).	ESC L *n*	1B4C *n*	27 76 *n*
	Sets bit graphics dual-normal speed (*n* is a string of control bytes).	ESC Y *n*	1B59 *n*	27 89 *n*
	Sets bit graphics high-half speed (*n* is a string of control bytes).	ESC Z *n*	1B5A *n*	27 90 *n*

Figure A-1 (Part 5 of 7). Printer Control Codes

Category	Function Performed	Control Name	Hex Code	ASCII Code
	Sets aspect ratio to 1:1.	ESC n 1	1B6E 1	27 110 1
	Sets aspect ratio to 5:6.	ESC n 0	1B6E 0	27 110 0
	Moves carriage to home position.	ESC <	1B3C	27 60
	Moves right *n*/120.	ESC d *n*	1B64 *n*	27 100 *n*
	Moves left *n*/120.	ESC e *n*	1B65 *n*	27 101 *n*
	Starts unidirectional printing.	ESC U 1	1B551	27 85 1
	Stops unidirectional printing.	ESC U 0	1B550	27 85 0
	Sets 7 dot line spacing.	ESC 1	1B31	27 49
	Sets graphics line spacing (*n* is the number of 1/216-inch steps).	ESC 3 *n*	1B33 *n*	27 51 n
	Sets variable space line feed (*n* is the number of 1/216-inch steps).	ESC J *n*	1B4A *n*	27 74 *n*
	Moves vertical presentation down the page in number of dots (*n* indicates how far to move the presentation). [1]	ESC [U *n*	1B5B55 0100 *n*	27 91 85 1 0 *n*
Selecting a Printer:	Selects the printer to accept data. [2]	DC1	11	17
	Deselects the printer so as to not receive data. [2]	DC3	13	19
	Sets initialize function on. [2]	ESC ? 1	1B3F 1	27 63 1
	Sets initialize function off. [2]	ESC ? 0	1B3F 0	27 63 0

Figure A-1 (Part 6 of 7). Printer Control Codes

Category	Function Performed	Control Name	Hex Code	ASCII Code
	Queries a parallel attached printer for identification. If the device queried is equal to *n*, this printer deactivates the select line. [2]	`ESC Q` *n*	`1B51` *n*	`27 81` *n*

Figure A-1 (Part 7 of 7). Printer Control Codes

[1] Use with the Color Jet Printer.

[2] Do not use these controls when using the print queue.

[3] These controls may not work on the installed printer. Use *passthrough* mode to send these codes to the printer.

Appendix B. Getting Started With Distributed Services Customization Commands

This appendix tells you how to use the commands that build the Distributed Network Node Table, the Network Users/Groups Table, and Distributed IPC Queues Table.

Note: You must have superuser authority or be a member of the system group to create or modify these tables. Any user can browse the tables for information.

Contents

Using Distributed Services Menus

To use a windows interface to build, examine, or modify the Distributed Network Node Table, the Network Users/Groups Table, or the Distributed IPC Queues Table, run one of the following commands:

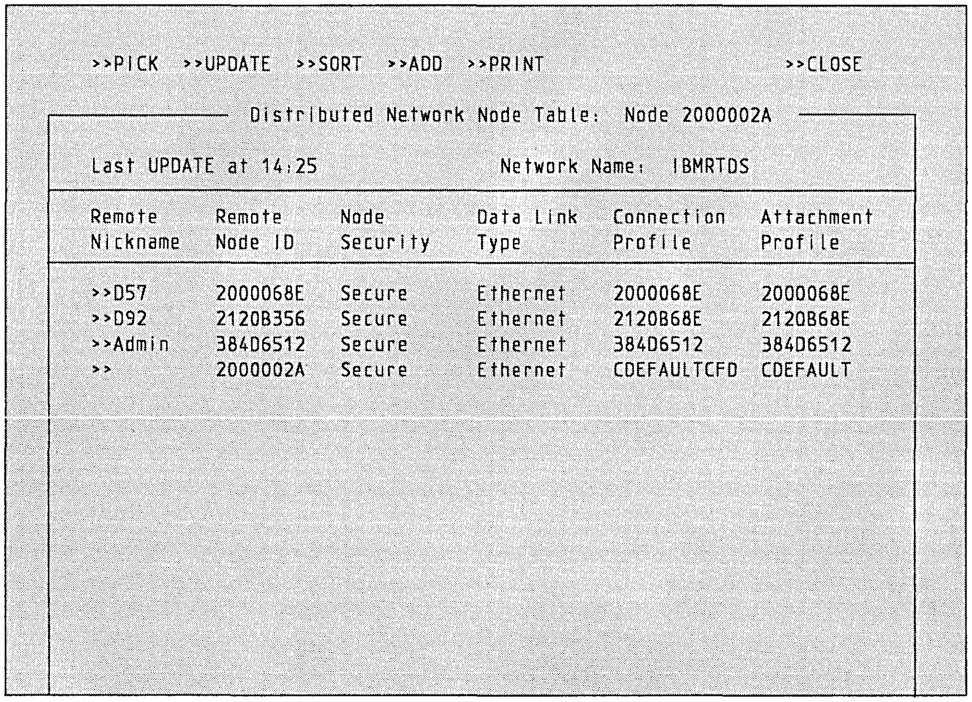
ndtable	To access the Distributed Network Node Table
ugtable	To access the Network Users/Groups Table
ipctable	To access the Distributed IPC Queues Table.

These commands use data files that reside, by default, in the local directory **/etc/profsvcs/pfslocal**. When you start up each command, it asks you to identify the data files you want to access:

```
Enter Node ID Nickname      ».............
(Default is the local id.)
```

If you specify a NID or node nickname, the files found in **/etc/profsvcs/pfslocal** on the specified node are accessed. If you specify a directory name, the command looks under **/etc/profsvcs** for a directory with that name and uses the files found in that directory.

The list of nodes, UID/GID translations, or IPC message queues currently configured then appears in a ***window***:

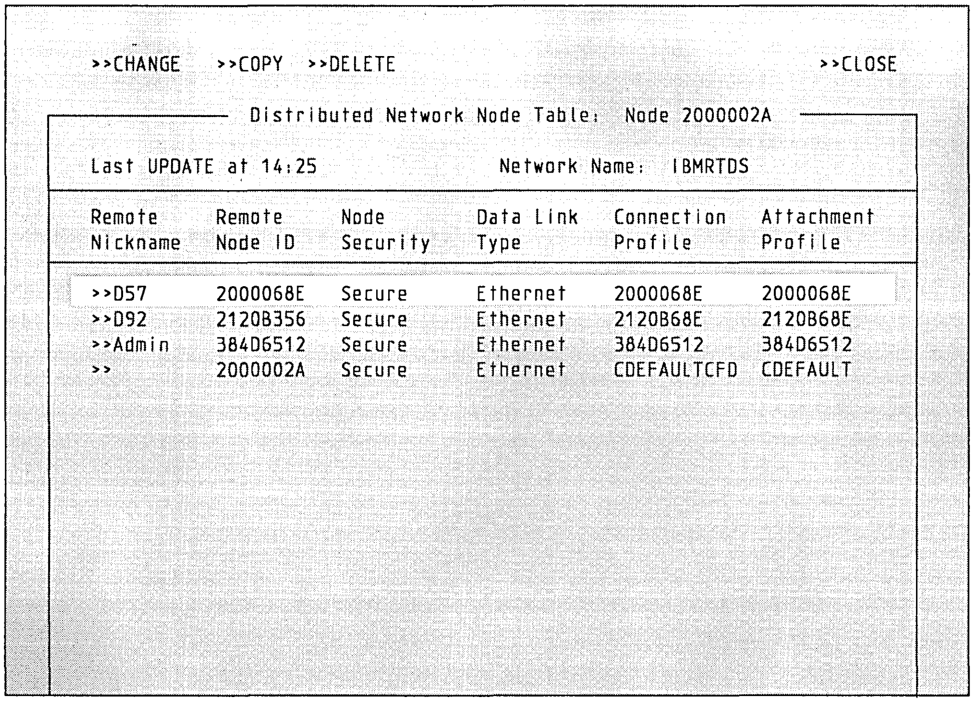


```
    >>PICK  >>UPDATE  >>SORT  >>ADD  >>PRINT                      >>CLOSE

 ------------- Distributed Network Node Table:  Node 2000002A ------------

    Last UPDATE at 14:25                   Network Name:  IBMRTDS

    Remote      Remote      Node        Data Link   Connection   Attachment
    Nickname    Node ID     Security    Type        Profile      Profile

    >>D57       2000068E    Secure      Ethernet    2000068E     2000068E
    >>D92       2120B356    Secure      Ethernet    2120B68E     2120B68E
    >>Admin     384D6512    Secure      Ethernet    384D6512     384D6512
    >>          2000002A    Secure      Ethernet    CDEFAULTCFD  CDEFAULT
```

AJ2DL007

Information also appears in a ***command bar*** above the window and in ***pop-up panels*** that appear when you make certain selections. In the display screen shown above, the available commands are `PICK`, `UPDATE`, `SORT`, `ADD`, `PRINT`, and `CLOSE`.

The command bar displays the commands currently available in a session. When you select an entry in the list, the command bar changes to show the commands that are available for that entry:



```
>>CHANGE   >>COPY  >>DELETE                                          >>CLOSE

            Distributed Network Node Table:  Node 2000002A

Last UPDATE at 14:25                  Network Name:  IBMRTDS

Remote     Remote     Node       Data Link   Connection   Attachment
Nickname   Node ID    Security   Type        Profile      Profile

>>D57      2000068E   Secure     Ethernet    2000068E     2000068E
>>D92      2120B356   Secure     Ethernet    2120B68E     2120B68E
>>Admin    384D6512   Secure     Ethernet    384D6512     384D6512
>>         2000002A   Secure     Ethernet    CDEFAULTCFD  CDEFAULT
```

AJ2DL006

In the display screen shown above, the commands that appear on the command bar when you select an entry are CHANGE, COPY, DELETE, and CLOSE.

Depending on the display station, either the > or » symbol appears before certain items on the display screen. Each item (word, phrase, or symbol) that follows this symbol is called a ***button*** and you can select it as a unit from the display screen. If an area filled with periods (...) follows this symbol, this area is called an ***input field***. An input field allows you to type in or change the data required in an entry.

You can use a mouse to select entries, buttons, and input fields, or you can use Usability Services Functions from the keyboard. To identify the necessary keys or key sequences, see the *Usability Services Functions* keyboard template that corresponds to your keyboard. The functions you will probably use most frequently are **Command Bar** (to move between the display window and the command bar), **Do**, **Help**, **Quit**, **Select**, and **Tab**.

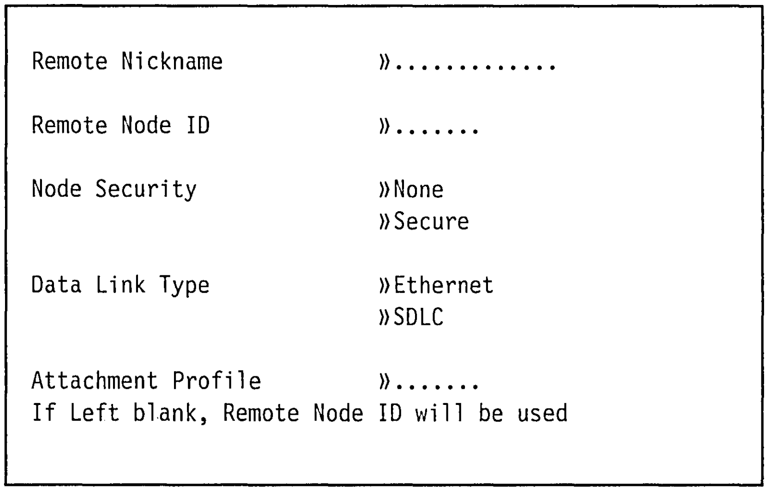
Command Bar Commands

ADD—Adding Entries to a Table

Within any tables list window, selecting ADD allows you to add a new entry to the list.

Adding a Node to the Network

When you select ADD from the command bar of the Distributed Network Node Table window, you see the following pop-up panel:



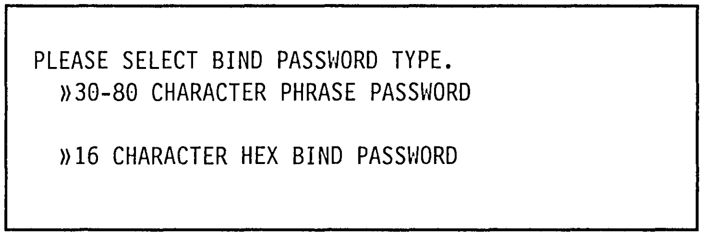
```
Remote Nickname          ».............

Remote Node ID           ».......

Node Security            »None
                         »Secure

Data Link Type           »Ethernet
                         »SDLC

Attachment Profile       ».......
If Left blank, Remote Node ID will be used
```

Figure B-1. Distributed Network Node Table ADD Panel

You must provide a Remote Node ID. All other input fields are either optional (Remote Nickname) or have default values (Node Security, Data Link Type, Attachment Profile).

The Remote Nickname can contain both uppercase and lowercase letters. In the Node ID input field, **ndtable** converts lowercase alphabetic characters to uppercase.

If you select Secure, you are given the following additional choices:



```
PLEASE SELECT BIND PASSWORD TYPE.
  »30-80 CHARACTER PHRASE PASSWORD

  »16 CHARACTER HEX BIND PASSWORD
```

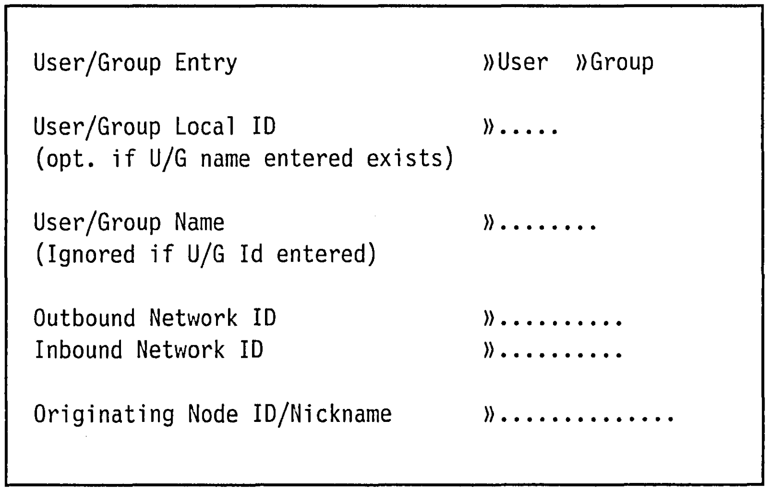
Figure B-2. Password Selection Panel

You are asked to enter your password twice:

When you have made all your choices in the **ADD** pop-up, press **Do** to add the new node to the list.

Adding a User or Group to the Network

When you select ADD from the command bar of the Network Users/Groups Table window, you will see the following pop-up panel:



```
User/Group Entry                   »User  »Group

User/Group Local ID                ».....
(opt. if U/G name entered exists)

User/Group Name                    »........
(Ignored if U/G Id entered)

Outbound Network ID                »..........
Inbound Network ID                 »..........

Originating Node ID/Nickname       »..............
```

Figure B-3. Network Users/Groups Table ADD Panel

Select User or Group to indicate whether the entry describes a user (the default) or a group ID. If you are adding an entry to a local table, you must specify either a numeric UID (GID) or the user (group) name. If you specify a numeric ID, the corresponding name from the **/etc/passwd** or **/etc/group** file will be displayed (even if you specify a different name). If you are adding an entry to a remote table, you must specify a numeric ID, and no corresponding name will be displayed.

If you enter a NID in the Originating Node ID/Nickname input field, you must enter all alphabetic characters as uppercase characters. Unlike **ndtable, ugtable** cannot convert lowercase characters to uppercase, since you can enter a node nickname in the same field, a name that can contain both upper- and lowercase characters.

You can have more than one entry in the table for a single local ID, but each entry must have the same outbound ID. If you try to add another entry that has a different outbound ID from the one already associated with that local ID, you will be asked if you want to change all the existing outbound IDs. You may then press **Do** to change all the outbound IDs or

Quit to return to the **ADD** pop-up panel where you can change the information for the new ID.

Adding a Message Queue to the Network

When you select ADD from the command bar of the Distributed IPC Queues Table window, you will see the following pop-up panel:

```
Queue Name                 ».............
Local Key                  »......     (196608-104575)
Remote Key                 »......     (196608-104575)
Remote Node ID/Nickname    ».............
```

Figure B-4. Distributed IPC Queues Table ADD Panel

If you are adding a local queue profile, you must specify a `Queue Name`. You can also specify a `Local Key` (one not already assigned). If you do not specify a local key, **ipctable** provides that value when you press **Do**.

If you are adding a remote queue profile, you must specify a `Queue Name`, a `Remote Key`, and a `Remote Node ID/Nickname`. The queue name and remote key must match those of the corresponding queue profile installed at the remote node.

If you enter a NID in the `Originating Node ID/Nickname` input field, you must enter all alphabetic characters as uppercase characters. Unlike **ndtable**, **ipctable** cannot convert lowercase characters to uppercase, since you can enter a node nickname in the same field, a name that can contain both upper- and lowercase characters.

CHANGE—Changing an Entry

Selecting CHANGE allows you to modify the description of an existing entry.

Changing An Existing Node Entry

When you select CHANGE, a pop-up panel appears containing the fields that you can change. When you change an existing node entry in the local node, the following actions occur, depending on the changes you have made:

- If you change the node nickname or node ID, all tables—the Distributed Network Node Table, the Network Users/Groups Table, and the Distributed IPC Queues Table—are updated to reflect that change.
- If you change node security from None to Secure or if you reselect Secure, you are prompted for a node password (see page Figure B-2 on page B-8). The Node password is encrypted with the Communication Authority Password key and stored in the SNA Services connection profile. (For more information see *SNA Services Guide and Reference.*) The node password must be the same on both communicating nodes (that is, in the connection profile on each node).

  If you change node security from Secure to None, the node password is deleted from the SNA Services connection profile.
- If you change the attachment profile name, all the other SNA Services connection profiles that refer to the attachment profile are also changed. If the new attachment profile does not exist, it is created and if no other profiles use the old profile, it is deleted.

Changing Network User/Group Entries

The change pop-up panel displays the values in the user/group entry selected. The name comment field will not be displayed if a remotely mounted node is being configured. The change may be either to an ID or a name and the same validation checks will be made for the name/ID as in adding and copying. If the outbound ID is changed and if other entries have the same local ID, then you see a pop-up panel that allows you to change the outbound IDs of all old entries to the new ID or to return to the add/copy pop-up panel to correct the outbound ID on the new entry.

Note: The system does not update the **passwd** or **group** files. You must run the **users** command to update these files if necessary.

Another way to change a profile is to copy it first with the COPY command and then change the information in the copied panel.

Changing IPC Message Queues

You can change the local queue name, the local IPC key, the remote IPC key, or the remote node ID or nickname.

CLOSE—Closing a Window

Use the CLOSE command when you have finished working with a window. This command removes the selected window from the screen and returns you to the AIX Operating System.

COPY—Copying Profiles

The COPY command copies a selected profile entry. It does not change the original entry.

Sometimes it is easier to copy an entry to make minor changes rather than adding an entry. When you copy a node entry, the default values supplied for the SNA Services profiles used for communicating with this node are taken from the selected node entry.

DELETE—Deleting an Entry

Warning: The DELETE command permanently erases the selected profile entry.

Deleting An Existing Remote Node Entry

When you delete an existing remote node entry from the local node's database, **ndtable** does the following:

- Deletes the node's SNA Services Protocol profiles, if they are not used by any other profiles.
- Removes any keys in the IPC key profile that contain the NID of the deleted node.
- Removes all Network Users/Groups Table entries that identify the deleted node as the originating node.

Deleting a User or Group

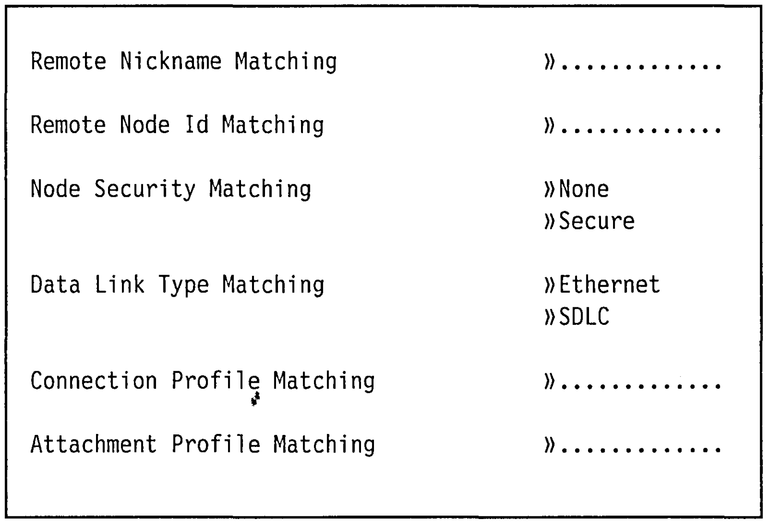
If you delete an entry in the local Network Users/Groups Table, the kernel rebuilds its Network Users/Groups Table when you have completed configuring the table.

Note: The system does not delete any entries in the **passwd** or **group** files. You must run the **users** command to make any required updates to these files.

PICK—Picking Entries

The PICK command allows you to select which entries to display in the window by specifying a pattern, a specific name, or a attribute to match.

When you select PICK, you will see one of the following pop-panels:



```
Remote Nickname Matching                »............

Remote Node Id Matching                 »............

Node Security Matching                  »None
                                        »Secure

Data Link Type Matching                 »Ethernet
                                        »SDLC

Connection Profile Matching             »............

Attachment Profile Matching             »............
```

Figure B-5. Distributed Network Node Table PICK panel

```
User/Group Name Matching                »............
User/Group Matching                     »User   »Group
Local ID Matching                       »............
Outbound Network ID Matching            »............
Inbound Network ID Matching             »............
Originating Node ID/Nickname Matching   »............
```

Figure B-6. Network Users/Groups Table PICK panel

```
Queue Name Matching                »..............
Local Key Matching                 »..............
Remote Key Matching                »..............
Remote Node ID/Nickname Matching   »..............
```

Figure B-7. Distributed IPC Queues Table PICK panel

The strings you enter into these pop-up fields can contain pattern-matching characters (*) at the beginning, at the end, or at both the beginning and end of the string. However, you cannot imbed pattern-matching characters within the string. Select DO to process the pop-up choices. The current panel changes to display the entries that meet the specified criteria.

Press **Quit** to cancel a pop-up panel without saving your changes. To display all of the profiles again (after using PICK), type * in the NAME field and press **Do** or select UPDATE.

PRINT—Printing a List

Select PRINT to write the list displayed on the screen to a file or to the printer.

When you select PRINT, you will see the following pop-up panel:

```
Print with Headings:      »Yes     »No

Output Print to:          »File    »Printer
```

Figure B-8. PRINT Panel

You can choose to print both the headings and list that appear on the screen or only the list.

If your print destination is a file you will see the following additional panel:

```
File Name:                »...............

File Action:              »Replace     »Return Error
(Action taken if file already Exists)
```

Figure B-9. PRINT File Action Panel

Select the action you want the system to take if the specified file name already exists.

SORT—Sorting Entries

The SORT command allows you to select the order in which to list the profiles that are currently displayed in the Distributed Network Node Table, Network Users/Groups Table, or Distributed IPC Queues Table windows.

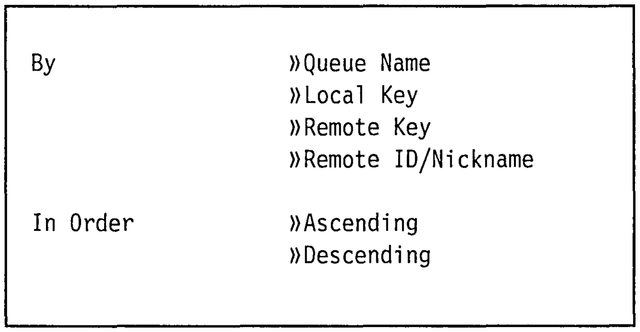
When you select SORT you see one of the following panels:

```
By                  »Remote Nickname
                    »Node ID
                    »Node Security
                    »Data Link Type
                    »Connection Profile
                    »Attachment Profile

In Order            »Ascending
                    »Descending
```

Figure B-10. Distributed Network Node Table SORT Panel

```
By:                 »User/Group Name
                    »User/Group
                    »Local ID
                    »Outbound Network ID
                    »Inbound Network ID
                    »Originating Node ID/Nickname

In Order:           »Ascending
                    »Descending
```

Figure B-11. Network Users/Groups Table SORT panel



```
By                  »Queue Name
                    »Local Key
                    »Remote Key
                    »Remote ID/Nickname

In Order            »Ascending
                    »Descending
```

Figure B-12. Distributed IPC Queues Table SORT Panel

You can select one field to sort by and one sort order.

UPDATE—Updating Profiles

Using UPDATE, you can refresh information displayed in a window with changes made from other windows. The UPDATE command is not a function automatically performed by the system. The time of the last UPDATE command is displayed so that you know when you last updated the list. For example:

```
Last Update at 15:30
```

Using the dsldxprof Command

The **dsldxprof** command can be very useful if you work with large Network Users/Groups Tables or if you need to make many changes. In addition, if you are building several Network Users/Groups Tables that have many translation entries in common, you can define those entries in a single ASCII file, and use **dsldxprof** to load them into each Network Users/Groups Table that you need to customize.

Adding Translation Entries

The **dsldxprof** command reads one line at a time from standard input or from an ASCII file specified with the **-f** flag. Each input line defines one translation entry and must contain the following six fields:

name *type* *local-ID* *outbound-ID* *inbound-ID* *node-ID*

where

name	Specifies a login name. If you provide a local ID number in the third field, you can use a - (minus) character as a place holder for this field.
type	Specifies whether the line defines a UID or a GID translation entry. The character **U** specifies a UID translation entry; the character **G** a GID translation entry. These can be lower- or uppercase characters.
local-ID	Specifies a local ID number. If you specify a valid user name in the first field, you can use a - (minus) character as a place holder for this field. **Note:** Unlike the **ugtable** command, **dsldxprof** does not warn you if the ID number is not defined in the **/etc/passwd** or **/etc/group** file.
outbound-ID	Specifies the outbound network ID number to associate with the local ID number. If you specify an inbound ID and a node ID, you can use a - (minus) character as a place holder for this field.

inbound-ID	Specifies the incoming network ID number to associate with the local ID number. If you do not specify a node ID, you can use a - (minus) character as a place holder for this field. If you do specify a node ID, you must provide a value for this field.
node-ID	Specifies the node ID number or nickname to associate with the inbound ID number specified in field four. If you do not specify an inbound ID, you can use a - (minus) character as a place holder for this field.

For example, the following input line:

```
-         U         0         0         0         *
```

specifies the same translation entry as the following **ugtable** panel:

```
User/Group Entry                     »User  »Group

User/Group Local ID                  »0.....
(opt. if U/G name entered exists)

User/Group Name                      ».......
(Ignored if U/G Id entered)

Outbound Network ID                  »0.........
Inbound Network ID                   »0.........

Originating Node ID/Nickname         »*............
```

Example One: The following file defines the translation entries necessary to set up limited network access for most users and to allow one network user to have local superuser authority:

```
-    U    100    100    *      *
-    G    100    100    *      *
-    U    0      -      201    20810CBF
```

Assuming that this file has the name `idtable` and is copied to each node in the network, the following command:

```
# dsldxprof -f idtable
```

builds a Network Users/Groups Table that enables user `201` working at node `20810CBF` to do any further network customization needed at any node in the network. Other users have limited access to the network until that customization is completed.

Note: The Distributed Network Node Tables must also be built at each node before user `201` can do remote customization.

Example Two: If all the nodes in the network share the same **/etc/passwd** and **/etc/group** files (see "Managing the Single System Image Environment" on page 3-70), you can use these files to build the Network Users/Groups Table.

Assume that a node has the following **/etc/passwd** and **/etc/group** files:

passwd file

```
root::0:0::/:
su:*:0:0::/:
daemon:*:1:1::/etc:
netmail:*:1:1::/usr/spool/qftp:/usr/lib/INnet/waxsrvr
bin:*:2:2::/bin:
sys:*:3:3::/usr/sys:
adm:*:4:4::/usr/adm:
uucp:*:5:1::/usr/spool/uucppublic:/usr/lib/uucp/uucico
adduser:*:0:0::/usr/adm:/etc/adduser
guest:*:100:100::/usr/guest:
tftpd:*:6:1::/tftpd:
smtpd:*:7:1::/smtpd:
ctw::201:1::/u/ctw:/bin/sh
mark::202:1::/u/mark:/bin/sh
mel::203:1::/u/mel:/bin/sh
gorg::205:1::/u/gorg:/bin/sh
geo::207:1::/u/geo:/bin/sh
maureen::208:1::/u/maureen:/bin/sh
```

group file

```
system::0:root,su,ctw
staff::1:ctw,mel,gorg,mark,geo,maureen
bin::2:root,su,bin
sys::3:root,su,bin,sys
adm::4:root,su,bin,adm
mail::6:root,su
usr::100:guest
daemons::7:tftpd,smtpd,root
```

Let us examine the results of running the sample shell program shown in Figure B-13 on page B-24 with this **passwd** and **group** file as input.

Lines 2-4 and 9-11 use the AIX **cut** and **paste** commands to pull out and recombine the first and third fields from the **/etc/passwd** and **/etc/group** files (the user or group name and the user or group number). The results are stored in two temporary files, `list1` and `list2`:

list1	**list2**
root:0 0 0	system:0 0 0
su:0 0 0	staff:1 1 1
daemon:1 1 1	bin:2 2 2
netmail:1 1 1	sys:3 3 3
bin:2 2 2	adm:4 4 4
sys:3 3 3	mail:6 6 6
adm:4 4 4	usr:100 100 100
uucp:5 5 5	daemons:7 7 7
adduser:0 0 0	
guest:100 100 100	
tftpd:6 6 6	
smtpd:7 7 7	
ctw:201 201 201	
mark:202 202 202	
mel:203 203 203	
gorg:205 205 205	
geo:207 207 207	
maureen:208 208 208	

```
#  @(#) make.idlist -- create a master ID list
#
#  Input:   /etc/passwd
#           /etc/group
#  Output:  ./id.list  (file containing master list of UIDs and GIDs)

dir=`pwd` ; cd /tmp

# Build master UID list

cut -f1,3 -d: /etc/passwd >names
cut -f3 -d: /etc/passwd >uids
paste -d" " names uids uids >list1

sed -e "s/:/ U /
s/$/ */
s/^[a-z]*/-/
s/100 100 100/100 100 */" list1 >uid.list

# Build master GID list

cut -f1,3 -d: /etc/group >names
cut -f3 -d: /etc/group >gids
paste -d" " names gids gids >list2

sed -e "s/:/ G /
s/$/ */
s/^[a-z]*/-/
s/100 100 100/100 100 */" list2 >gid.list

# Merge master ID lists

cat uid.list gid.list | sort +f2 -f3 | uniq >$dir/id.list

# Clean up

rm -r names uids gids list1 list2 uid.list gid.list
```

Figure B-13. make.idlist. Sample shell program that builds a master ID list

Next, this sample shell program uses the **sed** editor to:

Lines	**sed Action**
5, 12	Replaces the : (colon) separating the first two fields with " U " or " G ."
6, 13	Adds " * " to the end of each line.
7, 14	Replaces the user or group name with a place holder character.
8, 15	Replaces inbound id 100 with the wild card character.

The **sed** editor produces the following two temporary files:

uid.list	**gid.list**
`- U 0 0 0 *`	`- G 0 0 0 *`
`- U 0 0 0 *`	`- G 1 1 1 *`
`- U 1 1 1 *`	`- G 2 2 2 *`
`- U 1 1 1 *`	`- G 3 3 3 *`
`- U 2 2 2 *`	`- G 4 4 4 *`
`- U 3 3 3 *`	`- G 6 6 6 *`
`- U 4 4 4 *`	`- G 100 100 * *`
`- U 5 5 5 *`	`- G 7 7 7 *`
`- U 0 0 0 *`	
`- U 100 100 * *`	
`- U 6 6 6 *`	
`- U 7 7 7 *`	
`- U 201 201 201 *`	
`- U 202 202 202 *`	
`- U 203 203 203 *`	
`- U 205 205 205 *`	
`- U 207 207 207 *`	
`- U 208 208 208 *`	

Finally, in line 16, this program used the **cat** command to combine these two files, pipes the results to the **sort** command, and sends the sorted output to the **uniq** command to remove duplicate entries.

This produces the following file, ready to load with **dsldxprof**:

```
# dsldxprof -f id.list
```

id.list

```
- G 0 0 0 *
- U 0 0 0 *
- G 1 1 1 *
- U 1 1 1 *
- G 100 100 * *
- U 100 100 * *
- G 2 2 2 *
- U 2 2 2 *
- U 201 201 201 *
- U 202 202 202 *
- U 203 203 203 *
- U 205 205 205 *
- U 207 207 207 *
- U 208 208 208 *
- G 3 3 3 *
- U 3 3 3 *
- G 4 4 4 *
- U 4 4 4 *
- U 5 5 5 *
- G 6 6 6 *
- U 6 6 6 *
- G 7 7 7 *
- U 7 7 7 *
```

See the *AIX Operating System Commands Reference* for more information on the **cut**, **paste**, **sed**, **sort**, and **uniq** commands, and for other useful commands for modifying files.

Deleting Translation Entries

The **dsldxprof** command can simplify the process of removing a large number of translation entries from a Network Users/Groups Table. To construct a line that deletes a translation entry, append ## characters to the first field, for example:

```
##-   U   200   200   200   *
```

Suppose that you wanted to delete all the entries in a Network Users/Groups Table. By using **dsldxprof** to delete the entries, you can clear even a large table in only a few steps:

To Delete All Translation Entries

1. Use **ugtable** to access the Network Users/Groups Table. Select PRINT to copy the Network Users/Groups Table without headers to a file.

2. Use **sed** to convert each line in the file to a deletion line:

   ```
   # sed -e "s/^ */##/" idtable >rmtable
   ```

3. Use **dsldxprof** to remove the translation entries:

   ```
   # dsldxprof -f rmtable
   ```

Modifying Translation Entries

To understand how **dsldxprof** handles input lines intended to modify existing translation lines, you need to remember two rules governing the building of the Network Users/Groups Table:

1. A local ID can have more than one inbound ID.

2. A local ID can have only one outbound ID.

Suppose that you want to change both the inbound and the outbound IDs in the following translation entry from 20000000 to 30000000:

```
billd       U       200      20000000      20000000      *
```

You give **dsldxprof** the following line, specifying the **-a** flag (which instructs **dsldxprof** to replace any existing lines that conflict with the new line):

```
billd       U       200      30000000      30000000      *
```

However, **dsldxprof** does not interpret this line as telling it to change both the inbound and outbound IDs to 30000000. **dsldxprof** interprets this line as two instructions:

1. **Add** a translation entry that associates inbound ID 30000000 with local ID 200.

2. **Change** the outbound ID associated with local ID 200 from 20000000 to 30000000.

As a result, when **dsldxprof** finishes, you have two translation entries for this local ID:

```
billd       U       200      30000000      20000000      *
billd       U       200      30000000      30000000      *
```

Thus, if you want to change both the inbound and outbound IDs in one translation entry, first have **dsldxprof** delete the existing line, then add the new line:

```
##billd     U       200      20000000      20000000      *
billd       U       200      30000000      30000000      *
```

Using dsldxprof with Remote Tables

You can add, delete, and modify translation entries in a remote Network Users/Groups Table by running **dsldxprof** with the **-n** flag, for example:

```
# dsldxprof -n 20810CBF -f rmtable
```

In order to modify a remote table, you must have **system** group or superuser privileges at the remote node. In addition, the **dsxlate** command must be run at that node before your changes take effect (see "Loading Distributed Services Profiles" on page 3-42).

Appendix C. Distributed Services Customization Forms

This appendix provides forms to help define your network. Copy these forms as required for each node in the network, and then enter the information on the forms before defining the network. See "Customizing Distributed Services" on page 3-35 and Appendix B, "Getting Started With Distributed Services Customization Commands" on page B-1 for explanations of the information needed on the forms.

DEFINING A NODE

NODE: ________

Remote Nickname:	☐ None (default)	☐	____________
Remote Node ID:	____________		
Node Security:	☐ None (default)		
	☐ Secure		
	Password:[1]	____________	
Data Link Type:	☐ Ethernet (default)	☐	SDLC
Attachment Profile:	☐ Use Node ID (default)	☐	____________

[1] Do not write the password in this space if unauthorized people have access to this form. Password is either 16 hexadecimal characters or 30 to 80 alphameric characters.

INBOUND NETWORK ID TRANSLATIONS

NODE: ________

User IDs:

Network ID	Originating Node (* = Any Node)	Local ID

INBOUND NETWORK ID TRANSLATIONS

NODE: ________

Group IDs:

Network ID	Originating Node (* = Any Node)	Local ID

OUTBOUND NETWORK ID TRANSLATIONS

NODE: ________

User IDs:

User Name	Local User ID	Network ID

OUTBOUND NETWORK ID TRANSLATIONS

NODE: ________

Group IDs:

User Name	Local User ID	Network ID

FILE SYSTEM MOUNTS

NODE: ________

Mount Point (Local Path Name)	Location (nodename =)	Remote Path Name (dev =)	Auto Mount Type (type =)

Figures

Glossary

access. To obtain data from or put data in storage.

access permission. A group of designations that determine who can access a particular AIX file and how the user may access the file.

account. The log in directory and other information that give a user access to the system.

activity manager. A collection of system-supplied tasks allowing users to manage their activities. Provides the ability to list current activities (Activity List) and to begin, cancel, hide, and activate activities.

All Points Addressable (APA) display. A display that allows each pel to be individually addressed. An APA display allows for images to be displayed that are not made up of images predefined in character boxes. Contrast with *character display.*

allocate. To assign a resource, such as a disk file or a diskette file, to perform a specific task.

alphabetic. Pertaining to a set of letters a through z.

alphanumeric character. Consisting of letters, numbers and often other symbols, such as punctuation marks and mathematical symbols.

American National Standard Code for Information Interchange (ASCII). The code developed by ANSI for information interchange among data processing systems, data communications systems, and associated equipment. The ASCII character set consists of 7-bit control characters and symbolic characters.

American National Standards Institute. An organization sponsored by the Computer and Business Equipment Manufacturers Association for establishing voluntary industry standards.

application. A program or group of programs that apply to a particular business area, such as the Inventory Control or the Accounts Receivable application.

application program. A program used to perform an application or part of an application.

argument. Numbers, letters, or words that change the way a command works.

ASCII. See *American National Standard Code for Information Interchange.*

attribute. A characteristic. For example, the attribute for a displayed field could be blinking.

auto carrier return. The system function that places carrier returns automatically within the text and on the display. This is accomplished by moving whole words that exceed the line end zone to the next line.

backend. The program that sends output to a particular device. There are two types of backends: friendly and unfriendly.

background process. (1) A process that does not require operator intervention that can be run by the computer while the work station is used to do other work. (2) A mode of program execution in which the shell does not wait for program completion before prompting the user for another command.

backup copy. A copy, usually of a file or group of files, that is kept in case the original file or files are unintentionally changed or destroyed.

backup diskette. A diskette containing information copied from a fixed disk or from another diskette. It is used in case the original information becomes unusable.

bad block. A portion of a disk that can never be used reliably.

base address. The beginning address for resolving symbolic references to locations in storage.

base name. The last element to the right of a full path name. A filename specified without its parent directories.

batch printing. Queueing one or more documents to print as a separate job. The operator can type or revise additional documents at the same time. This is a background process.

batch processing. A processing method in which a program or programs process records with little or no operator action. This is a background process. Contrast with *interactive processing.*

binary. (1) Pertaining to a system of numbers to the base two; the binary digits are 0 and 1. (2) Involving a choice of two conditions, such as on-off or yes-no.

bit. Either of the binary digits 0 or 1 used in computers to store information. See also *byte.*

block. (1) A group of records that is recorded or processed as a unit. Same as *physical record.* (2) In data communications, a group of records that is recorded, processed, or sent as a unit. (3) A block is 512 bytes long. (4) A logical block is 2048 bytes long.

block file. A file listing the usage of blocks on a disk.

block special file. A special file that provides access to an input or output device is capable of supporting a file system. See also *character special file.*

bootstrap. A small program that loads larger programs during system initialization.

branch. In a computer program an instruction that selects one of two or more alternative sets of instructions. A conditional branch occurs only when a specified condition is met.

breakpoint. A place in a computer program, usually specified by an instruction, where execution may be interrupted by external intervention or by a monitor program.

buffer. (1) A temporary storage unit, especially one that accepts information at one rate and delivers it at another rate. (2) An area of storage, temporarily reserved for performing input or output, into which data is read, or from which data is written.

burst pages. On continuous-form paper, pages of output that can be separated at the perforations.

byte. The amount of storage required to represent one character; a byte is 8 bits.

call. (1) To activate a program or procedure at its entry point. Compare with *load.*

callouts. An AIX kernel parameter establishing the maximum number of scheduled activities that can be pending simultaneously.

cancel. To end a task before it is completed.

carrier return. (1) In text data, the action causing line ending formatting to be performed at the current cursor location followed by a line advance of the cursor. Equivalent to the carriage return of a typewriter. (2) A keystroke generally indicating the end of a command line.

case sensitive. Able to distinguish between uppercase and lowercase letters.

character. A letter, digit, or other symbol.

character display. A display that uses a character generator to display predefined character boxes of images (characters) on the screen. This kind of display cannot address the screen any less than one character box at a time. Contrast with *All Points Addressable display.*

character key. A keyboard key that allows the user to enter the character shown on the key. Compare with *function keys.*

character position. On a display, each location that a character or symbol can occupy.

character set. A group of characters used for a specific reason; for example, the set of characters a printer can print or a keyboard can support.

character special file. A special file that provides access to an input or output device. The character interface is used for devices that do not use block I/O. See also *block special file.*

character string. A sequence of consecutive characters.

character variable. The name of a character data item whose value may be assigned or changed while the program is running.

child. (1) Pertaining to a secured resource, either a file or library, that uses the user list of a parent resource. A child resource can have only one parent resource. (2) In the AIX Operating System, child is a *process* spawned by a parent process that shares resources of parent process. Contrast with *parent.*

C language. A general-purpose programming language that is the primary language of the AIX Operating System.

class. Pertaining to the I/O characteristics of a device. AIX devices are classified as block or character.

close. (1) To end an activity and remove that window from the display.

code. (1) Instructions for the computer. (2) To write instructions for the computer; to *program.* (3) A representation of a condition, such as an error code.

code segment. See *segment.*

collating sequence. The sequence in which characters are ordered within the computer for sorting, combining, or comparing.

color display. A display device capable of displaying more than two colors and the shades produced via the two colors, as opposed to a monochrome display.

column. A vertical arrangement of text or numbers.

column headings. Text appearing near the top of columns of data for the purpose of identifying or titling.

command. A request to perform an operation or run a program. When parameters, arguments, flags, or other operands are associated with a command, the resulting character string is a single command.

command interpreter. A program that sends instructions to the kernel; also called an interface.

command line. The area of the screen where commands are displayed as they are typed.

command line editing keys. Keys for editing the command line.

command programming language. Facility that allows programming by the combination of commands rather than by writing statements in a conventional programming language.

compile. (1) To translate a program written in a high-level programming language into a machine language program. (2) The computer actions required to transform a source file into an executable object file.

compress. (1) To move files and libraries together on disk to create one continuous area of unused space. (2) In data communications, to delete a series of duplicate characters in a character string.

concatenate. (1) To link together. (2) To join two character strings.

condition. An expression in a program or procedure that can be evaluated to a value of either true or false when the program or procedure is running.

configuration. The group of machines, devices, and programs that make up a computer system. See also *system customization.*

configuration file. A file that specifies the characteristics of a system or subsystem, for example, the AIX queueing system.

consistent. Pertaining to a file system, without internal discrepancies.

console. (1) The main AIX display station. (2) A device name associated with the main AIX display station.

constant. A data item with a value that does not change. Contrast with *variable.*

context search. A search through a file whose target is a character string.

control block. A storage area used by a program to hold control information.

control commands. Commands that allow conditional or looping logic flow in shell procedures.

control program. Part of the AIX Operating System system that determines the order in which basic functions should be performed.

controlled cancel. The system action that ends the job step being run, and saves any new data already created. The job that is running can continue with the next job step.

copy. The action by which the user makes a whole or partial duplicate of already existing data.

crash. An unexpected interruption of computer service, usually due to a serious hardware or software malfunction.

current directory. The directory that is active, and can be displayed with the **pwd** command.

current line. The line on which the cursor is located.

current working directory. See *current directory.*

cursor. (1) A movable symbol (such as an underline) on a display, used to indicate to the operator where the next typed character will be placed or where the next action will be directed. (2) A marker that indicates the current data access location within a file.

cursor movement keys. The directional keys used to move the cursor.

customize. To describe (to the system) the devices, programs, users, and user defaults for a particular data processing system.

cylinder. All fixed disk or diskette tracks that can be read or written without moving the disk drive or diskette drive read/write mechanism.

daemon. See *daemon process.*

daemon process. A process begun by the root or the root shell that can be stopped only by the root. Daemon processes generally provide services that must be available at all times such as sending data to a printer.

data block. See *block.*

data communications. The transmission of data between computers, or remote devices or both (usually over long distance).

data stream. All information (data and control information) transmitted over a data link.

debug. (1) To detect, locate, and correct mistakes in a program. (2) To find the cause of problems detected in software.

default. A value that is used when no alternative is specified by the operator.

default directory. The directory name supplied by the operating system if none is specified.

default drive. The drive name supplied by the operating system if none is specified.

default value. A value stored in the system that is used when no other value is specified.

delete. To remove. For example, to delete a file.

dependent work station. A work station having little or no standalone capability, that must be connected to a host or server in order to provide any meaningful capability to the user.

device. An electrical or electronic machine that is designed for a specific purpose and that attaches to your computer, for example, a printer, plotter, disk drive, and so forth.

device driver. A program that operates a specific device, such as a printer, disk drive, or display.

device name. A name reserved by the system that refers to a specific device.

diagnostic. Pertaining to the detection and isolation of an error.

diagnostic aid. A tool (procedure, program, reference manual) used to detect and isolate a device or program malfunction or error.

diagnostic routine. A computer program that recognizes, locates, and explains either a fault in equipment or a mistake in a computer program.

digit. Any of the numerals from 0 through 9.

directory. A type of file containing the names and controlling information for other files or other directories.

disable. To make nonfunctional.

discipline. Pertaining to the order in which requests are serviced, for example, first-come-first-served (fcfs) or shortest job next (sjn).

disk I/O. Fixed-disk input and output.

diskette. A thin, flexible magnetic plate that is permanently sealed in a protective cover. It can be used to store information copies from the disk or another diskette.

diskette drive. The mechanism used to read and write information on diskettes.

display device. An output unit that gives a visual representation of data.

display screen. The part of the display device that displays information visually.

display station. A device that includes a keyboard from which an operator can send information to the system and a display screen on which an operator can see the information sent to or received from the computer.

dump. (1) To copy the contents of all or part of storage, usually to an output device. (2) Data that has been dumped.

dump diskette. A diskette that contains a dump or is prepared to receive a dump.

dump formatter. Program for analyzing a dump.

EBCDIC. See *extended binary-coded decimal interchange code.*

EBCDIC character. Any one of the symbols included in the 8-bit EBCDIC set.

edit. To modify the form or format of data.

edit buffer. A temporary storage area used by an editor.

editor. A program used to enter and modify programs, text, and other types of documents and data.

emulation. Imitation; for example, when one computer imitates the characteristics of another computer.

enable. To make functional.

enter. To send information to the computer by pressing the **Enter** key.

entry. A single input operation on a work station.

environment. The settings for shell variables and paths set associated with each process. These variables can be modified later by the user.

error-correct backspace. An editing key that performs editing based on a cursor position; the cursor is moved one position toward the beginning of the line, the character at the new cursor location is deleted, and all characters following the cursor are moved one position toward the beginning of the line (to fill the vacancy left by the deleted element).

escape character. A character that suppresses the special meaning of one or more characters that follow.

exit value. A numeric value that a command returns to indicate whether it completed successfully. Some commands return exit values that give other information, such as whether a file exists. Shell programs can test exit values to control branching and looping.

expression. A representation of a value. For example, variables and constants appearing alone or in combination with operators.

extended binary-coded decimal interchange code (EBCDIC). A set of 256 eight-bit characters.

feature. A programming or hardware option, usually available at an extra cost.

field. (1) An area in a record or panel used to contain a particular category of data. (2) The smallest component of a record that can be referred to by a name.

FIFO. See *first-in-first-out.*

file. A collection of related data that is stored and retrieved by an assigned name.

file name. The name used by a program to identify a file. See also *label.*

filename. In DOS, that portion of the file name that precedes the extension.

file specification (filespec). The name and location of a file. A file specification consists of a drive specifier, a path name, and a file name.

file system. The collection of files and file management structures on a physical or logical mass storage device, such as a diskette or minidisk.

filetab. An AIX kernel parameter establishing the maximum number of files that can be open simultaneously.

filter. A command that reads standard input data, modifies the data, and sends it to standard output.

first-in-first-out (FIFO). A named permanent pipe. A FIFO allows two unrelated processes to exchange information using a pipe connection.

fixed disk. A flat, circular, nonremoveable plate with a magnetizable surface layer on which data can be stored by magnetic recording.

fixed-disk drive. The mechanism used to read and write information on fixed disk.

flag. A modifier that appears on a command line with the command name that defines the action of the command. Flags in the AIX Operating System almost always are preceded by a dash.

font. A family or assortment of characters of a given size and style.

foreground. A mode of program execution in which the shell waits for the program specified on the command line to complete before returning your prompt.

format. (1) A defined arrangement of such things as characters, fields, and lines, usually used for displays, printouts, or files. (2) The pattern which determines how data is recorded.

formatted diskette. A diskette on which control information for a particular computer system has been written but which may or may not contain any data.

free list. A list of available space on each file system. This is sometimes called the free-block list.

free-block list. See *free list.*

full path name. The name of any directory or file expressed as a string of directories and files beginning with the root directory.

function. A synonym for procedure. The C language treats a function as a data type that contains executable code and returns a single value to the calling function.

function keys. Keys that request actions but do not display or print characters. Included are the keys that normally produce a printed character, but when used with the code key produce a function instead. Compare with *character key.*

generation. For some remote systems, the translation of configuration information into machine language.

Gid. See *group number.*

global. Pertains to information available to more than one program or subroutine.

global action. An action having general applicability, independent of the context established by any task.

global character. The special characters * and ? that can be used in a file specification to match one or more characters. For example, placing a ? in a file specification means any character can be in that position.

global search. The process of having the system look through a document for specific characters, words, or groups of characters.

global variable. A symbol defined in one program module, but used in other independently assembled program modules.

graphic character. A character that can be displayed or printed.

group name. A name that uniquely identifies a group of users to the system.

group number (Gid). A unique number assigned to a group of related users. The group number can often be substituted in commands that take a group name as an argument.

hardware. The equipment, as opposed to the programming, of a computer system.

header. Constant text that is formatted to be in the top margin of one or more pages.

header label. A special set of records on a diskette describing the contents of the diskette.

here document. Data contained within a shell program or procedure (also called *inline input*).

highlight. To emphasize an area on the display by any of several methods, such as brightening the area or reversing the color of characters within the area.

history file. A file containing a log of system actions and operator responses.

hog factor. In system accounting, an analysis of how many times each command was run, how much processor time and memory it used, and how intensive that use was.

home directory. (1) A directory associated with an individual user. (2) The user's current directory on login or after issuing the **cd** command with no argument.

I/O. See *input/output.*

ID. Identification.

IF expressions. Expressions within a procedure, used to test for a condition.

indirect block. A block containing pointers to other blocks. Indirect blocks can be single-indirect, double-indirect, or triple-indirect.

informational message. A message providing information to the operator, that does not require a response.

initial program load (IPL). The process of loading the system programs and preparing the system to run jobs. See *initialize.*

initialize. To set counters, switches, addresses, or contents of storage to zero or other starting values at the beginning of, or at prescribed points in, the operation of a computer routine.

inline input. See *here document.*

i-node. The internal structure for managing files in the system. I-nodes contain all of the information pertaining to the node, type, owner, and location of a file. A table of i-nodes is stored near the beginning of a file system.

i-number. A number specifying a particular i-node on a file system.

inodetab. An AIX kernel parameter that establishes a table in memory for storing copies of i-nodes for all active files.

input. Data to be processed.

input device. Physical devices used to provide data to a computer.

input file. A file opened by a program so that the program can read from that file.

input list. A list of variables to which values are assigned from input data.

input redirection. The specification of an input source other than the standard one.

input-output file. A file opened for input and output use.

input-output device number. A value assigned to a device driver by the guest operating system or to the virtual device by the virtual resource manager. This number uniquely identifies the device regardless of whether it is real or virtual.

input/output (I/O). Pertaining to either input, output, or both between a computer and a device.

interactive processing. A processing method in which each system user action causes response from the program or the system. Contrast with *batch processing.*

interface. A shared boundary between two or more entities. An interface might be a hardware component to link two devices together or it might be a portion of storage or registers accessed by two or more computer programs.

interleave factor. Specification of the ratio between contiguous physical blocks (on a fixed-disk) and logically contiguous blocks (as in a file).

interrupt. (1) To temporarily stop a process. (2) In data communications, to take an action at a receiving station that causes the sending station to end a transmission. (3) A signal sent by an I/O device to the processor when an error has occurred or when assistance is needed to complete I/O. An interrupt usually suspends execution of the currently executing program.

IPL. See *initial program load.*

job. (1) A unit of work to be done by a system. (2) One or more related procedures or programs grouped into a procedure.

job queue. A list, on disk, of jobs waiting to be processed by the system.

justify. To print a document with even right and left margins.

kbuffers. An AIX kernel parameter establishing the number of buffers that can be used by the kernel.

K-byte. See *kilobyte.*

kernel. The memory-resident part of the AIX Operating System containing functions needed immediately and frequently. The kernel supervises the input and output, manages and controls the hardware, and schedules the user processes for execution.

kernel parameters. Variables that specify how the kernel allocates certain system resources.

key pad. A physical grouping of keys on a keyboard (for example, numeric key pad, and cursor key pad).

keyboard. An input device consisting of various keys allowing the user to input data, control cursor and pointer locations, and to control the dialog between the user and the display station

keylock feature. A security feature in which a lock and key can be used to restrict the use of the display station.

keyword. One of the predefined words of a programming language; a reserved word.

keyword argument. One type of variable assignment that can be made on the command line.

kill. An AIX Operating System command that stops a process.

kill character. The character that is used to delete a line of characters entered after the user's prompt.

kilobyte. 1024 bytes.

kprocs. An AIX kernel parameter establishing the maximum number of processes that the kernel can run simultaneously.

label. (1) The name in the disk or diskette volume table of contents that identifies a file. See also *file name.* (2) The field of an instruction that assigns a symbolic name to the location at which the instruction begins, or such a symbolic name.

left margin. The area on a page between the left paper edge and the leftmost character position on the page.

left-adjust. The process of aligning lines of text at the left margin or at a tab setting such that the leftmost character in the line or filed is in the leftmost position. Contrast with *right-adjust.*

library. A collection of functions, calls, subroutines, or other data.

licensed program product (LPP). Software programs that remain the property of the manufacturer, for which customers pay a license fee.

line editor. An editor that modifies the contents of a file one line at a time.

linefeed. An ASCII character that causes an output device to move forward one line.

link. A connection between an i-node and one or more file names associated with it.

literal. A symbol or a quantity in a source program that is itself data, rather than a reference to data.

load. (1) To move data or programs into storage. (2) To place a diskette into a diskette drive, or a magazine into a diskette magazine drive. (3) To insert paper into a printer.

loader. A program that reads run files into main storage, thus preparing them for execution.

local. Pertaining to a device directly connected to your system without the use of a communications line. Contrast with *remote.*

log. To record; for example, to log all messages on the system printer. A list of this type is called a log, such as an error log.

log in. To begin a session at a display station.

log in shell. The program, or command interpreter, started for a user at log in.

log off. To end a session at a display station.

log out. To end a session at a display station.

logical device. A file for conducting input or output with a physical device.

loop. A sequence of instructions performed repeatedly until an ending condition is reached.

main storage. The part of the processing unit where programs are run.

maintenance system. A special version of the AIX Operating System which is loaded from diskette and used to perform system management tasks.

major device number. A system identification number for each device or type of device.

mapped files. Files on the fixed-disk that are accessed as if they are in memory.

mask. A pattern of characters that controls the keeping, deleting, or testing of portions of another pattern of characters.

matrix. An array arranged in rows and columns.

maxprocs. An AIX kernel parameter establishing the maximum number of processes that can be run simultaneously by a user.

memory. Storage on electronic chips. Examples of memory are random access memory, read only memory, or registers. See *storage.*

menu. A displayed list of items from which an operator can make a selection.

message. (1) A response from the system to inform the operator of a condition which may affect further processing of a current program. (2) Information sent from one user in a multi-user operating system to another.

minidisk. A logical division of a fixed disk.

minor device number. A number used to specify various types of information about a particular device, for example, to distinguish among several printers of the same type.

mode word. An i-node field that describes the type and state of the i-node.

modem. See *modulator-demodulator.*

modulation. Changing the frequency or size of one signal by using the frequency or size of another signal.

modulator-demodulator (modem). A device that converts data from the computer to a signal that can be transmitted on a communications line, and converts the signal received to data for the computer.

module. (1) A discrete programming unit that usually performs a specific task or set of tasks. Modules are subroutines and calling programs that are assembled separately, then linked to make a complete program. (2) See *load module.*

mount. To make a file system accessible.

mountab. An AIX kernel parameter establishing the maximum number of file systems that can be mounted simultaneously.

multiprogramming. The processing of two or more programs at the same time.

multivolume file. A diskette file occupying more than one diskette.

nest. To incorporate a structure or structures of some kind into a structure of the same kind. For example, to nest one loop (the nested loop) within another loop (the nesting loop); to nest one subroutine (the nested subroutine) within another subroutine (the nesting subroutine).

network. A collection of products connected by communication lines for information exchange between locations.

new-line character. A control character that causes the print or display position to move to the first position on the next line.

null. Having no value, containing nothing.

null character (NUL). The character hex 00, used to represent the absence of a printed or displayed character.

numeric. Pertaining to any of the digits 0 through 9.

object code. Machine-executable instruction, usually generated by a compiler from source code written in a higher level language. consists of directly executable machine code. For programs that must be linked, object code consists of relocatable machine code.

octal. A base eight numbering system.

open. (1) To make a file available to a program for processing.

operating system. Software that controls the running of programs; in addition, an operating system may provide services such as resource allocation, scheduling, input/output control, and data management.

operation. A specific action (such as move, add, multiply, load) that the computer performs when requested.

operator. A symbol representing an operation to be done.

output. The result of processing data.

output devices. Physical devices used by a computer to present data to a user.

output file. A file that is opened by a program so that the program can write to that file.

output redirection. The specification of an output destination other than the standard one.

override. (1) A parameter or value that replaces a previous parameter or value. (2) To replace a parameter or value.

overwrite. To write output into a storage or file space that is already occupied by data.

owner. The user who has the highest level of access authority to a data object or action, as defined by the object or action.

pad. To fill unused positions in a field with dummy data, usually zeros or blanks.

page. A block of instructions, data, or both.

page space minidisk. The area on a fixed disk that temporarily stores instructions or data currently being run. See also *minidisk.*

pagination. The process of adjusting text to fit within margins and/or page boundaries.

paging. The action of transferring instructions, data, or both between real storage and external page storage.

parallel processing. The condition in which multiple tasks are being performed simultaneously within the same activity.

parameter. Information that the user supplies to a panel, command, or function.

parent. Pertaining to a secured resource, either a file or library, whose user list is shared with one or more other files or libraries. Contrast with *child.*

parent directory. The directory one level above the current directory.

partition. See *minidisk.*

password. A string of characters that, when entered along with a user identification, allows an operator to sign on to the system.

password security. A program product option that helps prevent the unauthorized use of a display station, by checking the password entered by each operator at sign-on.

path name. See *full path name* and *relative path name.*

pattern-matching character. Special characters such as * or ? that can be used in search patterns. Some used in a file specification to match one or more characters. For example, placing a ? in a file specification means any character can be in that position. Pattern-matching characters are also called wildcards.

permission code. A three-digit octal code, or a nine-letter alphabetic code, indicating the access permissions. The access permissions are read, write, and execute.

permission field. One of the three-character fields within the permissions column of a directory listing indicating the read, write, and run permissions for the file or directory owner, group, and all others.

phase. One of several stages file system checking and repair performed by the **fsck** command.

physical device. See *device.*

physical file. An indexed file containing data for which one or more alternative indexes have been created.

physical record. (1) A group of records recorded or processed as a unit. Same as *block.* (2) A unit of data moved into or out of the computer.

PID. See *process ID.*

pipe. To direct the data so that the output from one process becomes the input to another process.

pipeline. A direct, one-way connection between two or more processes.

pitch. A unit of width of typewriter type, based on the number of times a letter can be set in a linear inch. For example, 10-pitch type has 10 characters per inch.

platen. The support mechanism for paper on a printer, commonly cylindrical, against which printing mechanisms strike to produce an impression.

pointer. A logical connection between physical blocks.

port. (1) To make the programming changes necessary to allow a program that runs on one type of computer to run on another type of computer. (2) An access point for data input to or data output from a computer system. See *connector.*

position. The location of a character in a series, as in a record, a displayed message, or a computer printout.

positional parameter. A shell facility for assigning values from the command line to variables in a program.

print queue. A file containing a list of the names of files waiting to be printed.

printout. Information from the computer produced by a printer.

priority. The relative ranking of items. For example, a job with high priority in the job queue will be run before one with medium or low priority.

priority number. A number that establishes the relative priority of printer requests.

privileged user. The account with superuser authority.

problem determination. The process of identifying why the system is not working. Often this process identifies programs, equipment, data communications facilities, or user errors as the source of the problem.

problem determination procedure. A prescribed sequence of steps aimed at recovery from, or circumvention of, problem conditions.

procedure. See *shell procedure.*

process. (1) A sequence of actions required to produce a desired result. (2) An entity receiving a portion of the processor's time for executing a program. (3) An activity within the system begun by entering a command, running a shell program, or being started by another process.

process accounting. An analysis of the use each process makes of the processing unit, memory, and I/O resources.

process ID (PID). A unique number assigned to a process that is running.

profile. (1) A file containing customized settings for a system or user (2) Data describing the significant features of a user, program, or device.

program. A file containing a set of instructions conforming to a particular programming language syntax.

prompt. A displayed request for information or operator action.

propagation time. The time necessary for a signal to travel from one point on a communications line to another.

qdaemon. The daemon process that maintains a list of outstanding jobs and sends them to the specified device at the appropriate time.

queue. A line or list formed by items waiting to be processed.

queued message. A message from the system that is added to a list of messages stored in a file for viewing by the user at a later time. This is in contrast to a message that is sent directly to the screen for the user to see immediately.

quit. A key, command, or action that tells the system to return to a previous state or stop a process.

quote. To mask the special meaning of certain characters; to cause them to be taken literally.

random access. An access mode in which records can be read from, written to, or removed from a file in any order.

readonly. Pertaining to file system mounting, a condition that allows data to be read, but not modified.

recovery procedure. (1) An action performed by the operator when an error message appears on the display screen. Usually, this action permits the program to continue or permits the operator to run the next job. (2) The method of returning the system to the point where a major system error occurred and running the recent critical jobs again.

redirect. To divert data from a process to a file or device to which it would not normally go.

reference count. In an i-node, a record of the total number of directory entries that refer to the i-node.

relational expression. A logical statement describing the relationship (such as greater than or equal) of two arithmetic expressions or data items.

relational operator. The reserved words or symbols used to express a relational condition or a relational expression.

relative address. An address specified relative to the address of a symbol. When a program is relocated, the addresses themselves will change, but the specification of relative addresses remains the same.

relative addressing. A means of addressing instructions and data areas by designating their locations relative to some symbol.

relative path name. The name of a directory or file expressed as a sequence of directories followed by a file name, beginning from the current directory.

remote. Pertaining to a system or device that is connected to your system through a communications line. Contrast with *local.*

reserved character. A character or symbol that has a special (non-literal) meaning unless quoted.

reserved word. A word that is defined in a programming language for a special purpose, and that must not appear as a user-declared identifier.

reset. To return a device or circuit to a clear state.

restore. To return to an original value or image. For example, to restore a library from diskette.

right adjust. The process of aligning lines of text at the right margin or tab setting such that the rightmost character in the line or file is in the rightmost position.

right justify. See right align.

right margin. The area on a page between the last text character and the right upper edge.

right-adjust. To place or move an entry in a field so that the rightmost character of the field is in the rightmost position. Contrast with *left-adjust.*

root. Another name sometimes used for superuser.

root directory. The top level of a tree-structured directory system.

root file system. The basic AIX Operating System file system, which contains operating system files and onto which other file systems can be mounted. The root file system is the file system that contains the files that are run to start the system running.

routine. A set of statements in a program causing the system to perform an operation or a series of related operations.

run. To cause a program, utility, or other machine function to be performed.

run-time environment. A collection of subroutines and shell variables that provide commonly used functions and information for system components.

scratch file. A file, usually used as a work file, that exists until the program that uses it ends.

screen. See *display screen.*

scroll. To move information vertically or horizontally to bring into view information that is outside the display screen boundaries.

sector. (1) An area on a disk track or a diskette track reserved to record information. (2) The smallest amount of information that can be written to or read from a disk or diskette during a single read or write operation.

security. The protection of data, system operations, and devices from accidental or intentional ruin, damage, or exposure.

segment. A contiguous area of virtual storage allocated to a job or system task. A program segment can be run by itself, even if the whole program is not in main storage.

separator. A character used to separate parts of a command or file.

sequential access. An access method in which records are read from, written to, or removed from a file based on the logical order of the records in the file.

session records. In the accounting system, a record of time connected and line usage for connected display stations, produced from log in and log out records.

set flags. Flags that can be put into effect with the shell set command.

shared printer. A printer that is used by more than one work station.

shell. See *shell program.*

shell procedure. A series of commands combined in a file that carry out a particular function when the file is run or when the file is specified as an argument to the sh command. Shell procedures are frequently called shell scripts.

shell program. A program that accepts and interprets commands for the operating system (there is an AIX shell program and a DOS shell program).

shell prompt. The character string on the command line indicating the the system can accept a command (typically the $ character).

shell script. See *shell* procedure.

shell variables. Facilities of the shell program for assigning variable values to constant names.

size field. In an i-node, a field that indicates the size, in bytes, of the file associated with the i-node.

software. Programs.

sort. To rearrange some or all of a group of items based upon the contents or characteristics of those items.

source diskette. The diskette containing data to be copied, compared, restored, or backed up.

source program. A set of instructions written in a programming language, that must be translated to machine language compiled before the program can be run.

special character. A character other than an alphabetic or numeric character. For example; *, +, and % are special characters.

special file. Special files are used in the AIX system to provide an interface to input/output devices. There is at least one special file for each device connected to the computer. Contrast with *directory* and *file.* See also *block special file* and *character special file.*

spool files. Files used in the transmission of data among devices.

standalone shell. A limited version of the shell program used for system maintenance.

standalone work station. A work station that can be used to preform tasks independent of (without being connected to) other resources such as servers or host systems.

standard error. The place where many programs place error messages.

standard input. The primary source of data going into a command. Standard input comes from the keyboard unless redirection or piping is used, in which case standard input can be from a file or the output from another command.

standard output. The primary destination of data coming from a command. Standard output goes to the display unless redirection or piping is used, in which case standard output can be to a file or another command.

stanza. A group of lines in a file that together have a common function. Stanzas are usually separated by blank lines, and each stanza has a name.

statement. An instruction in a program or procedure.

status. (1) The current condition or state of a program or device. For example, the status of a printer. (2) The condition of the hardware or software, usually represented in a status code.

storage. (1) The location of saved information. (2) In contrast to memory, the saving of information on physical devices such as disk or tape. See *memory.*

storage device. A device for storing and/or retrieving data.

string. A linear sequence of entities such as characters or physical elements. Examples of strings are alphabetic string, binary element string, bit string, character string, search string, and symbol string.

su. See *superuser.*

subdirectory. A directory contained within another directory in the file system hierarchy.

subprogram. A program invoked by another program, such as a subshell.

subroutine. (1) A sequenced set of statements that may be used in one or more computer programs and at one or more points in a computer program. (2) A routine that can be part of another routine.

subscript. An integer or variable whose value refers to a particular element in a table or an array.

subshell. An instance of the shell program started from an existing shell program.

substring. A part of a character string.

subsystem. A secondary or subordinate system, usually capable of operating independently of, or synchronously with, a controlling system.

superblock. The most critical part of the file system containing information about every allocation or deallocation of a block in the file system.

superuser (su). The user who can operate without the restrictions designed to prevent data loss or damage to the system (User ID 0).

superuser authority. The unrestricted ability to access and modify any part of the operating system associated with the user who manages the system. The authority obtained when one logs in as **root**.

system. The computer and its associated devices and programs.

system call. A request by an active process for a service by the system kernel.

system customization. A process of specifying the devices, programs, and users for a particular data processing system.

system date. The date assigned by the system user during setup and maintained by the system.

system dump. A copy of memory from all active programs (and their associated data) whenever an error stops the system. Contrast with *task dump.*

system management. The tasks involved in maintaining the system in good working order and modifying the system to meet changing requirements.

system parameters. See *kernel parameters.*

system profile. A file containing the default values used in system operations.

system unit. The part of the system that contains the processing unit, the disk drives, and the diskette drives.

system user. A person who uses a computer system.

target diskette. The diskette to be used to receive data from a source diskette.

task. A basic unit of work to be performed. Examples are a user task, a server task, and a processor task.

task dump. A copy of memory from a program that failed (and its associated data). Contrast with *system dump.*

terminal. An input/output device containing a keyboard and either a display device or a printer. Terminals usually are connected to a computer and allow a person to interact with the computer.

text. A type of data consisting of a set of linguistic characters (for example, alphabet, numbers, and symbols) and formatting controls.

text application. A program defined for the purpose of processing text data (for example, memos, reports, and letters).

text editing program. See *editor* and *text application.*

texttab. A kernel parameter establishing the size of the text table, in memory, that contains one entry each active shared program text segment.

trace. To record data that provides a history of events occurring in the system.

trace table. A storage area into which a record of the performance of computer program instructions is stored.

track. A circular path on the surface of a fixed disk or diskette on which information is magnetically recorded and from which recorded information is read.

trap. An unprogrammed, hardware-initiated jump to a specific address. Occurs as a result of an error or certain other conditions.

tree-structured directories. A method for connecting directories such that each directory is listed in another directory except for the root directory, which is at the top of the tree.

truncate. To shorten a field or statement to a specified length.

typematic key. A key that repeats its function multiple times when held down.

typestyle. Characters of a given size, style and design.

Uid. See *user number.*

update. An improvement for some part of the system.

user. The name associated with an account.

user account. See *account.*

user ID. See *user number.*

user name. A name that uniquely identifies a user to the system.

user number (Uid). (1) A unique number identifying an operator to the system. This string of characters limits the functions and information the operator is allowed to use. The Uid can often be substituted in commands that take a user's name as an argument.

user profile. A file containing a description of user characteristics and defaults (for example, printer assignment, formats, group ID) to be conveyed to the system while the user is signed on.

utility. A service; in programming, a program that performs a common service function.

valid. (1) Allowed. (2) True, in conforming to an appropriate standard or authority.

value. (1) In Usability Services, information selected or typed into a pop-up. (2) A set of characters or a quantity associated with a parameter or name. (3) In programming, the contents of a storage location.

variable. A name used to represent a data item whose value can change while the program is running. Contrast with *constant.*

verify. To confirm the correctness of something.

version. Information in addition to an object's name that identifies different modification levels of the same logical object.

virtual device. A device that appears to the user as a separate entity but is actually a shared portion of a real device. For example, several virtual terminals may exist simultaneously, but only one is active at any given time.

virtual machine. A functional simulation of a computer and its related devices.

virtual machine interface (VMI). A software interface between work stations and the operating system. The VMI shields operating system software from hardware changes and low-level interfaces and provides for concurrent execution of multiple virtual machines.

virtual resource manager (VRM). A set of programs that manage the hardware resources (main storage, disk storage, display stations, and printers) of the system so that these resources can be used independently of each other.

virtual resources. See *virtual resource manager.*

virtual storage. Addressable space that appears to be real storage. From virtual storage, instructions and data are mapped into real storage locations.

virtual terminal. Any of several logical equivalents of a display station available at a single physical display station.

Volume ID (Vol ID). A series of characters recorded on the diskette used to identify the diskette to the user and to the system.

VRM. See *virtual resource manager.*

wildcard. See *pattern-matching characters.*

word. A contiguous series of 32 bits (4 bytes) in storage, addressable as a unit. The address of the first byte of a word is evenly divisible by four.

work file. A file used for temporary storage of data being processed.

work station. A device at which an individual may transmit information to, or receive information from, a computer for the purpose of performing a task, for example, a display station or printer. See *programmable work station* and *dependent work station.*

working directory. See *current directory.*

wrap around. Movement of the point of reference in a file from the end of one line to the beginning of the next, or from one end of a file to the other.

Index

IBM

The IBM RT PC
Family

Reader's Comment Form

Managing the AIX Operating System

SX23-0793-0

Your comments assist us in improving our products. IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

For prompt resolution to questions regarding set up, operation, program support, and new program literature, contact the authorized IBM RT PC dealer in your area.

Comments:

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department 997, Building 998
11400 Burnet Rd.
Austin, Texas 78758

Fold and tape

Fold and tape

Cut or Fold Along Line

IBM RT PC Managing the AIX Operating System — **SC23-0793**

Book Title — **Order No.**

Book Evaluation Form

Your comments can help us produce better books. You may use this form to communicate your comments about this book, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you. Please take a few minutes to evaluate this book as soon as you become familiar with it. Circle Y (Yes) or N (No) for each question that applies and give us any information that may improve this book.

Y N Is the purpose of this book clear?

Y N Is the table of contents helpful?

Y N Is the index complete?

Y N Are the chapter titles and other headings meaningful?

Y N Is the information organized appropriately?

Y N Is the information accurate?

Y N Is the information complete?

Y N Is only necessary information included?

Y N Does the book refer you to the appropriate places for more information?

Y N Are terms defined clearly?

Y N Are terms used consistently?

Y N Are the abbreviations and acronyms understandable?

Y N Are the examples clear?

Y N Are examples provided where they are needed?

Y N Are the illustrations clear?

Y N Is the format of the book (shape, size, color) effective?

Other Comments

What could we do to make this book or the entire set of books for this system easier to use?

Optional Information

Your name ____________

Company name ____________

Street address ____________

City, State, ZIP ____________

No postage necessary if mailed in the U.S.A.

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department 997, Building 998
11400 Burnet Rd.
Austin, Texas 78758

Fold and tape

Fold and tape

Cut or Fold Along Line

International Business
Machines Corporation
Department 997, Building 998
11400 Burnet Rd.
Austin, Texas 78758

Printed in the
United States of America

SC23-0793-0

®

SC23-0793-00

92X1265