Bull

AIX 5L System Management Concepts
Operating System and Devices

AIX

ORDER REFERENCE
86 A2 48EM 02

Bull

AIX 5L System Management Concepts Operating System and Devices

AIX

Software

October 2005

BULL CEDOC
357 AVENUE PATTON
B.P.20845
49008 ANGERS CEDEX 01
FRANCE

ORDER REFERENCE
86 A2 48EM 02

Printed in France

Suggestions and criticisms concerning the form, content, and presentation of this book are invited. A form is provided at the end of this book for this purpose.

To order additional copies of this book or other Bull Technical Publications, you are invited to use the Ordering Form also provided at the end of this book.

Trademarks and Acknowledgements

We acknowledge the right of proprietors of trademarks mentioned in this book.

AIX® is a registered trademark of International Business Machines Corporation, and is being used under licence.

UNIX is a registered trademark in the United States of America and other countries licensed exclusively through the Open Group.

Linux is a registered trademark of Linus Torvalds.

Contents

About This Book

This book provides system administrators with conceptual information that can affect your selection of options when performing such tasks as backing up and restoring the system, managing physical and logical storage, sizing appropriate paging space, and so on. This book is intended to be used as a companion to the System Management Guide: Operating System and Devices. This publication is also available on the documentation CD that is shipped with the operating system.

How to Use This Book

This book is organized to provide overview information and, when needed, conceptual details for major elements of the AIX operating system. Only general descriptions of basic tasks are provided. For task instructions, see the *AIX 5L Version 5.3 System Management Guide: Operating System and Devices*.

Beginning with the AIX 5.2 documentation library, any information that this book contained regarding AIX system security, or any security-related topic, has moved. For all security-related information, see the *AIX 5L Version 5.3 Security Guide*.

Highlighting

The following highlighting conventions are used in this book:

Bold	Identifies commands, subroutines, keywords, files, structures, directories, and other items whose names are predefined by the system. Also identifies graphical objects such as buttons, labels, and icons that the user selects.
Italics	Identifies parameters whose actual names or values are to be supplied by the user.
`Monospace`	Identifies examples of specific data values, examples of text similar to what you might see displayed, examples of portions of program code similar to what you might write as a programmer, messages from the system, or information you should actually type.

Case-Sensitivity in AIX

Everything in the AIX® operating system is case-sensitive, which means that it distinguishes between uppercase and lowercase letters. For example, you can use the **ls** command to list files. If you type `LS`, the system responds that the command is ″not found.″ Likewise, **FILEA**, **FiLea**, and **filea** are three distinct file names, even if they reside in the same directory. To avoid causing undesirable actions to be performed, always ensure that you use the correct case.

ISO 9000

ISO 9000 registered quality systems were used in the development and manufacturing of this product.

Related Publications

- *AIX 5L Version 5.3 System Management Guide: Operating System and Devices*
- *AIX 5L Version 5.3 Security Guide*
- *AIX 5L Version 5.3 Installation Guide and Reference*
- *AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs*
- *AIX 5L Version 5.3 Communications Programming Concepts*
- *AIX 5L Version 5.3 Kernel Extensions and Device Support Programming Concepts*
- *AIX 5L Version 5.3 Files Reference*
- *Performance Toolbox Version 2 and 3 for AIX: Guide and Reference*

- *AIX 5L Version 5.3 Performance Management Guide*
- *Common Desktop Environment 1.0: Advanced User's and System Administrator's Guide*.

Chapter 1. System Management

System management is the task of an individual who is usually referred to, in UNIX® literature, as the system administrator. Unfortunately, only a few system administrator activities are straightforward enough to be correctly called administration. This and related guides are intended to help system administrators with their numerous duties.

Topics covered in this chapter are:

Unique Aspects of System Management in AIX

This operating system provides its own particular version of system-management support in order to promote ease of use and to improve security and integrity. This chapter presents information on these unique features:

Available System Management Interfaces

In addition to conventional command line system administration, this operating system provides the following interfaces:

- System Management Interface Tool (SMIT), a menu-based user interface that constructs commands from the options you choose and executes them.

  With SMIT, you can:

  – Install, update, and maintain software
  – Configure devices
  – Configure disk storage units into volume groups and logical volumes
  – Make and extend file systems and paging space
  – Manage users and groups
  – Configure networks and communication applications
  – Print
  – Perform problem determination
  – Schedule jobs
  – Manage system resources and workload
  – Manage system environments
  – Manage cluster system data

  See Chapter 13, "System Management Interface Tool," on page 121 for more information on managing your system with SMIT.
- Web-based System Manager , an object-oriented graphical user interface that supports the same system management tasks as SMIT, but eases system management tasks by:
  – Reducing user errors through error checking and dialog design
  – Offering step-by-step procedures for new or complex tasks
  – Offering advanced options for more experienced administrators
  – Making it easier to visualize complex data or relationships among system objects

– Monitoring system activity and alerting the administrator when predefined events occur
– Providing context-sensitive helps, overviews, tips, and links to online documentation

Web-based System Manager can be configured to run in a variety of operating modes. The operating environments in which it can be started are standalone application, client-server, applet, and remote client. See Chapter 12, “Web-based System Manager,” on page 119 for more information about managing your system with Web-based System Manager.

Unique Features of the Operating System

Following are brief discussions of unique system management features of the operating system.

Logical Volume Manager

The Logical Volume Manager (LVM) maintains the hierarchy of logical structures that manage disk storage. Disk drives are defined within this hierarchy as physical volumes. Every physical volume in use belongs to a volume group. Within each volume group, one or more logical volumes of information are defined. Data on logical volumes appears to be contiguous to the user, but can be discontiguous on the physical volume. This allows file systems, paging space, and other logical volumes to be resized or relocated, span multiple physical volumes, and have their contents replicated for greater flexibility and availability.

For more detailed information, see the “Logical Volume Storage Overview” on page 11.

System Resource Controller

The System Resource Controller (SRC) provides a set of commands and subroutines for creating and controlling subsystems and is designed to minimize the need for human intervention in system processing. It provides a mechanism to control subsystem processes by using a command-line or C interface. This allows you to start, stop, and collect status information on subsystem processes with shell scripts, commands, or user-written programs.

For more detailed information, see the “System Resource Controller Overview” on page 109.

Object Data Manager

The Object Data Manager (ODM) is a data manager intended for the storage of system data. Many system management functions use the ODM database. Information used in many commands and SMIT functions is stored and maintained as objects with associated characteristics. System data managed by ODM includes:

- Device configuration information
- Display information for SMIT (menus, selectors, and dialogs)
- Vital product data for installation and update procedures
- Communications configuration information
- System resource information.

Software Vital Product Data

Certain information about software products and their installable options is maintained in the Software Vital Product Data (SWVPD) database. The SWVPD consists of a set of commands and Object Data Manager (ODM) object classes for the maintenance of software product information. The SWVPD commands are provided for the user to query (**lslpp**) and verify (**lppchk**) installed software products. The ODM object classes define the scope and format of the software product information that is maintained.

The **installp** command uses the ODM to maintain the following information in the SWVPD database:

- Name of the installed software product
- Version of the software product

- Release level of the software product, which indicates changes to the external programming interface of the software product
- Modification level of the software product, which indicates changes that do not affect the external programming interface of the software product
- Fix level of the software product, which indicates small updates that are to be built into a regular modification level at a later time
- Fix identification field
- Names, checksums, and sizes of the files that make up the software product or option
- Installation state of the software product: applying, applied, committing, committed, rejecting, or broken.

Workload Manager (WLM)

Workload Manager (WLM) lets you create different classes of service for jobs, as well as specify attributes for those classes. These attributes specify minimum and maximum amounts of CPU, physical memory, and disk I/O throughput to be allocated to a class. WLM then assigns jobs automatically to classes using class assignment rules provided by a system administrator. These assignment rules are based on the values of a set of attributes for a process. Either the system administrator or a privileged user can also manually assign jobs to classes, overriding the automatic assignment. For more information, see Chapter 9, “Workload Management,” on page 79.

Operating System Updates

The operating system package is divided into filesets, where each fileset contains a group of logically related customer deliverable files. Each fileset can be individually installed and updated.

Revisions to filesets are tracked using the version, release, maintenance, and fix (VRMF) levels. By convention, each time an AIX fileset update is applied, the fix level is adjusted. Each time an AIX maintenance level is applied, the modification level is adjusted, and the fix level is reset to zero. The initial installation of an AIX version, for example, AIX 5.2, is called a base installation. The operating system provides updates to its features and functionality, which might be packaged as a maintenance level, a program temporary fix (PTF), or a recommended maintenance package.

Maintenance Levels
: A maintenance level provides new functionality that is intended to upgrade the release. The maintenance part of the VRMF is updated in a maintenance level. For example, the first maintenance level for AIX 5.2 would be 5.2.1.0; the second would be , 5.2.2.0, and so forth.

PTFs
: Between releases, you might receive PTFs to correct or prevent a particular problem. A particular installation might need some, all, or even none of the available PTFs.

Recommended Maintenance Packages
: A recommended maintenance package is a set of PTFs between maintenance levels that have been extensively tested together and are recommended for preventive maintenance.

Determining What Is Installed

To determine the maintenance level installed on a particular system, type:

```
oslevel
```

To determine which filesets need updating for the system to reach a specific maintenance level (in this example, 4.3.3.0), use the following command:

```
oslevel -l 4.3.3.0
```

To determine if a recommended maintenance package is installed (in this example, 5100-02), use the following command:

```
oslevel -r 5100-02
```

To determine which filesets need updating for the system to reach the 5100-02 level, use the following command:

```
oslevel -rl 5100-02
```

To determine the level of a particular fileset (in this example, **bos.mp**), use the following command:

```
lslpp -L bos.mp
```

man Command

The **man** command is used mainly to access reference information on commands, subroutines, and files. For example, to view information on the **ls** command, enter:

```
>man ls
```

Most of the information displayed is actually taken from formatted HTML files. Many system managers find using the **man** command more convenient than starting a web browser session when they simply need to find out about a certain flag or the syntax of a given command.

For more information on the **man** command, see the *AIX 5L Version 5.3 Commands Reference*. Also see “Online Documentation and man Command for BSD 4.3 System Managers” on page 165.

Chapter 2. Starting and Stopping the System

This chapter explains system startup activities such as booting, creating boot images or files for starting the system, and setting the system run level. The chapter also focuses on stopping the system using the **reboot** and **shutdown** commands.

Topics covered in this chapter are:

Starting the System

When the base operating system starts, the system initiates a complex set of tasks. Under normal conditions, these tasks are performed automatically.

Booting the System

There are some situations when you want to instruct the system to reboot; for example, to cause the system to recognize newly installed software, to reset peripheral devices, to perform routine maintenance tasks like checking file systems, or to recover from a system hang or crash. For information on these procedures, see:

- Booting an Uninstalled System
- Rebooting a Running System
- Booting a System That Crashed.

Creating Boot Images

When the system is first installed, the **bosboot** command creates a boot image from a RAM (random access memory) disk file system image and the operating system kernel. The boot image is transferred to a particular media such as the hard disk. When the machine is rebooted, the boot image is loaded from the media into memory.

For more information, see "Creating Boot Images" in the*AIX 5L Version 5.3 System Management Guide: Operating System and Devices*.

Identifying and Changing the System Run Level

The system run level specifies the system state and defines which processes are started. For example, when the system run level is 3, all processes defined to operate at that run level are started. Near the end of the system boot phase of the boot process, the run level is read from the initdefault entry of the **/etc/inittab** file. The system run level can be changed with the **init** command. The **/etc/inittab** file contains a record for each process that defines run levels for that process. When the system boots, the **init** command reads the **/etc/inittab** file to determine which processes to start. For information on these procedures, see:

- Identifying System Run Levels
- Changing System Run Levels
- Changing the /etc/inittab File.

Understanding the Boot Process

During the boot process, the system tests the hardware, loads and runs the operating system, and configures devices. To boot the operating system, the following resources are required:

- A *boot image* that can be loaded after the machine is turned on or reset.
- Access to the root (**/**) and **/usr** file systems.

There are three types of system boots:

Hard Disk Boot	A machine is started for normal operations. For more information, see "Understanding System Boot Processing."
Diskless Network Boot	A diskless or dataless workstation is started remotely over a network. A machine is started for normal operations. One or more remote file servers provide the files and programs that diskless or dataless workstations need to boot.
Maintenance Boot	A machine is started from a hard disk, network, tape, or CD-ROM in maintenance mode. A system administrator can perform tasks such as installing new or updated software and running diagnostic checks. For more information, see "Understanding the Maintenance Boot Process" on page 8.

During a hard disk boot, the boot image is found on a local disk created when the operating system was installed. During the boot process, the system configures all devices found in the machine and initializes other basic software required for the system to operate (such as the Logical Volume Manager). At the end of this process, the file systems are mounted and ready for use. For more information about the file system used during boot processing, see "Understanding the RAM File System" on page 8.

The same general requirements apply to diskless network clients. They also require a boot image and access to the operating system file tree. Diskless network clients have no local file systems and get all their information by way of remote access.

Understanding System Boot Processing

Most users perform a hard disk boot when starting the system for general operations. The system finds all information necessary to the boot process on its disk drive.

When the system is started by turning on the power switch (a cold boot) or restarted with the **reboot** or **shutdown** commands (a warm boot), a number of events must occur before the system is ready for use. These events can be divided into the following phases:

1. Read Only Storage (ROS) Kernel Init Phase
2. Base Device Configuration Phase
3. Maintenance Boot Phase.

ROS Kernel Init Phase

The ROS kernel resides in firmware. Its initialization phase involves the following steps:

1. The firmware checks to see if there are any problems with the system board. Control is passed to ROS, which performs a power-on self-test (POST).
2. The ROS initial program load (IPL) checks the user boot list, a list of available boot devices. This boot list can be altered to suit your requirements using the **bootlist** command. If the user boot list in non-volatile random access memory (NVRAM) is not valid or if a valid boot device is not found, the default boot list is then checked. In either case, the first valid boot device found in the boot list is used for system startup. If a valid user boot list exists in NVRAM, the devices in the list are checked in order. If no user boot list exists, all adapters and devices on the bus are checked. In either case, devices are checked in a continuous loop until a valid boot device is found for system startup.

Note: The system maintains a default boot list located in ROS and a user boot list stored in NVRAM, for a normal boot. Separate default and user boot lists are also maintained for booting from the Service key position.

3. When a valid boot device is found, the first record or program sector number (PSN) is checked. If it is a valid boot record, it is read into memory and is added to the IPL control block in memory. Included in the key boot record data are the starting location of the boot image on the boot device, the length of the boot image, and instructions on where to load the boot image in memory.
4. The boot image is read sequentially from the boot device into memory starting at the location specified in NVRAM. The disk boot image consists of the kernel, a RAM file system, and base customized device information.
5. Control is passed to the kernel, which begins system initialization.
6. The kernel runs **init**, which runs phase 1 of the **rc.boot** script.

When the kernel initialization phase is completed, base device configuration begins.

Base Device Configuration Phase

The **init** process starts the **rc.boot** script. Phase 1 of the **rc.boot** script performs the base device configuration, and it includes the following steps:

1. The boot script calls the **restbase** program to build the customized Object Data Manager (ODM) database in the RAM file system from the compressed customized data.
2. The boot script starts the configuration manager, which accesses phase 1 ODM configuration rules to configure the base devices.
3. The configuration manager starts the **sys**, **bus**, **disk**, SCSI, and the Logical Volume Manager (LVM) and rootvg volume group configuration methods.
4. The configuration methods load the device drivers, create special files, and update the customized data in the ODM database.

System Boot Phase

The System Boot Phase involved the following steps:

1. The **init** process starts phase 2 running of the **rc.boot** script. Phase 2 of **rc.boot** includes the following steps:
   a. Call the **ipl_varyon** program to vary on the rootvg volume group.
   b. Mount the hard disk file systems onto their normal mount points.
   c. Run the **swapon** program to start paging.
   d. Copy the customized data from the ODM database in the RAM file system to the ODM database in the hard disk file system.
   e. Exit the **rc.boot** script.
2. After phase 2 of **rc.boot**, the boot process switches from the RAM file system to the hard disk root file system.
3. Then the **init** process runs the processes defined by records in the **/etc/inittab** file. One of the instructions in the **/etc/inittab** file runs phase 3 of the **rc.boot** script, which includes the following steps:
   a. Mount the **/tmp** hard disk file system.
   b. Start the configuration manager phase 2 to configure all remaining devices.
   c. Use the **savebase** command to save the customized data to the boot logical volume
   d. Exit the **rc.boot** script.

At the end of this process, the system is up and ready for use.

Understanding the Maintenance Boot Process

Occasions might arise when a boot is needed to perform special tasks such as installing new or updated software, performing diagnostic checks, or for maintenance. In this case, the system starts from a bootable medium such as a CD-ROM, tape drive, network, or disk drive.

The maintenance boot sequence of events is similar to the sequence of a normal boot.

1. The firmware checks to see if there are any problems with the system board.
2. Control is passed to ROS, which performs a power-on self-test.
3. ROS checks the user boot list. You can use the **bootlist** command to alter the user boot list to suit your requirements. If the user boot list in NVRAM is not valid or if no valid boot device is found, the default boot list is checked. In either case, the first valid boot device found in the boot list is used for system startup.

   Note: For a normal boot, the system maintains a default boot list located in ROS, and a user boot list stored in NVRAM. Separate default and user boot lists are also maintained for booting in maintenance mode.
4. When a valid boot device is found, the first record or program sector number (PSN) is checked. If it is a valid boot record, it is read into memory and is added to the initial program load (IPL) control block in memory. Included in the key boot record data are the starting location of the boot image on the boot device, the length of the boot image, and the offset to the entry point to start running when the boot image is in memory.
5. The boot image is read sequentially from the boot device into memory, starting at the location specified in NVRAM.
6. Control is passed to the kernel, which begins running programs in the RAM file system.
7. The ODM database contents determine which devices are present, and the **cfgmgr** command dynamically configures all devices found, including all disks which are to contain the root file system.
8. If a CD-ROM, tape, or the network is used to boot the system, the rootvg volume group (or rootvg) is not varied on, because the rootvg might not exist (as is the case when installing the operating system on a new system). Network configuration can occur at this time. No paging occurs when a maintenance boot is performed.

At the end of this process, the system is ready for installation, maintenance, or diagnostics.

Note: If the system is booted from the hard disk, the rootvg is varied on, the hard disk root file system and the hard disk user file system are mounted in the RAM file system, a menu is displayed that allows you to enter various diagnostics modes or single-user mode. Selecting single-user mode allows the user to continue the boot process and enter single-user mode, where the **init** run level is set to ″S″. The system is then ready for maintenance, software updates, or running the **bosboot** command.

Understanding the RAM File System

The RAM file system, part of the boot image, is totally memory-resident and contains all programs that allow the boot process to continue. The files in the RAM file system are specific to the type of boot.

A maintenance boot RAM file system might not have the logical volume routines, because the rootvg might not need to be varied on. During a hard disk boot, however, it is desirable that the rootvg be varied on and paging activated as soon as possible. Although there are differences in these two boot scenarios, the structure of the RAM file system does not vary to a great extent.

The **init** command on the RAM file system used during boot is actually the simple shell (**ssh**) program. The **ssh** program controls the boot process by calling the **rc.boot** script. The first step for **rc.boot** is to determine from what device the machine was booted. The boot device determines which devices are to be

configured on the RAM file system. If the machine is booted over the network, the network devices need to be configured so that the client file systems can be remotely mounted. In the case of a tape or CD-ROM boot, the console is configured to display the BOS installation menus. After the **rc.boot** script identifies the boot device, then the appropriate configuration routines are called from the RAM file system. The **rc.boot** script itself is called twice by the **ssh** program to match the two configuration phases during boot. A third call to **rc.boot** occurs during a disk or a network boot when the real **init** command is called. The **inittab** file contains an **rc.boot** stanza that does the final configuration of the machine.

The RAM file system for each boot device is also unique because of the various types of devices to be configured. A prototype file is associated with each type of boot device. The prototype file is a template of files making up the RAM file system. The **bosboot** command uses the **mkfs** command to create the RAM file system using the various prototype files. See the **bosboot** command for more details.

Understanding the Shutdown Process

There are several controlled situations in which you might want to shut down your system:

- After installing new software or changing the configuration for existing software
- When a hardware problem exists
- When the system is irrevocably hung
- When system performance is degraded
- When the file system is possibly corrupt.

System Hang Detection

AIX can detect system hang conditions and try to recover from such situations, based on user defined actions. Currently AIX supports two types of hang detections.

- Priority Hang Detection
- Lost I/O Hang Detection

The following sections describe these types. For more information on system hang detection, see System Hang Management in *AIX 5L Version 5.3 System Management Guide: Operating System and Devices*.

Priority Hang Detection

All processes (also known as threads) run at a priority. This priority is numerically inverted in the range 0-126. Zero is highest priority and 126 is the lowest priority. The default priority for all threads is 60. The priority of a process can be lowered by any user with the **nice** command. Anyone with root authority can also raise a process's priority.

The kernel scheduler always picks the highest priority runnable thread to put on a CPU. It is therefore possible for a sufficient number of high priority threads to completely tie up the machine such that low priority threads can never run. If the running threads are at a priority higher than the default of 60, this can lock out all normal shells and logins to the point where the system appears hung.

The System Hang Detection feature provides a mechanism to detect this situation and allow the system administrator a means to recover. This feature is implemented as a daemon (**shdaemon**) that runs at the highest process priority. This daemon queries the kernel for the lowest priority thread run over a specified interval. If the priority is above a configured threshold, the daemon can take one of several actions. Each of these actions can be independently enabled, and each can be configured to trigger at any priority and over any time interval. The actions and their defaults are:

```
    Action           Default    Default     Default   Default
                     Enabled    Priority    Timeout   Device

1)  Log an error        no         60          2
2)  Console message     no         60          2      /dev/console
```

```
3)  High priority       yes        60          2      /dev/tty0
    login shell
4)  Run a command at    no         60          2
    high priority
5)  Crash and reboot    no         39          5
```

Lost I/O Hang Detection

Because of I/O errors, the I/O path can become blocked and further I/O on that path is affected. In these circumstances it is essential that the operating system alert the user and execute user defined actions. As part of the Lost I/O detection and notification, the **shdaemon**, with the help of the Logical Volume Manager, monitors the I/O buffers over a period of time and checks whether any I/O is pending for too long a period of time. If the wait time exceeds the threshold wait time defined by the **shconf** file, a lost I/O is detected and further actions are taken. The information about the lost I/O is documented in the error log. Also based on the settings in the **shconf** file, the system might be rebooted to recover from the lost I/O situation.

For lost I/O detection, you can set the time out value and also enable the following actions:

Action	Default Enabled	Default Device
Console message	no	/dev/console
Crash and reboot	no	-

Chapter 3. Logical Volumes

This chapter describes concepts for managing logical volume storage and covers the following topics:

Logical Volume Storage Overview

A hierarchy of structures is used to manage disk storage. Each individual disk drive, called a *physical volume* (PV) has a name, such as `/dev/hdisk0`. Every physical volume in use belongs to a *volume group* (VG). All of the physical volumes in a volume group are divided into *physical partitions* (PPs) of the same size. For space-allocation purposes, each physical volume is divided into five regions (outer_edge, inner_edge, outer_middle, inner_middle and center). The number of physical partitions in each region varies, depending on the total capacity of the disk drive.

Within each volume group, one or more *logical volumes* (LVs) are defined. Logical volumes are groups of information located on physical volumes. Data on logical volumes appears to be contiguous to the user but can be discontiguous on the physical volume. This allows file systems, paging space, and other logical volumes to be resized or relocated, to span multiple physical volumes, and to have their contents replicated for greater flexibility and availability in the storage of data.

Each logical volume consists of one or more *logical partitions* (LPs). Each logical partition corresponds to at least one physical partition. If mirroring is specified for the logical volume, additional physical partitions are allocated to store the additional copies of each logical partition. Although the logical partitions are numbered consecutively, the underlying physical partitions are not necessarily consecutive or contiguous.

Logical volumes can serve a number of system purposes, such as paging, but each logical volume serves a single purpose only. Many logical volumes contain a single journaled file system (JFS or JFS2). Each JFS consists of a pool of page-size (4 KB) blocks. When data is to be written to a file, one or more additional blocks are allocated to that file. These blocks might not be contiguous with one another or with other blocks previously allocated to the file. A given file system can be defined as having a fragment size of less than 4 KB (512 bytes, 1 KB, 2 KB).

After installation, the system has one volume group (the rootvg volume group) consisting of a base set of logical volumes required to start the system and any other logical volumes you specify to the installation script. Any other physical volumes you have connected to the system can be added to a volume group (using the **extendvg** command). You can add the physical volume either to the rootvg volume group or to another volume group (defined by using the **mkvg** command). Logical volumes can be tailored using the commands, the menu-driven System Management Interface Tool (SMIT) interface, or Web-based System Manager.

Logical Volume Storage Concepts

The basic logical storage concepts are as follows:

The following figure illustrates the relationships among these concepts.

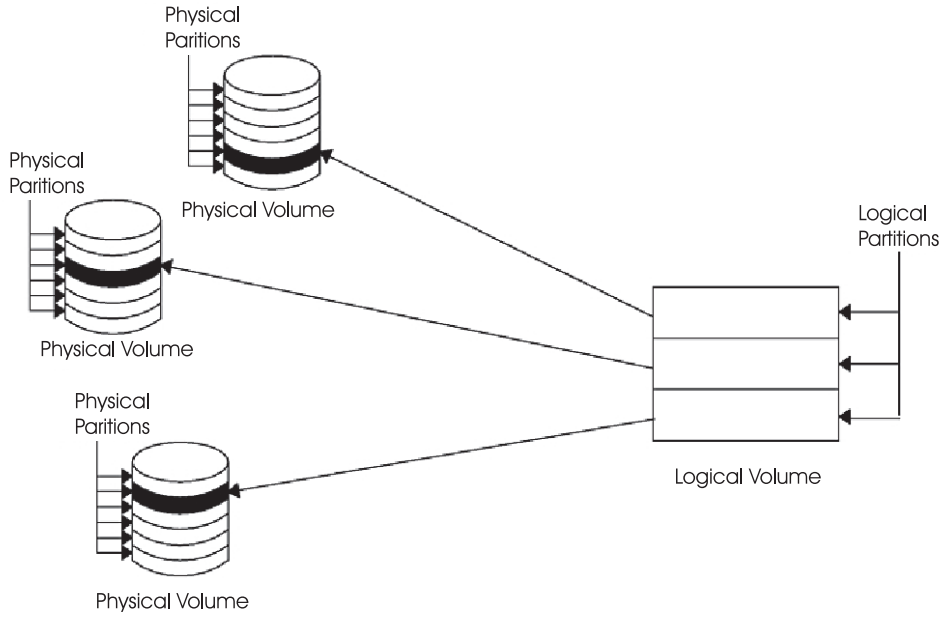

Figure 1. Volume Group. This illustration shows a volume group composed of three physical volumes with the maximum range specified. The logical volume (which can span physical volumes) is composed of logical partitions allocated onto physical partitions.

Physical Volumes

A disk must be designated as a physical volume and be put into an available state before it can be assigned to a volume group. A physical volume has certain configuration and identification information written on it. This information includes a physical volume identifier that is unique to the system.

In AIX 5.2 and later versions, the LVM can make use of the additional space that a redundant array of identical disks (RAID) can add to a logical unit number (LUN), by adding physical partitions to the physical volume associated with the LUN.

Volume Groups

The physical volume must now become part of a *volume group*. A normal volume group is a collection of 1 to 32 physical volumes of varying sizes and types. A big volume group can have from 1 to 128 physical volumes. A scalable volume group can have up to 1024 physical volumes. A physical volume can belong to only one volume group per system; there can be up to 255 active volume groups.

When a physical volume is assigned to a volume group, the physical blocks of storage media on it are organized into physical partitions of a size you specify when you create the volume group. For more information, see “Physical Partitions” on page 13.

When you install the system, one volume group (the root volume group, called rootvg) is automatically created that contains the base set of logical volumes required to start the system, as well as any other logical volumes you specify to the installation script. The rootvg includes paging space, the journal log, boot data, and dump storage, each in its own separate logical volume. The rootvg has attributes that differ

from user-defined volume groups. For example, the rootvg cannot be imported or exported. When performing a command or procedure on the rootvg, you must be familiar with its unique characteristics.

You create a volume group with the **mkvg** command. You add a physical volume to a volume group with the **extendvg** command, make use of the changed size of a physical volume with the **chvg** command, and remove a physical volume from a volume group with the **reducevg** command. Some of the other commands that you use on volume groups include: list (**lsvg**), remove (**exportvg**), install (**importvg**), reorganize (**reorgvg**), synchronize (**syncvg**), make available for use (**varyonvg**), and make unavailable for use (**varyoffvg**).

Small systems might require only one volume group to contain all the physical volumes attached to the system. You might want to create separate volume groups, however, for security reasons, because each volume group can have its own security permissions. Separate volume groups also make maintenance easier because groups other than the one being serviced can remain active. Because the rootvg must always be online, it contains only the minimum number of physical volumes necessary for system operation.

You can move data from one physical volume to other physical volumes *in the same volume group* with the **migratepv** command. This command allows you to free a physical volume so it can be removed from the volume group. For example, you could move data from a physical volume that is to be replaced.

A volume group that is created with smaller physical and logical volume limits can be converted to a format which can hold more physical volumes and more logical volumes. This operation requires that there be enough free partitions on every physical volume in the volume group for the volume group descriptor area (VGDA) expansion. The number of free partitions required depends on the size of the current VGDA and the physical partition size. Because the VGDA resides on the edge of the disk and it requires contiguous space, the free partitions are required on the edge of the disk. If those partitions are allocated for a user's use, they are migrated to other free partitions on the same disk. The rest of the physical partitions are renumbered to reflect the loss of the partitions for VGDA usage. This renumbering changes the mappings of the logical to physical partitions in all the physical volumes of this volume group. If you have saved the mappings of the logical volumes for a potential recovery operation, generate the maps again after the completion of the conversion operation. Also, if the backup of the volume group is taken with map option and you plan to restore using those maps, the restore operation might fail because the partition number might no longer exist (due to reduction). It is recommended that backup is taken before the conversion and right after the conversion if the map option is used. Because the VGDA space has been increased substantially, every VGDA update operation (creating a logical volume, changing a logical volume, adding a physical volume, and so on) might take considerably longer to run.

Physical Partitions

When you add a physical volume to a volume group, the physical volume is partitioned into contiguous, equal-sized units of space called *physical partitions*. A physical partition is the smallest unit of storage space allocation and is a contiguous space on a physical volume.

Physical volumes inherit the volume group's physical partition size, which you can set only when you create the volume group (for example, using the **mkvg -s** command). The following illustration shows the relationship between physical partitions on physical volumes and volume groups.

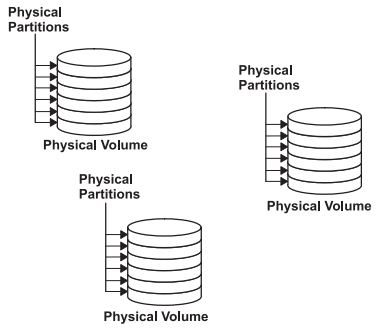

Figure 2. A Volume Group Containing Three Physical Volumes. This illustration shows three physical volumes, each with six physical partitions, within a single volume group.

Logical Volumes

After you create a volume group, you can create logical volumes within that volume group. A *logical volume,* although it can reside on noncontiguous physical partitions or even on more than one physical volume, appears to users and applications as a single, contiguous, extensible disk volume. You can create additional logical volumes with the **mklv** command. This command allows you to specify the name of the logical volume and define its characteristics, including the number and location of logical partitions to allocate for it.

After you create a logical volume, you can change its name and characteristics with the **chlv** command, and you can increase the number of logical partitions allocated to it with the **extendlv** command. The default maximum size for a logical volume at creation is 512 logical partitions, unless specified to be larger. The **chlv** command is used to override this limitation.

> **Note:** After you create a logical volume, the characteristic LV STATE, which can be seen using the **lslv** command, is closed. It becomes open when, for example, a file system has been created in the logical volume and the logical volume is mounted.

Logical volumes can also be copied with the **cplv** command, listed with the **lslv** command, removed with the **rmlv** command, and have the number of copies they maintain increased or decreased with the **mklvcopy** and the **rmlvcopy** commands, respectively. Logical Volumes can also be relocated when the volume group is reorganized.

The system allows you to define up to 255 logical volumes per standard volume group (511 for a big volume group and 4095 for a scalable volume group), but the actual number you can define depends on the total amount of physical storage defined for that volume group and the size of the logical volumes you define.

Logical Partitions

When you create a logical volume, you specify the number of *logical partitions* for the logical volume. A logical partition is one, two, or three physical partitions, depending on the number of instances of your data you want maintained. Specifying one instance means there is only one copy of the logical volume (the default). In this case, there is a direct mapping of one logical partition to one physical partition. Each instance, including the first, is termed a *copy*. Where physical partitions are located (that is, how near each other physically) is determined by options you specify when you create the logical volume.

File Systems

The logical volume defines allocation of disk space down to the physical-partition level. Finer levels of data management are accomplished by higher-level software components such as the Virtual Memory Manager or the file system. Therefore, the final step in the evolution of a disk is the creation of *file systems*. You

can create one file system per logical volume. To create a file system, use the **crfs** command. For more information on file systems, see Chapter 5, “File Systems,” on page 35.

Limitations for Logical Storage Management

The following table shows the limitations for logical storage management. Although the default maximum number of physical volumes per volume group is 32 (128 for a big volume group, 1024 for a scalable volume group), you can set the maximum for user-defined volume groups when you use the **mkvg** command. For the rootvg, however, this variable is automatically set to the maximum by the system during the installation.

Limitations for Logical Storage Management

Category	Limit
Volume group	255 per system
Physical volume	(MAXPVS/volume group factor) per volume group. MAXPVS is 32 for a standard volume group, 128 for a big volume group, and 1024 for a scalable volume group.
Physical partition	Normal and Big Volume groups: (1016 x volume group factor) per physical volume up to 1024 MB each in size. Scalable volume groups: 2097152 partitions up to 128 GB in size. There is no volume group factor for scalable volume groups.
Logical volume	MAXLVS per volume group, which is 255 for a standard volume group, 511 for a big volume group, and 4095 for a scalable volume group.

If you previously created a volume group before the 1016 physical partitions per physical volume restriction was enforced, stale partitions (no longer containing the most current data) in the volume group are not correctly tracked unless you convert the volume group to a supported state. You can convert the volume group with the **chvg -t** command. A suitable factor value is chosen by default to accommodate the largest disk in the volume group.

For example, if you created a volume group with a 9 GB disk and 4 MB partition size, this volume group will have approximately 2250 partitions. Using a conversion factor of 3 (1016 * 3 = 3048) allows all 2250 partitions to be tracked correctly. Converting a standard or big volume group with a higher factor enables inclusion of a larger disk of partitions up to the 1016* factor. You can also specify a higher factor when you create the volume group in order to accommodate a larger disk with a small partition size.

These operations reduce the total number of disks that you can add to a volume group. The new maximum number of disks you can add would be a MAXPVS/factor. For example, for a regular volume group, a factor of 2 decreases the maximum number of disks in the volume group to 16 (32/2). For a big volume group, a factor of 2 decreases the maximum number of disks in the volume group to 64 (128/2). For a scalable volume group, a factor of 2 decreases the maximum number of disks in the volume group to 512 (1024/2).

Logical Volume Manager

The set of operating system commands, library subroutines, and other tools that allow you to establish and control logical volume storage is called the Logical Volume Manager (LVM). The LVM controls disk resources by mapping data between a more simple and flexible *logical* view of storage space and the actual *physical* disks. The LVM does this using a layer of device-driver code that runs above traditional disk device drivers.

The LVM consists of the logical volume device driver (LVDD) and the LVM subroutine interface library. The *logical volume device driver* (LVDD) is a pseudo-device driver that manages and processes all I/O. It translates logical addresses into physical addresses and sends I/O requests to specific device drivers. The

LVM subroutine interface library contains routines that are used by the system management commands to perform system management tasks for the logical and physical volumes of a system.

For more information about how LVM works, see Understanding the Logical Volume Device Driver in *AIX 5L Version 5.3 Kernel Extensions and Device Support Programming Concepts* and Logical Volume Programming Overview in *AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs*.

Quorum Concepts

The following sections describe the vary-on process and the quorum, which are the mechanisms that the LVM uses to ensure that a volume group is ready to use and contains the most up-to-date data.

Vary-On Process

The **varyonvg** and **varyoffvg** commands activate or deactivate (make available or unavailable for use) a volume group that you have defined to the system. The volume group must be varied on before the system can access it. During the vary-on process, the LVM reads management data from the physical volumes defined in the volume group. This management data, which includes a volume group descriptor area (VGDA) and a volume group status area (VGSA), is stored on each physical volume of the volume group.

The VGDA contains information that describes the mapping of physical partitions to logical partitions for each logical volume in the volume group, as well as other vital information, including a time stamp. The VGSA contains information such as which physical partitions are stale and which physical volumes are missing (that is, not available or active) when a vary-on operation is attempted on a volume group.

If the vary-on operation cannot access one or more of the physical volumes defined in the volume group, the command displays the names of all physical volumes defined for that volume group and their status. This helps you decide whether to vary-off this volume group.

Quorum

A quorum is a vote of the number of Volume Group Descriptor Areas and Volume Group Status Areas (VGDA/VGSA) that are active. A quorum ensures data integrity of the VGDA/VGSA areas in the event of a disk failure. Each physical disk in a volume group has at least one VGDA/VGSA. When a volume group is created onto a single disk, it initially has two VGDA/VGSA areas residing on the disk. If a volume group consists of two disks, one disk still has two VGDA/VGSA areas, but the other disk has one VGDA/VGSA. When the volume group is made up of three or more disks, then each disk is allocated just one VGDA/VGSA.

A quorum is lost when enough disks and their VGDA/VGSA areas are unreachable so that a 51% majority of VGDA/VGSA areas no longer exists. In a two-disk volume group, if the disk with only one VGDA/VGSA is lost, a quorum still exists because two of the three VGDA/VGSA areas still are reachable. If the disk with two VGDA/VGSA areas is lost, this statement is no longer true. The more disks that make up a volume group, the lower the chances of quorum being lost when one disk fails.

When a quorum is lost, the volume group varies itself off so that the disks are no longer accessible by the LVM. This prevents further disk I/O to that volume group so that data is not lost or assumed to be written when physical problems occur. Additionally, as a result of the vary-off, the user is notified in the error log that a hardware error has occurred and service must be performed.

There are cases when it is desirable to continue operating the volume group even though a quorum is lost. In these cases, quorum checking can be turned off for the volume group. This type of volume group is referred to as a *nonquorum volume group*. The most common case for a nonquorum volume group occurs when the logical volumes have been mirrored. When a disk is lost, the data is not lost if a copy of the logical volume resides on a disk that is not disabled and can be accessed. However, there can be instances in nonquorum volume groups, mirrored or nonmirrored, when the data (including copies) resides

on the disk or disks that have become unavailable. In those instances, the data might not be accessible even though the volume group continues to be varied on.

Nonquorum Volume Groups

The Logical Volume Manager (LVM) automatically deactivates the volume group when it lacks a quorum of VGDAs or VGSAs. However, you can choose an option that allows the group to stay online as long as there is one VGDA/VGSA intact. This option produces a *nonquorum volume group.* The LVM requires access to all of the disks in nonquorum volume groups before allowing reactivation to ensure up-to-date VGDA and VGSA.

You might want to use this procedure in systems where every logical volume has at least two copies. If a disk failure occurs, the volume group remains active as long as there is one active disk.

Note: Both user-defined and rootvg volume groups can operate in nonquorum status, but the methods used to configure user-defined volume groups and rootvg volume groups as nonquorum and for recovery after hardware failures are different. Be sure you use the correct method for the appropriate volume group.

Developing a Volume Group Strategy

Disk failure is the most common hardware failure in the storage system, followed by failure of adapters and power supplies. Protection against disk failure primarily involves the configuration of the logical volumes. See “Developing a Logical Volume Strategy” on page 18 for more information. Volume group size can also play a part.

To protect against adapter and power supply failure, consider a special hardware configuration for any specific volume group. Such a configuration includes two adapters and at least one disk per adapter, with mirroring across adapters, and a nonquorum volume group configuration. The additional expense of this configuration is not appropriate for all sites or systems. It is recommended only where high (up-to-the-last second) availability is a priority. Depending on the configuration, high availability can cover hardware failures that occur between the most recent backup and the current data entry. High availability does not apply to files deleted by accident.

When to Create Separate Volume Groups

You might want to organize physical volumes into volume groups separate from rootvg for the following reasons:

- For safer and easier maintenance.
  - Operating system updates, reinstallations, and crash recoveries are safer because you can separate user file systems from the operating system so that user files are not jeopardized during these operations.
  - Maintenance is easier because you can update or reinstall the operating system without having to restore user data. For example, before updating, you can remove a user-defined volume group from the system by unmounting its file systems. Deactivate it using the **varyoffvg** command, then export the group using the **exportvg** command. After updating the system software, you can reintroduce the user-defined volume group using the **importvg** command, then remount its file systems.
- For different physical-partition sizes. All physical volumes within the same volume group must have the same physical partition size. To have physical volumes with different physical partition sizes, place each size in a separate volume group.
- When different quorum characteristics are required. If you have a file system for which you want to create a nonquorum volume group, maintain a separate volume group for that data; all of the other file systems should remain in volume groups operating under a quorum.
- For security. For example, you might want to remove a volume group at night.
- To switch physical volumes between systems. If you create a separate volume group for each system on an adapter that is accessible from more than one system, you can switch the physical volumes

between the systems that are accessible on that adapter without interrupting the normal operation of either (see the **varyoffvg**, **exportvg**, **importvg**, and **varyonvg** commands).

High Availability in Case of Disk Failure

The primary methods used to protect against disk failure involve logical volume configuration settings, such as mirroring. While the volume group considerations are secondary, they have significant economic implications because they involve the number of physical volumes per volume group:

- The quorum configuration, which is the default, keeps the volume group active (varied on) as long as a quorum (51%) of the disks is present. For more information about requirements, see “Vary-On Process” on page 16. In most cases, you need at least three disks with mirrored copies in the volume group to protect against disk failure.
- The nonquorum configuration keeps the volume group active (varied on) as long as one VGDA is available on a disk (see ″Changing a Volume Group to Nonquorum Status″ in *AIX 5L Version 5.3 System Management Guide: Operating System and Devices*). With this configuration, you need only two disks with mirrored copies in the volume group to protect against disk failure.

When deciding on the number of disks in each volume group, you must also plan for room to mirror the data. Keep in mind that you can only mirror and move data between disks that are in the same volume group. If the site uses large file systems, finding disk space on which to mirror could become a problem at a later time. Be aware of the implications on availability of inter-disk settings for logical volume copies (see “Inter-Disk Settings for Logical Volume Copies” on page 23) and intra-disk allocation (see “Choosing an Intra-Disk Allocation Policy for Each Logical Volume” on page 24) for a logical volume.

High Availability in Case of Adapter or Power Supply Failure

To protect against adapter or power supply failure, depending on your requirements, do one or more of the following:

- Use two adapters, located in the same or different chassis. Locating the adapters in different chassis protects against losing both adapters if there is a power supply failure in one chassis.
- Use two adapters, attaching at least one disk to each adapter. This protects against a failure at either adapter (or power supply if adapters are in separate cabinets) by still maintaining a quorum in the volume group, assuming *cross-mirroring* (copies for a logical partition cannot share the same physical volume) between the logical volumes on disk A (adapter A) and the logical volumes on disk B (adapter B). This means that you copy the logical volumes that reside on the disks attached to adapter A to the disks that reside on adapter B and also that you copy the logical volumes that reside on the disks attached to adapter B to the disks that reside on adapter A as well.
- Configure all disks from both adapters into the same volume group. This ensures that at least one logical volume copy remains intact in case an adapter fails, or, if cabinets are separate, in case a power supply fails.
- Make the volume group a nonquorum volume group. This allows the volume group to remain active as long as one Volume Group Descriptor Area (VGDA) is accessible on any disk in the volume group. See Changing a Volume Group to Nonquorum Status for more information.
- If there are two disks in the volume group, implement cross-mirroring between the adapters. If more than one disk is available on each adapter, implement double-mirroring. In that case, you create a mirrored copy on a disk that uses the same adapter and one on a disk using a different adapter.

Developing a Logical Volume Strategy

The policies described in this section help you set a strategy for logical volume use that is oriented toward a combination of availability, performance, and cost that is appropriate for your site.

Availability is the ability to access data even if its associated disk has failed or is inaccessible. The data might remain accessible through copies of the data that are made and maintained on separate disks and adapters during normal system operation. Techniques such as mirroring and the use of hot spare disks can help ensure data availability.

Performance is the average speed at which data is accessed. Policies such as write-verify and mirroring enhance availability but add to the system processing load, and thus degrade performance. Mirroring doubles or triples the size of the logical volume. In general, increasing availability degrades performance. Disk striping can increase performance. Beginning with AIX 4.3.3, disk striping is allowed with mirroring. Beginning with AIX 5.1, you can detect and remedy hot-spot problems that occur when some of the logical partitions on your disk have so much disk I/O that your system performance noticeably suffers.

By controlling the allocation of data on the disk and between disks, you can tune the storage system for the highest possible performance. See *AIX 5L Version 5.3 Performance Management Guide* for detailed information on how to maximize storage-system performance.

Use the following sections to evaluate the trade-offs among performance, availability, and cost. Remember that increased availability often decreases performance, and vice versa. Mirroring may increase performance, however, if the LVM chooses the copy on the least busy disk for Reads.

> **Note:** Mirroring does not protect against the loss of individual files that are accidentally deleted or lost because of software problems. These files can only be restored from conventional tape or disk backups.

Analyzing Needs for Mirroring or Striping

Determine whether the data that is stored in the logical volume is valuable enough to warrant the processing and disk-space costs of mirroring. If you have a large sequential-access file system that is performance-sensitive, you may want to consider disk striping.

Performance and mirroring are not always opposed. If the different instances (copies) of the logical partitions are on different physical volumes, preferably attached to different adapters, the LVM can improve Read performance by reading the copy on the least busy disk. Write performance, unless disks are attached to different adapters, always cost the same because you must update all copies. It is only necessary to read one copy for a Read operation.

AIX LVM supports the following RAID options:

Table 1. Logical Volume Manager Support for RAID

RAID 0	Striping
RAID 1	Mirroring
RAID 10 or 0+1	Mirroring and striping

While mirroring improves storage system availability, it is not intended as a substitute for conventional tape backup arrangements.

You can mirror the rootvg, but if you do, create a separate dump logical volume. Dumping to a mirrored logical volume can result in an inconsistent dump. Also, because the default dump device is the primary paging logical volume, create a separate dump logical volume if you mirror your paging logical volumes.

Normally, whenever data on a logical partition is updated, all the physical partitions containing that logical partition are automatically updated. However, physical partitions can become *stale* (no longer containing the most current data) because of system malfunctions or because the physical volume was unavailable at the time of an update. The LVM can refresh stale partitions to a consistent state by copying the current data from an up-to-date physical partition to the stale partition. This process is called *mirror*

synchronization. The refresh can take place when the system is restarted, when the physical volume comes back online, or when you issue the **syncvg** command.

Any change that affects the physical partition makeup of a boot logical volume requires that you run the **bosboot** command after that change. This means that actions such as changing the mirroring of a boot logical volume require a **bosboot**.

Determining Scheduling Policy for Mirrored Writes to Disk

For data that has only one physical copy, the logical volume device driver (LVDD) translates a logical Read or Write request address into a physical address and calls the appropriate physical device driver to service the request. This single-copy or nonmirrored policy handles bad block relocation for Write requests and returns all Read errors to the calling process.

If you use mirrored logical volumes, the following scheduling policies for writing to disk can be set for a logical volume with multiple copies:

Sequential-scheduling policy
: Performs Writes to multiple copies or mirrors in order. The multiple physical partitions representing the mirrored copies of a single logical partition are designated primary, secondary, and tertiary. In sequential scheduling, the physical partitions are written to in sequence. The system waits for the Write operation for one physical partition to complete before starting the Write operation for the next one. When all write operations have been completed for all mirrors, the Write operation is complete.

Parallel-scheduling policy
: Simultaneously starts the Write operation for all the physical partitions in a logical partition. When the Write operation to the physical partition that takes the longest to complete finishes, the Write operation is completed. Specifying mirrored logical volumes with a parallel-scheduling policy might improve I/O read-operation performance, because multiple copies allow the system to direct the read operation to the least busy disk for this logical volume.

Parallel write with sequential read-scheduling policy
: Simultaneously starts the Write operation for all the physical partitions in a logical partition. The primary copy of the read is always read first. If that Read operation is unsuccessful, the next copy is read. During the Read retry operation on the next copy, the failed primary copy is corrected by the LVM with a hardware relocation. This patches the bad block for future access.

Parallel write with round robin read-scheduling policy
: Simultaneously starts the Write operation for all the physical partitions in a logical partition. Reads are switched back and forth between the mirrored copies.

Bad block policy
: Indicates whether the volume group is enabled for bad block relocation. The default value is *yes*. When the value is set to *yes* for the volume group, the bad blocks can be relocated. When the value is set to *no*, the policy overrides the logical volume settings. When the value is changed, all logical volumes continue with the previous setting. The value indicates whether or not a requested I/O must be directed to a relocated block. If the value is set to *yes*, the volume group allows bad block relocation. If the value is set to *no*, bad block allocation is not completed. The LVM performs software relocation only when hardware relocation fails. Otherwise, the LVM bad block relocation (BBR) flag has no effect.

  Note: Bad block relocation is disabled unless the bad block policy settings for volume group and logical volume are both set to *yes*.

Determine Mirror Write Consistency (MWC) Policy for a Logical Volume

When Mirror Write Consistency (MWC) is turned ON, logical partitions that might be inconsistent if the system or the volume group is not shut down properly are identified. When the volume group is varied back online, this information is used to make logical partitions consistent. This is referred to as *active MWC*.

When a logical volume is using active MWC, then requests for this logical volume are held within the scheduling layer until the MWC cache blocks can be updated on the target physical volumes. When the MWC cache blocks have been updated, the request proceeds with the physical data Write operations. Only the disks where the data actually resides must have these MWC cache blocks written to it before the Write can proceed.

When active MWC is being used, system performance can be adversely affected. Adverse effects are caused by the overhead of logging or journaling that a Write request in which a Logical Track Group (LTG) is active. The allowed LTG sizes for a volume group are 128 K, 256 K, 512 K, 1024 K, 2 MB, 4 MB, 8 MB, and 16 MB.

Note: To have an LTG size greater than 128 K, the disks contained in the volume group must support I/O requests of this size from the disk's strategy routines. The LTG is a contiguous block contained within the logical volume and is aligned on the size of the LTG. This overhead is for mirrored Writes only.

It is necessary to guarantee data consistency between mirrors only if the system or volume group crashes before the Write to all mirrors has been completed. All logical volumes in a volume group share the MWC log. The MWC log is maintained on the outer edge of each disk. Locate the logical volumes that use Active MWC at the outer edge of the disk so that the logical volume is in close proximity to the MWC log on disk.

When MWC is set to passive, the volume group logs that the logical volume has been opened. After a crash when the volume group is varied on, an automatic force sync of the logical volume is started. Consistency is maintained while the force sync is in progress by using a copy of the read recovery policy that propagates the blocks being read to the other mirrors in the logical volume. This policy is only supported on the BIG volume group type.

When MWC is turned OFF, the mirrors of a mirrored logical volume can be left in an inconsistent state in the event of a system or volume group crash. There is no automatic protection of mirror consistency. Writes outstanding at the time of the crash can leave mirrors with inconsistent data the next time the volume group is varied on. After a crash, any mirrored logical volume that has MWC turned OFF should perform a forced sync before the data within the logical volume is used. For example,

```
syncvg -f -l LTVname
```

An exception to forced sync is logical volumes whose content is only valid while the logical volume is open, such as paging spaces.

A mirrored logical volume is the same as a nonmirrored logical volume with respect to a Write. When LVM completely finishes with a Write request, the data has been written to all the drive(s) below LVM. The outcome of the Write is unknown until LVM issues an **iodone** on the Write. After this is complete, no recovery after a crash is necessary. Any blocks being written that have not been completed (**iodone**) when a machine crashes should be checked and rewritten, regardless of the MWC setting or whether they are mirrored.

Because a mirrored logical volume is the same as a nonmirrored logical volume, there is no such thing as latest data. All applications that care about data validity need to determine the validity of the data of outstanding or in-flight writes that did not complete before the volume group or system crashed, whether or not the logical volume was mirrored.

Active and passive MWC make mirrors consistent only when the volume group is varied back online after a crash by picking one mirror and propagating that data to the other mirrors. These MWC policies do not keep track of the latest data. Active MWC only keeps track of LTGs currently being written, therefore MWC does not guarantee that the latest data will be propagated to all the mirrors. Passive MWC makes mirrors consistent by going into a propagate-on-read mode after a crash. It is the application above LVM that has to determine the validity of the data after a crash. From the LVM perspective, if the application always

reissues all outstanding Writes from the time of the crash, the possibly inconsistent mirrors will be consistent when these Writes finish, (as long as the same blocks are written after the crash as were outstanding at the time of the crash).

Note: Mirrored logical volumes containing either JFS logs or file systems must be synchronized after a crash either by forced sync before use, by turning MWC on, or turning passive MWC on.

Choosing an Inter-Disk Allocation Policy for Your System

The inter-disk allocation policy specifies the number of disks on which the physical partitions of a logical volume are located. The physical partitions for a logical volume might be located on a single disk or spread across all the disks in a volume group. The following options are used with the **mklv** and **chlv** commands to determine inter-disk policy:

- The *Range* option determines the number of disks used for a single physical copy of the logical volume.
- The *Strict* option determines whether the **mklv** operation succeeds if two or more copies must occupy the same physical volume.
- The *Super Strict* option specifies that the partitions allocated for one mirror cannot share a physical volume with the partitions from another mirror.
- Striped logical volumes can only have a maximum range and a super strict inter-disk policy.

Inter-Disk Settings for a Single Copy of the Logical Volume

If you select the minimum inter-disk setting (`Range = minimum`), the physical partitions assigned to the logical volume are located on a single disk to enhance availability. If you select the maximum inter-disk setting (`Range = maximum`), the physical partitions are located on multiple disks to enhance performance. The allocation of mirrored copies of the original partitions is discussed in the following section.

For nonmirrored logical volumes, use the minimum setting to provide the greatest availability (access to data in case of hardware failure). The minimum setting indicates that one physical volume contains all the original physical partitions of this logical volume if possible. If the allocation program must use two or more physical volumes, it uses the minimum number, while remaining consistent with other parameters.

By using the minimum number of physical volumes, you reduce the risk of losing data because of a disk failure. Each additional physical volume used for a single physical copy increases that risk. A nonmirrored logical volume spread across four physical volumes is four times as likely to lose data because of one physical volume failure than a logical volume contained on one physical volume.

The following figure illustrates a minimum inter-disk allocation policy.

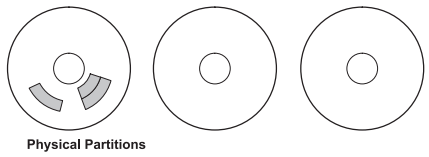

Figure 3. Minimum Inter-Disk Allocation Policy. This illustration shows three disks. One disk contains three physical partitions; the others have no physical partitions.

The maximum setting, considering other constraints, spreads the physical partitions of the logical volume as evenly as possible over as many physical volumes as possible. This is a performance-oriented option, because spreading the physical partitions over several disks tends to decrease the average access time for the logical volume. To improve availability, the maximum setting is only used with mirrored logical volumes.

The following figure illustrates a maximum inter-disk allocation policy.

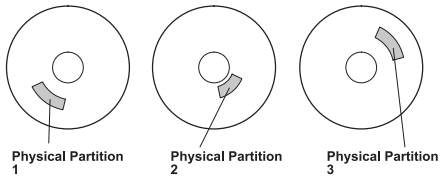

Figure 4. Maximum Inter-Disk Allocation Policy. This illustration shows three disks, each containing one physical partition.

These definitions are also applicable when extending or copying an existing logical volume. The allocation of new physical partitions is determined by your current allocation policy and where the existing used physical partitions are located.

Inter-Disk Settings for Logical Volume Copies

The allocation of a single copy of a logical volume on disk is fairly straightforward. When you create mirrored copies, however, the resulting allocation is somewhat complex. The figures that follow show minimum maximum and inter-disk (Range) settings for the first instance of a logical volume, along with the available Strict settings for the mirrored logical volume copies.

For example, if there are mirrored copies of the logical volume, the minimum setting causes the physical partitions containing the first instance of the logical volume to be allocated on a single physical volume, if possible. Then, depending on the setting of the Strict option, the additional copy or copies are allocated on the same or on separate physical volumes. In other words, the algorithm uses the minimum number of physical volumes possible, within the constraints imposed by other parameters such as the Strict option, to hold all the physical partitions.

The setting `Strict = y` means that each copy of the logical partition is placed on a different physical volume. The setting `Strict = n` means that the copies are not restricted to different physical volumes. By comparison, the `Super Strict` option would not allow any physical partition from one mirror to be on the same disk as a physical partition from another mirror of the same logical volume.

Note: If there are fewer physical volumes in the volume group than the number of copies per logical partition you have chosen, set Strict to **n**. If Strict is set to **y**, an error message is returned when you try to create the logical volume.

The following figure illustrates a minimum inter-disk allocation policy with differing Strict settings:

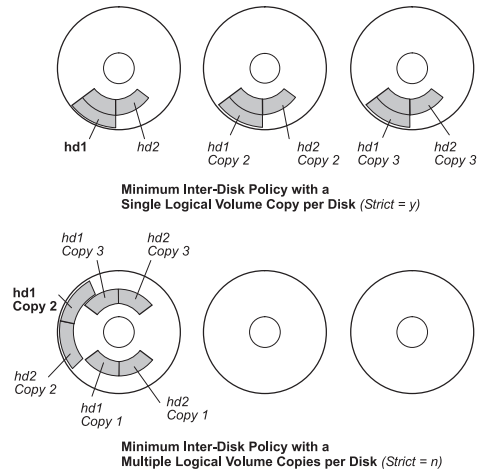

Figure 5. Minimum Inter-Disk Policy/Strict. This illustration shows that if the Strict option is equal to Yes, each copy of the logical partition is on a different physical volume. If Strict is equal to No, all copies of the logical partitions are on a single physical volume.

The following figure illustrates a maximum inter-disk allocation policy with differing *Strict* settings:

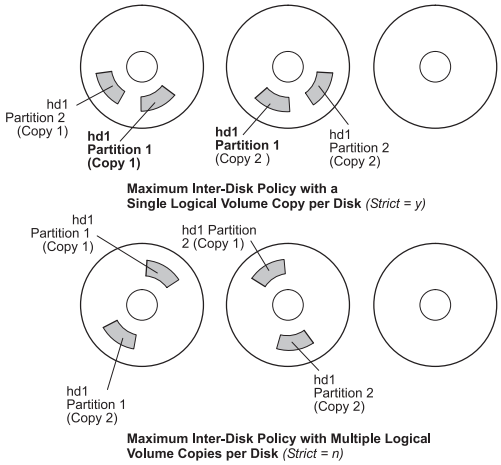

Figure 6. Maximum Inter-Disk Policy/Strict. This illustration shows that if the Strict option is equal to Yes, each copy of a partition is on a separate physical volume. If Strict is equal to No, all copies are on a single physical volume.

Choosing an Intra-Disk Allocation Policy for Each Logical Volume

The closer a given physical partition is to the center of a physical volume, the lower the average seek time is because the center has the shortest average seek distance from any other part of the disk.

The file system log is a good candidate for allocation at the center of a physical volume because it is so frequently used by the operating system. At the other extreme, the boot logical volume is used infrequently and therefore is allocated at the edge or middle of the physical volume.

The general rule is that the more I/Os, either absolutely or during the running of an important application, the closer to the center of the physical volumes the physical partitions of the logical volume needs to be allocated.

This rule has one important exception: Mirrored logical volumes with Mirror Write Consistency (MWC) set to On are at the outer edge because that is where the system writes MWC data. If mirroring is not in effect, MWC does not apply and does not affect performance.

The intra-disk allocation policy choices are based on the five regions of a disk where physical partitions can be located. The five regions are as follows:

1. outer edge
2. inner edge
3. outer middle
4. inner middle
5. center

The edge partitions have the slowest average seek times, which generally result in longer response times for any application that uses them. The center partitions have the fastest average seek times, which generally result in the best response time for any application that uses them. There are, however, fewer partitions on a physical volume at the center than at the other regions.

Combining Allocation Policies

If you select inter-disk and intra-disk policies that are not compatible, you might get unpredictable results. The system assigns physical partitions by allowing one policy to take precedence over the other. For example, if you choose an intra-disk policy of center and an inter-disk policy of minimum, the inter-disk policy takes precedence. The system places all of the partitions for the logical volume on one disk if possible, even if the partitions do not all fit into the center region. Make sure you understand the interaction of the policies you choose before implementing them.

Using Map Files for Precise Allocation

If the default options provided by the inter- and intra-disk policies are not sufficient for your needs, consider creating map files to specify the exact order and location of the physical partitions for a logical volume.

You can use Web-based System Manager, SMIT, or the **mklv -m** command to create map files.

For example, to create a ten-partition logical volume called lv06 in the rootvg in partitions 1 through 3, 41 through 45, and 50 through 60 of hdisk1, you could use the following procedure from the command line.

1. To verify that the physical partitions you plan to use are free to be allocated, type:

   ```
   lspv -p hdisk1
   ```

2. Create a file, such as `/tmp/mymap1`, containing:

   ```
   hdisk1:1-3
   hdisk1:41-45
   hdisk1:50-60
   ```

   The **mklv** command allocates the physical partitions in the order that they appear in the map file. Be sure that there are sufficient physical partitions in the map file to allocate the entire logical volume that you specify with the **mklv** command. (You can list more than you need.)

3. Type the command:

   ```
   mklv -t jfs -y lv06 -m /tmp/mymap1 rootvg 10
   ```

Developing a Striped Logical Volume Strategy

Striped logical volumes are used for large sequential file systems that are frequently accessed and performance-sensitive. Striping is intended to improve performance.

Note: A dump space or boot logical volume cannot be striped. The boot logical volume must be contiguous physical partitions.

To create a 12-partition striped logical volume called 1v07 in VGName with a stripe size of 16 KB across hdisk1, hdisk2, and hdisk3, type:

```
mklv -y lv07 -S 16K VGName 12 hdisk1 hdisk2 hdisk3
```

To create a 12-partition striped logical volume called 1v08 in VGName with a stripe size of 8 KB across any three disks within VGName, type:

```
mklv -y lv08 -S 8K -u 3 VGName 12
```

For more information on how to improve performance by using disk striping, see *AIX 5L Version 5.3 Performance Management Guide*.

Determining a Write-Verify Policy

Using the write-verify option causes all Write operations to be verified by an immediate follow-up Read operation to check the success of the Write. If the Write operation is not successful, you get an error message. This policy enhances availability but degrades performance because of the extra time needed for the Read. You can specify the use of a write-verify policy on a logical volume either when you create it using the **mklv** command, or later by changing it with the **chlv** command.

Determining Hot-Spare Disk Policies

Beginning with AIX 5.1, you can designate disks as hot-spare disks for a volume group with mirrored logical volumes. When you designate which disks to use as hot-spare disks, you can specify a policy to be used if a disk or disks start to fail and you can specify synchronization characteristics. This support complements but does not replace the sparing support available with serial storage architecture (SSA) disks. You can also use hot-spare disks with SSA disks when you add one to your volume group.

If you add a physical volume to a volume group (to mark it as a hot-spare disk), the disk must have at least the same capacity as the smallest disk already in the volume group. When this feature is implemented, data will be migrated to a hot-spare disk when Mirror Write Consistency (MWC) write failures mark a physical volume missing.

The commands to enable hot-speare disk support, **chvg** and **chpv**, provide several options in how you implement the feature at your site, as shown by the following syntax:

```
chvg -hhotsparepolicy -ssyncpolicy VolumeGroup
```

Where *hotsparepolicy* determines which of the following policies you want to use when a disk is failing:

y Automatically migrates partitions from one failing disk to one spare disk. From the pool of hot spare disks, the smallest one that is big enough to substitute for the failing disk will be used.

Y Automatically migrates partitions from a failing disk, but might use the complete pool of hot-spare disks.

n Does not migrate automatically (default).

r Removes all disks from the pool of hot-spare disks for this volume group.

The *syncpolicy* argument determines whether you want to synchronize any stale partitions automatically:

y Automatically tries to synchronize stale partitions.

n Does not automatically try to synchronize stale partitions. (This option is the default.)

The *VolumeGroup* argument specifies the name of the associated mirrored volume group.

Managing Hot Spots in Logical Volumes

Beginning with AIX 5.1, you can identify *hot spot* problems with your logical volumes and remedy those problems without interrupting the use of your system. A hot-spot problem occurs when some of the logical partitions on your disk have so much disk I/O that your system performance noticeably suffers.

The first step toward solving the problem is to identify it. By default, the system does not collect statistics for logical volume use. After you enable the gathering of these statistics, the first time you enter the

lvmstat command, the system displays the counter values since the previous system reboot. Thereafter, each time you enter the **lvmstat** command, the system displays the difference since the previous **lvmstat** command.

By interpreting the output of the **lvmstat** command, you can identify the logical partitions with the heaviest traffic. If you have several logical partitions with heavy usage on one physical disk and want to balance these across the available disks, you can use the **migratelp** command to move these logical partitions to other physical disks.

In the following example, the gathering of statistics is enabled and the **lvmstat** command is used repeatedly to gather a baseline of statistics:

```
# lvmstat -v rootvg -e
# lvmstat -v rootvg -C
# lvmstat -v rootvg
```

The output is similar to the following:

```
Logical Volume            iocnt      Kb_read     Kb_wrtn      Kbps
  hd8                         4            0          16      0.00
  paging01                    0            0           0      0.00
  lv01                        0            0           0      0.00
  hd1                         0            0           0      0.00
  hd3                         0            0           0      0.00
  hd9var                      0            0           0      0.00
  hd2                         0            0           0      0.00
  hd4                         0            0           0      0.00
  hd6                         0            0           0      0.00
  hd5                         0            0           0      0.00
```

The previous output shows that all counters have been reset to zero. In the following example, data is first copied from the **/unix** directory to the **/tmp** directory. The **lvmstat** command output reflects the activity for the rootvg:

```
# cp -p /unix /tmp
# lvmstat -v rootvg

Logical Volume            iocnt      Kb_read     Kb_wrtn      Kbps
  hd3                       296            0        6916      0.04
  hd8                        47            0         188      0.00
  hd4                        29            0         128      0.00
  hd2                        16            0          72      0.00
  paging01                    0            0           0      0.00
  lv01                        0            0           0      0.00
  hd1                         0            0           0      0.00
  hd9var                      0            0           0      0.00
  hd6                         0            0           0      0.00
  hd5                         0            0           0      0.00
```

The output shows activity on the hd3 logical volume, which is mounted in the **/tmp** directory, on hd8, which is the JFS log logical volume, on hd4, which is **/** (root), on hd2, which is the **/usr** directory, and on hd9var, which is the **/var** directory. The following output provides details for hd3 and hd2:

```
# lvmstat -l hd3

Log_part    mirror#    iocnt    Kb_read    Kb_wrtn     Kbps
       1          1      299          0       6896     0.04
       3          1        4          0         52     0.00
       2          1        0          0          0     0.00
       4          1        0          0          0     0.00
# lvmstat -l hd2
Log_part    mirror#    iocnt    Kb_read    Kb_wrtn     Kbps
       2          1        9          0         52     0.00
       3          1        9          0         36     0.00
       7          1        9          0         36     0.00
       4          1        4          0         16     0.00
```

```
 9         1         1         0         4     0.00
14         1         1         0         4     0.00
 1         1         0         0         0     0.00
```

The output for a volume group provides a summary for all the I/O activity of a logical volume. It is separated into the number of I/O requests (iocnt), the kilobytes read and written (Kb_read and Kb_wrtn, respectively), and the transferred data in KB/s (Kbps). If you request the information for a logical volume, you receive the same information, but for each logical partition separately. If you have mirrored logical volumes, you receive statistics for each of the mirror volumes. In the previous sample output, several lines for logical partitions without any activity were omitted. The output is always sorted in decreasing order on the iocnt column.

The **migratelp** command uses, as parameters, the name of the logical volume, the number of the logical partition (as it is displayed in the **lvmstat** output), and an optional number for a specific mirror copy. If information is omitted, the first mirror copy is used. You must specify the target physical volume for the move; in addition, you can specify a target physical partition number. If successful, the output is similar to the following:

```
# migratelp hd3/1 hdisk1/109
  migratelp: Mirror copy 1 of logical partition 1 of logical volume
        hd3 migrated to physical partition 109 of hdisk1.
```

After the hot spot feature is enabled, either for a logical volume or a volume group, you can define your reporting and statistics, display your statistics, select logical partitions to migrate, specify the destination physical partition, and verify the information before committing your changes. The Web-based System Manager helps you configure hot-spot reporting and manage the result. For instructions on enabling hot spot reporting, see *AIX 5L Version 5.3 System Management Guide: Operating System and Devices*.

Implementing Volume Group Policies

After you have decided which volume group policies you want to use, analyze your current configuration by typing the **lspv** command on the command line. The standard configuration provides a single volume group that includes multiple physical volumes attached to the same disk adapter and other supporting hardware. In a standard configuration, the more disks that make up a quorum volume group the better the chance of the quorum remaining when a disk failure occurs. In a nonquorum group, a minimum of two disks must make up the volume group. To implement your volume group policy changes, do the following:

1. Use the **lspv** command output to check your allocated and free physical volumes.
2. Ensure a quorum by adding one or more physical volumes. For instructions, see Adding Disks while the System Remains Available, Adding a Fixed Disk Without Data to an Existing Volume Group, or Adding a Fixed Disk Without Data to a New Volume Group.
3. To change a volume group to nonquorum status, see Changing a Volume Group to Nonquorum Status.
4. Reconfigure the hardware only if necessary to ensure high availability. For instructions, see the service guide for your system.

Chapter 4. Paging Space and Virtual Memory

AIX uses virtual memory to address more memory than is physically available in the system. The management of memory pages in RAM or on disk is handled by the Virtual Memory Manager (VMM). Virtual-memory segments are partitioned in units called *pages*. A *paging space* is a type of logical volume with allocated disk space that stores information which is resident in virtual memory but is not currently being accessed. This logical volume has an attribute type equal to paging, and is usually simply referred to as paging space or *swap space*. When the amount of free RAM in the system is low, programs or data that have not been used recently are moved from memory to paging space to release memory for other activities.

For information about managing paging spaces or virtual memory, see Paging Space and Virtual Memory in *AIX 5L Version 5.3 System Management Guide: Operating System and Devices*.

Virtual Memory Manager (VMM) Overview

The Virtual Memory Manager (VMM) services memory requests from the system and its applications. Virtual-memory segments are partitioned in units called *pages*; each page is either located in real physical memory (RAM) or stored on disk until it is needed. AIX uses virtual memory to address more memory than is physically available in the system. The management of memory pages in RAM or on disk is handled by the VMM.

Real Memory Management

In AIX, virtual-memory segments are partitioned into 4096-byte units called pages. Real memory is divided into 4096-byte page frames. The VMM has two major functions:

- Manage the allocation of page frames
- Resolve references to virtual-memory pages that are not currently in RAM (stored in paging space) or do not yet exist.

To accomplish these functions, the VMM maintains a *free list* of available page frames. The VMM also uses a page-replacement algorithm to determine which virtual-memory pages currently in RAM will have their page frames reassigned to the free list. The page-replacement algorithm takes into account the existence of persistent versus working segments, repaging, and VMM thresholds.

Free List

The VMM maintains a list of free (unallocated) page frames that it uses to satisfy page faults. AIX tries to use all of RAM all of the time, except for a small amount which it maintains on the free list. To maintain this small amount of unallocated pages the VMM uses *page outs* and *page steals* to free up space and reassign those page frames to the free list. The virtual-memory pages whose page frames are to be reassigned are selected using the VMM's page-replacement algorithm.

Persistent or Working Memory Segments

AIX distinguishes between different types of memory segments. To understand the VMM, it is important to understand the difference between working and persistent segments. A *persistent segment* has a permanent storage location on disk. Files containing data or executable programs are mapped to persistent segments. When a JFS or JFS2 file is opened and accessed, the file data is copied into RAM. VMM parameters control when physical memory frames allocated to persistent pages should be overwritten and used to store other data.

Working segments are transitory and exist only during their use by a process. Working segments have no permanent disk storage location. Process stack and data regions are mapped to working segments and shared library text segments. Pages of working segments must also occupy disk storage locations when

they cannot be kept in real memory. The disk paging space is used for this purpose. When a program exits, all of its working pages are placed back on the free list immediately.

Working Segments and Paging Space

Working pages in RAM that can be modified and paged out are assigned a corresponding slot in paging space. The allocated paging space is used only if the page needs to be paged out. However, an allocated page in paging space cannot be used by another page. It remains reserved for a particular page for as long as that page exists in virtual memory. Because persistent pages are paged out to the same location on disk from which they came, paging space does not need to be allocated for persistent pages residing in RAM.

The VMM has two modes for allocating paging space: *early* and *late*. Early allocation policy reserves paging space whenever a memory request for a working page is made. Late allocation policy only assigns paging space when the working page is actually paged out of memory, which significantly reduces the paging space requirements of the system.

VMM Memory Load Control Facility

When a process references a virtual-memory page that is on disk, because it either has been paged out or has never been read, the referenced page must be paged in, and this might cause one or more pages to be paged out if the number of available (free) page frames is low. The VMM attempts to steal page frames that have not been recently referenced and, therefore, are not likely to be referenced in the near future, using a page-replacement algorithm.

A successful page-replacement keeps the memory pages of all currently active processes in RAM, while the memory pages of inactive processes are paged out. However, when RAM is over-committed, it becomes difficult to choose pages for page out because, they will probably be referenced in the near future by currently running processes. The result is that pages that are likely to be referenced soon might still get paged out and then paged in again when actually referenced. When RAM is over-committed, continuous paging in and paging out, called *thrashing*, can occur. When a system is thrashing, the system spends most of its time paging in and paging out instead of executing useful instructions, and none of the active processes make any significant progress. The VMM has a memory load control algorithm that detects when the system is thrashing and then attempts to correct the condition.

Paging Space Overview

A *paging space* is a type of logical volume with allocated disk space that stores information which is resident in virtual memory but is not currently being accessed. This logical volume has an attribute type equal to paging, and is usually simply referred to as paging space or *swap space*. When the amount of free RAM in the system is low, programs or data that have not been used recently are moved from memory to paging space to release memory for other activities.

Another type of paging space is available that can be accessed through a device that uses an NFS server for paging-space storage. For an NFS client to access this paging space, the NFS server must have a file created and exported to that client. The file size represents the paging space size for the client.

The amount of paging space required depends on the type of activities performed on the system. If paging space runs low, processes can be lost, and if paging space runs out, the system can panic. When a paging-space low condition is detected, define additional paging space. For instructions, see Paging Space Troubleshooting in *AIX 5L Version 5.3 System Management Guide: Operating System and Devices*.

The logical volume paging space is defined by making a new paging-space logical volume or by increasing the size of existing paging-space logical volumes. To increase the size of an NFS paging space, the file that resides on the server must be increased by the correct actions on the server.

The total space available to the system for paging is the sum of the sizes of all active paging-space logical volumes.

The following sections provide more information about paging spaces:

- "Paging Space Allocation Policies"
- "Paging Space Default Size" on page 33
- "Paging Space File, Commands, and Options" on page 33

Paging Space Allocation Policies

AIX uses two modes for paging space allocation. The **PSALLOC** environment variable determines which paging space allocation algorithm is used: late or early. The default is late. You can switch to an early paging space allocation mode by changing the value of the **PSALLOC** environment variable, but there are several factors to consider before making such a change. When using the early allocation algorithm, in a worst-case scenario, it is possible to crash the system by using up all available paging space.

Comparing Late and Early Paging Space Allocation

The operating system uses the **PSALLOC** environment variable to determine the mechanism used for memory and paging space allocation. If the **PSALLOC** environment variable is not set, is set to `null`, or is set to any value other than `early`, the system uses the default `late` allocation algorithm.

The `late` allocation algorithm assists in the efficient use of disk resources and supports applications that prefer a sparse allocation algorithm for resource management. This algorithm does not reserve paging space when a memory request is made; it approves the request and assigns paging space when pages are touched. Some programs allocate large amounts of virtual memory and then use only a fraction of the memory. Examples of such programs are technical applications that use sparse vectors or matrices as data structures. The late allocation algorithm is also more efficient for a real-time, demand-paged kernel such as the one in the operating system.

However, this paging space might never be used, especially on systems with large real memory where paging is rare. Therefore, the late algorithm further delays allocation of paging space until it is necessary to page out the page, which results in no wasted paging space allocation. This does result, however, in additional overcommitment of paging space.

It is possible to overcommit resources when using the late allocation algorithm for paging space allocation. In this case, when one process gets the resource before another, a failure results. The operating system attempts to avoid complete system failure by killing processes affected by the resource overcommitment. The **SIGDANGER** signal is sent to notify processes that the amount of free paging space is low. If the paging space situation reaches an even more critical state, selected processes that did not receive the **SIGDANGER** signal are sent a **SIGKILL** signal.

You can use the **PSALLOC** environment variable to switch to an `early` allocation algorithm, which allocates paging space for the executing process at the time the memory is requested. If there is insufficient paging space available at the time of the request, the early allocation mechanism fails the memory request.

If the **PSALLOC** environment variable is set to `early`, then every program started in that environment from that point on, but not including currently running processes, runs in the early allocation environment. In the early allocation environment, interfaces such as the **malloc** subroutine and the **brk** subroutine will fail if sufficient paging space cannot be reserved when the request is made.

Processes run in the early allocation environment mode are not sent the **SIGKILL** signal if a low paging space condition occur.

There are different ways to change the **PSALLOC** environment variable to `early`, depending on how broadly you want to apply the change. (See Setting the PSALLOC Environment Variable for Early Allocation Mode in *AIX 5L Version 5.3 System Management Guide: Operating System and Devices*.)

The following memory allocation interface subroutines are affected by a switch to an early allocation environment:

- **malloc**
- **free**
- **calloc**
- **realloc**
- **brk**
- **sbrk**
- **shmget**
- **shmctl**

Early Allocation Mode Considerations: The early allocation algorithm guarantees as much paging space as requested by a memory allocation request. Thus, correct paging space allocation on the system disk is important for efficient operations. When available paging space drops below a certain threshold, new processes cannot be started and currently running processes might not be able to get more memory. Any processes running under the default late allocation mode become highly vulnerable to the **SIGKILL** signal mechanism. In addition, because the operating system kernel sometimes requires memory allocation, it is possible to crash the system by using up all available paging space.

Before you use the early allocation mode throughout the system, it is very important to define an adequate amount of paging space for the system. The paging space required for early allocation mode is almost always greater than the paging space required for the default late allocation mode. How much paging space to define depends on how your system is used and what programs you run. A good starting point for determining the right mix for your system is to define a paging space four times greater than the amount of physical memory.

Certain applications can use extreme amounts of paging space if they are run in early allocation mode. The AIXwindows server currently requires more than 250 MB of paging space when the application runs in early allocation mode. The paging space required for any application depends on how the application is written and how it is run.

All commands and subroutines that show paging space and process memory use include paging space allocated under early allocation mode. The **lsps** command uses the **-s** flag to display total paging space allocation, including paging space allocated under early allocation mode.

Programming Interface

The programming interface that controls the paging space allocation mode uses the **PSALLOC** environment variable. To ensure that an application always runs under the desired mode (with or without early paging space allocation), do the following:

1. Use the **getenv** subroutine to examine the current state of the **PSALLOC** environment variable.
2. If the value of the **PSALLOC** environment variable is not the value required by the application, use the **setenv** subroutine to alter the value of the environment variable. Because only the **execve** subroutine examines the state of the **PSALLOC** environment variable, call the **execve** subroutine with the same set of parameters and environment received by the application. When the application reexamines the state of the **PSALLOC** environment variable and finds the correct value, the application continues normally.
3. If the **getenv** subroutine reveals that the current state of the **PSALLOC** environment variable is correct, no modification is needed. The application continues normally.

Paging Space Default Size

The default paging space size is determined during the system customization phase of AIX installation according to the following standards:

- Paging space can use no less than 16 MB, except for hd6 which can use no less than 64 MB in AIX 4.3 and later.
- Paging space can use no more than 20% of total disk space.
- If real memory is less than 256 MB, paging space is two times real memory.
- If real memory is greater than or equal to 256 MB, paging space is 512 MB.

Paging Space File, Commands, and Options

The **/etc/swapspaces** file specifies the paging-space devices that are activated by the **swapon -a** command. A paging space is added to this file when it is created by the **mkps -a** command, removed from the file when it is deleted by the **rmps** command, and added or removed by the **chps -a** command. If the paging space size is too large, you can subtract logical partitions from the paging space without rebooting using the **chps -d** command.

The following commands are used to manage paging space:

chps	Changes the attributes of a paging space.
lsps	Displays the characteristics of a paging space.
mkps	Adds an additional paging space. The **mkps** command uses the **mklv** command with a specific set of options when creating a paging space logical volume. To create NFS paging spaces, the **mkps** command uses the **mkdev** command with a different set of options. For NFS paging spaces, the **mkps** command needs the host name of the NFS server and the path name of the file that is exported from the server.
rmps	Removes an inactive paging space.
swapoff	Deactivates one or more paging space without rebooting the system. Information in the paging space is moved to other active paging space areas. The deactivated paging space can then be removed using the **rmps** command.
swapon	Activates a paging space. The **swapon** command is used during early system initialization to activate the initial paging-space device. During a later phase of initialization, when other devices become available, the **swapon** command is used to activate additional paging spaces so that paging activity occurs across several devices.

The **paging type** option is required for all logical volume paging spaces.

The following options are used to maximize paging performance with a logical volume:

- Allocate in the middle of the disk to reduce disk arm travel
- Use multiple paging spaces, each allocated from a separate physical volume.

Chapter 5. File Systems

A *file system* is a hierarchical structure (file tree) of files and directories. This type of structure resembles an inverted tree with the roots at the top and branches at the bottom. This file tree uses directories to organize data and programs into groups, allowing the management of several directories and files at one time. For information on the structure of the file system, see "Organization and Contents of the File Tree."

A file system resides on a single logical volume. Every file and directory belongs to a file system within a logical volume. Because of its structure, some tasks are performed more efficiently on a file system than on each directory within the file system. For example, you can back up, move, or secure an entire file system. You can make an point-in-time image of a JFS file system or a JFS2 file system, called a *snapshot* (AIX 5.2 and later).

Note: The maximum number of logical partitions per logical volume is 32,512. For more information on file system logical volume characteristics, see the **chlv** command.

The **mkfs** (make file system) command or the System Management Interface Tool (**smit** command) creates a file system on a logical volume. For more information on managing file systems, see "File System Management Tasks" on page 41.

To be accessible, a file system must be mounted onto a directory mount point. When multiple file systems are mounted, a directory structure is created that presents the image of a single file system. It is a hierarchical structure with a single root. This structure includes the base file systems and any file systems you create. You can access both local and remote file systems using the **mount** command. This makes the file system available for read and write access from your system. Mounting or unmounting a file system usually requires system group membership. File systems can be mounted automatically, if they are defined in the **/etc/filesystems** file. You can unmount a local or remote file system with the **umount** command, unless a user or process is accessing that file system. For more information on mounting a file system, see "Mounting Overview" on page 42.

Multiple file system types are supported for AIX 5.2, including the journaled file system (JFS) and the enhanced journaled file system (JFS2). For more information on file system types and the characteristics of each type, see "File System Types" on page 46.

Organization and Contents of the File Tree

The file tree organizes files into directories containing similar information. This organization facilitates remote mounting of directories and files. System administrators can use these directories as building blocks to construct a unique file tree for each client mounting individual directories from one or more servers. Mounting files and directories remotely, rather than keeping all information local, has the following advantages:

- Conserves disk space
- Allows easy, centralized system administration
- Provides a more secure environment

The file tree has the following characteristics:

- Files that can be shared by machines of the same hardware architecture are located in the **/usr** file system.
- Variable per-client files, such as spool and mail files, are located in the **/var** file system.
- Architecture-independent, shareable text files, such as manual pages, are located in the **/usr/share** directory.

- The **/** (root) file system contains files and directories critical for system operation. For example, it contains a device directory, programs used for system startup, and mount points where file systems can be mounted onto the root file system.
- The **/home** file system is the mount point for user home directories.
- For servers, the **/export** directory contains paging-space files, per-client (unshared) root file systems, dump, home, and **/usr/share** directories for diskless clients, as well as exported **/usr** directories.

Understanding the Root File System

The root file system is the top of the hierarchical file tree. It contains the files and directories critical for system operation, including the device directory and programs for booting the system. The root file system also contains mount points where file systems can be mounted to connect to the root file system hierarchy.

The following diagram shows many of the subdirectories of the root file system.

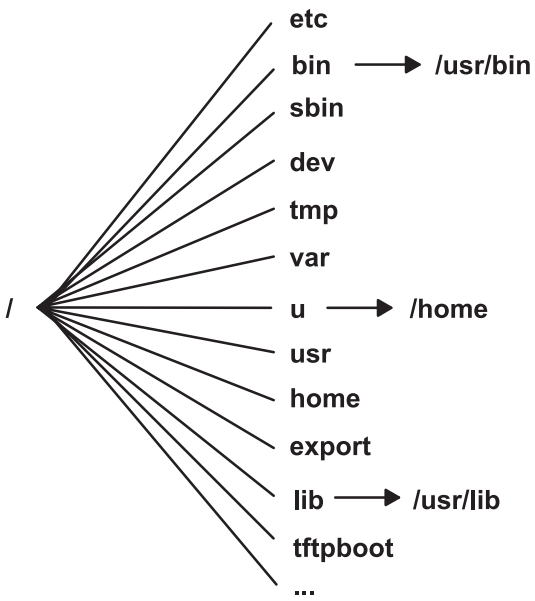

Figure 7. Root File System. This diagram shows the root file system and its subdirectories. The **/bin** *subdirectory points to the* **/usr/bin** *directory. The* **/lib** *subdirectory points to the* **/usr/lib** *directory. The* **/u** *subdirectory points to the* **/home** *directory.*

The following list provides information about the contents of some of the subdirectories of the **/** (root) file system.

/etc Contains configuration files that vary for each machine. Examples include:

- **/etc/hosts**
- **/etc/passwd**

The **/etc** directory contains the files generally used in system administration. Most of the commands that previously resided in the **/etc** directory now reside in the **/usr/sbin** directory. However, for compatibility, the **/usr/sbin** directory contains symbolic links to the locations of some executable files. Examples include:

- **/etc/chown** is a symbolic link to **/usr/bin/chown**.
- **/etc/exportvg** is a symbolic link to **/usr/sbin/exportvg**.

/bin Symbolic link to the **/usr/bin** directory. In prior UNIX file systems, the **/bin** directory contained user commands that now reside in the **/usr/bin** directory.

/sbin	Contains files needed to boot the machine and mount the **/usr** file system. Most of the commands used during booting come from the boot image's RAM disk file system; therefore, very few commands reside in the **/sbin** directory.
/dev	Contains device nodes for special files for local devices. The **/dev** directory contains special files for tape drives, printers, disk partitions, and terminals.
/tmp	Serves as a mount point for a file system that contains system-generated temporary files. The **/tmp** file system is an empty directory.
/var	Serves as a mount point for files that vary on each machine. The **/var** file system is configured as a file system since the files it contains tend to grow. See "Understanding the /var File System" on page 39 for more information.
/u	Symbolic link to the **/home** directory.
/usr	Contains files that do not change and can be shared by machines such as executables and ASCII documentation. Standalone machines mount the root of a separate local file system over the **/usr** directory. Diskless machines and machines with limited disk resources mount a directory from a remote server over the **/usr** file system. See "Understanding the /usr File System" for more information about the file tree mounted over the **/usr** directory.
/home	Serves as a mount point for a file system containing user home directories. The **/home** file system contains per-user files and directories. In a standalone machine, the **/home** directory is contained in a separate file system whose root is mounted over the **/home** directory root file system. In a network, a server might contain user files that are accessible from several machines. In this case, the server copy of the **/home** directory is remotely mounted onto a local **/home** file system.
/export	Contains the directories and files on a server that are for remote clients. See "Understanding the /export Directory" on page 40 for more information about the file tree that resides under the **/export** directory.
/lib	Symbolic link to the **/usr/lib** directory. See "Understanding the /usr File System" for more information.
/tftpboot	Contains boot images and boot information for diskless clients.

Understanding the /usr File System

The **/usr** file system contains executable files that can be shared among machines. The major subdirectories of the **/usr** directory are shown in the following diagram.

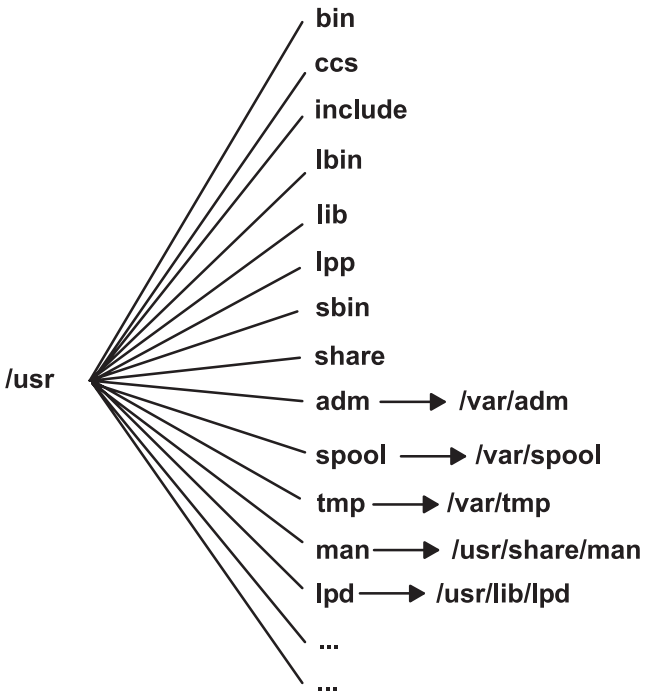

Figure 8. /usr File System. This diagram shows the major subdirectories of the /usr directory, which includes: **/bin**, **/ccs**, **/lib**, **/lpp**, **/adm** *and its* **/var/adm** *subdirectory, and* **/man** *and its* **/usr/share/man** *subdirectory.*

On a standalone machine the **/usr** file system is a separate file system (in the **/dev/hd2** logical volume). On a diskless machine or a machine with limited disk resources, a directory from a remote server is mounted with read-only permissions over the local **/usr** file system. The **/usr** file system contains read-only commands, libraries, and data.

Except for the contents of the **/usr/share** directory, the files and directories in the **/usr** file system can be shared by all machines of the same hardware architecture.

The **/usr** file system includes the following directories:

/usr/bin	Contains ordinary commands and shell scripts. For example, the **/usr/bin** directory contains the **ls**, **cat**, and **mkdir** commands.
/usr/ccs	Contains unbundled development package binaries.
/usr/include	Contains include, or header, files.
/usr/lbin	Contains executable files that are backends to commands.
/usr/lib	Contains architecture-independent libraries with names of the form **lib*.a**. The **/lib** directory in **/** (root) is a symbolic link to the **/usr/lib** directory, so all files that were once in the **/lib** directory are now in the **/usr/lib** directory. This includes a few nonlibrary files for compatibility.
/usr/lpp	Contains optionally installed products.
/usr/sbin	Contains utilities used in system administration, including System Management Interface Tool (SMIT) commands. Most of the commands that once resided in the **/etc** directory now reside in the **/usr/sbin** directory.
/usr/share	Contains files that can be shared among machines with different architectures. See “Understanding the /usr/share Directory” on page 39 for more information.

Symbolic Links to the /var Directory

/usr/adm	Symbolic link to the **/var/adm** directory
/usr/mail	Symbolic link to the **/var/spool/mail** directory
/usr/news	Symbolic link to the **/var/news** directory
/usr/preserve	Symbolic link to the **/var/preserve** directory

/usr/spool	Symbolic link to the **/var/spool** directory
/usr/tmp	Symbolic link to the **/var/tmp** directory, because the **/usr** directory is potentially shared by many nodes and is read-only

Symbolic Links to the /usr/share and /usr/lib Directory

/usr/dict	Symbolic link to the **/usr/share/dict** directory
/usr/man	Symbolic link to the **/usr/share/man** directory
/usr/lpd	Symbolic link to the **/usr/lib/lpd** directory

Understanding the /usr/share Directory

The **/usr/share** directory contains architecture-independent shareable text files. The contents of this directory can be shared by all machines, regardless of hardware architecture.

In a mixed architecture environment, the typical diskless client mounts one server directory over its own **/usr** directory and then mounts a different directory over the **/usr/share** directory. The files below the **/usr/share** directory are contained in one or more separately installable packages. Thus, a node might have the other parts of the **/usr** directory it depends on locally installed while using a server to provide the **/usr/share** directory.

Some of the files in the **/usr/share** directory include the directories and files shown in the following diagram.

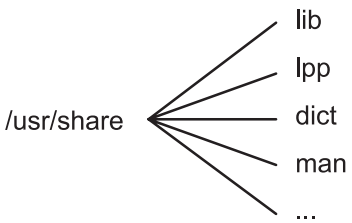

Figure 9. /usr/share Directory. This diagram shows several directories under the **/usr/share** *directory, including* **/lib**, **/lpp**, **/dict**, *and* **/man**.

The **/usr/share** directory includes the following:

/usr/share/man	Contains the manual pages if they have been loaded
/usr/share/dict	Contains the spelling dictionary and its indexes
/usr/share/lib	Contains architecture-independent data files, including **terminfo**, **learn**, **tmac**, **me**, and **macros**
/usr/share/lpp	Contains data and information about optionally installable products on the system

Understanding the /var File System

Attention: The **/var** file system tends to grow because it contains subdirectories and data files that are used by busy applications such as accounting, mail, and the print spooler. If applications on your system use the **/var** file system extensively, routinely run the **skulker** command or increase the file system size beyond the 4MB **/var** default.

Specific **/var** files that warrant periodic monitoring are **/var/adm/wtmp** and **/var/adm/ras/errlog**.

Other **/var** files to monitor are:

/var/adm/ras/trcfile	If the trace facility is turned on
/var/tmp/snmpd.log	If the **snmpd** command is running on your system

The /var directory diagram shows some of the directories in the **/var** file system.

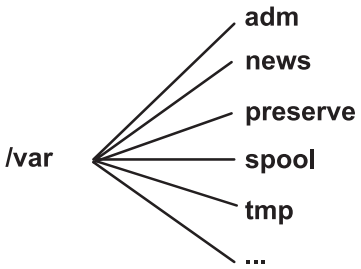

Figure 10. /var Directory. This diagram shows the major subdirectories of the /var directory, including **/adm***,* **/news***,* **/preserve***,* **/spool***, and* **/tmp***.*

/var/adm	Contains system logging and accounting files
/var/news	Contains system news
/var/preserve	Contains preserved data from interrupted edit sessions; similar to the **/usr/preserve** directory in previous releases
/var/spool	Contains files being processed by programs such as electronic mail; similar to the **/usr/spool** directory in previous releases
/var/tmp	Contains temporary files; similar to the **/usr/tmp** directory in previous releases. The **/usr/tmp** directory is now a symbolic link to **/var/tmp**.

Understanding the /export Directory

The **/export** directory contains server files exported to clients, such as diskless, dataless, or disk-poor machines. A server can export several types of disk space, including packages of executable programs, paging space for diskless clients, and root file systems for diskless clients or those with low disk resources. The standard location for such disk space in the file tree is the **/export** directory. Some of the subdirectories of the **/export** directory are shown in the following list:

/exec Contains directories that diskless clients mount over their **/usr** file systems

/swap Contains files for diskless clients' remote paging

/share Contains directories that diskless clients mount over their **/usr/share** directory

/root Contains directories that diskless clients mount over their **/ (root)** file system

/dump Contains directories for diskless clients' remote dump files

/home Contains directories that diskless clients mount over their **/home** file system

The **/export** directory is the default location for client resources for the diskless commands. The **/export** directory is only the location of client resources on the server. Because clients mount these resources onto their own file tree, these resources appear to clients at the normal places in a file tree. The major subdirectories of the **/export** directory, and their corresponding mount points on a client file tree, include:

/export/root

This directory is mounted over the client root (/) file system. Client root directories are located in the **/export/root** directory by default and are named with the client's host name.

/export/exec
Also called the Shared Product Object Tree (SPOT) directory. This directory is mounted over the client **/usr** file system. SPOTs are versions of the **/usr** file system stored in the **/export/exec** directory and have names that reflect their release level. By default, the name is **RISCAIX**.

/export/share
This directory is mounted over the client **/usr/share** directory. This directory contains data that can be shared by many architectures. The default location is **/export/share/AIX/usr/share**.

/export/home
This directory is mounted over the client **/home** file system. It contains user directories grouped by client host names. The default location for client home directories is **/export/home**.

/export/swap
Also called the paging directory. In standalone or dataless systems, paging is provided by a local disk; for diskless clients, this service is provided by a file on a server. This file is named after the client's host name and by default is found in the **/export/swap** directory.

/export/dump
Standalone systems use a local disk as the dump device; diskless clients use a file on a server. The file resides in a directory named after the client host name and by default is found in the **/export/dump** directory.

microcode
This directory contains microcode for physical devices. The default location is **/export/exec/RISCAIX/usr/lib/microcode**.

File System Management Tasks

A file system is a complete directory structure, including a root directory and any subdirectories and files beneath it. File systems are confined to a single logical volume. Some of the most important system management tasks are concerning file systems, specifically:

- Allocating space for file systems on logical volumes
- 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 failures
- Making a snapshot to capture a consistent block-level image of a file system at a given point in time
- Maintaining file systems in a consistent state.

Following is a list of system management commands that help manage file systems:

backup	Performs a full or incremental backup of a file system
chfs -a splitcopy	Creates an online backup of a mounted JFS file system
dd	Copies data directly from one device to another for making file system backups
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 logical volume
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
snapshot	Creates a snapshot of a JFS2 file system
umount	Removes a file system from the system-wide naming structure, making the files and directories in the file system inaccessible.

File System Commands

There are a number of commands designed to operate on file systems, regardless of type. The **/etc/filesystems** file controls the list of file systems that the following commands can manipulate:

chfs	Changes the characteristics of a file system
crfs	Adds a file system
lsfs	Displays the characteristics of a file system
rmfs	Removes a file system
mount	Makes a file system available for use

Four commands operate on virtual file systems types. The **/etc/vfs** file contains the information on the file system types that the following commands manipulate:

chvfs	Changes the characteristics of a file system type
crvfs	Adds a new file system type
lsvfs	Lists the characteristics of a file system type
rmvfs	Removes a file system type

Mounting Overview

Mounting makes file systems, files, directories, devices, and special files available for use at a particular location. It is the only way a file system is made accessible. The **mount** command instructs the operating system to attach a file system at a specified directory.

You can mount a file or directory if you have access to the file or directory being mounted and write permission for the mount point. Members of the system group can also perform device mounts (in which devices or file systems are mounted over directories) and the mounts described in the **/etc/filesystems** file. A user operating with root user authority can mount a file system arbitrarily by naming both the device and the directory on the command line. The **/etc/filesystems** file is used to define mounts to be automatic at system initialization. The **mount** command is used to mount after system startup.

Understanding Mount Points

A *mount point* is a directory or file at which a new file system, directory, or file is made accessible. To mount a file system or a directory, the mount point must be a directory; and to mount a file, the mount point must be a file.

Typically, a file system, directory, or file is mounted over an empty mount point, but that is not required. If the file or directory that serves as the mount point contains any data, that data is not accessible while it is mounted over by another file or directory. In effect, the mounted file or directory covers what was previously in that directory. The original directory or file that has been mounted over is accessible again once the mount over it is undone.

When a file system is mounted over a directory, the permissions of the root directory of the mounted file system take precedence over the permissions of the mount point. The one exception involves the .. (dot dot) parent directory entry in the mounted-over directory. In order for the operating system to access the new file system, the mount point parent directory information must be available.

For example, if the current working directory is **/home/frank**, the command **cd ..** changes the working directory to **/home**. If **/home/frank** directory is the root of a mounted file system, the operating system must find the parent directory information in the **/home/frank** directory in order for the **cd ..** command to succeed.

For any command that requires parent directory information in order to succeed, users must have search permission in the mounted-over directory. Failure of the mounted-over directory to grant search permission can have unpredictable results, especially since the mounted-over directory permissions are not visible. A common problem is failure of the **pwd** command. Without search permission in the mounted-over directory, the **pwd** command returns this message:

```
pwd: Permission denied
```

This problem can be avoided by always setting the permissions of the mounted-over directory to at least 111.

Mounting File Systems, Directories, and Files

There are two types of mounts, a remote mount and a local mount. *Remote mounts* are done on a remote system on which data is transmitted over a telecommunication line. Remote file systems, such as Network File System (NFS), require that the files be exported before they can be mounted. *Local mounts* are mounts done on your local system.

Each file system is associated with a different device (logical volume). Before you can use a file system, it must be connected to the existing directory structure (either the root file system or to another file system that is already connected). The **mount** command makes this connection.

The same file system, directory, or file can be accessed by multiple paths. For example, if you have one database and several users using this database, it can be useful to have several mounts of the same database. Each mount should have its own name and password for tracking and job-separating purposes. This is accomplished by mounting the same file system on different mount points. For example, you can mount from `/home/server/database` to the mount point specified as `/home/user1`, `/home/user2`, and `/home/user3`:

```
/home/server/database        /home/user1
/home/server/database        /home/user2
/home/server/database        /home/user3
```

A file system, directory, or file can be made available to various users through the use of symbolic links. Symbolic links are created with the **ln -s** command. Linking multiple users to a central file ensures that all changes to the file are reflected each time a user accesses the file.

Controlling Automatic Mounts

Mounts can be set to occur automatically during system initialization. There are two types of automatic mounts. The first type consists of those mounts that are required to boot and run the system. These file systems are explicitly mounted by the boot process. The stanzas of such file systems in the **/etc/filesystems** file have `mount = automatic`. The second type of automatic mount is user-controlled. These file systems are mounted by the **/etc/rc** script when it issues the **mount all** command. The stanzas of user-controlled automatic mounts have `mount = true` in **/etc/filesystems**.

The **/etc/filesystems** file controls automatic mounts; they are done hierarchically, one mount point at a time. They can also be placed in a specific order that can be changed and rearranged.

The **/etc/filesystems** file is organized into stanzas, one for each mount. A stanza describes the attributes of the corresponding file system and how it is mounted. The system mounts file systems in the order they appear in the **/etc/filesystems** file. The following is an example of stanzas within the **/etc/filesystems** file:

```
/:
 dev=/dev/hd4
 vol="root"
 mount=automatic
 check=false
 free=true
```

```
 vfs=jfs
 log=/dev/hd8
 type-bootfs

/home:
 dev=/dev/hd1
 vfs=jfs
 log=/dev/hd8
 mount=true
 check=true
 vol="/home"
 free=false

/usr:
 /dev=/dev/hd2
 vfs=jfs
 log=/dev/hd8
 mount=automatic
 check=false
 type=bootfs
 vol="/usr"
 free=false
```

You can edit the **/etc/filesystems** file to control the order in which mounts occur. If a mount is unsuccessful, any of the following mounts defined in the **/etc/filesystems** file continue to mount. For example, if the mount of the **/home** file system is unsuccessful, the mount for the **/usr** file system continues and be mounted. Mounts can be unsuccessful for reasons such as typographical errors, dependency, or a system problem.

Understanding Mount Security for Diskless Workstations

Diskless workstations must have the ability to create and access device-special files on remote machines to have their **/dev** directories mounted from a server. Because servers cannot distinguish device-special files intended for a client from those intended for the server, a user on the server might be able to access the physical devices of the server using the special files of the client device.

For example, the ownership for a **tty** is automatically set to the user using the **tty**. If the user IDs are not the same on both the client and server, a nonprivileged user on the server can access a **tty** that is being used by a different user on the server.

A user who is privileged on a client can create device-special files to match physical devices on the server and have them not require privilege for access. The user can then use an unprivileged account on the server to access the normally protected devices using the new device-special files.

A similar security problem involves the use of **setuid** and **setgid** programs on the client and server. Diskless clients must be able to create and run**setuid** and **setgid** programs on the server for normal operation. Again, the server cannot distinguish between those programs intended for the server and those intended for the client.

In addition, the user IDs and group IDs might not match between the server and client, so users on the server might be able to run programs with capabilities that were not intended for them.

The problem exists because the **setuid** and **setgid** programs and device-special files should only be usable on the machine that created them.

The solution is to use security options to the **mount** command that restrict the ability to use these objects. These options can also be used in stanzas in the **/etc/filesystems** file.

The `nosuid` option in the mount command prevents the execution of **setuid** and **setgid** programs that are accessed via the resulting mounted file system. This option is used for any file system that is being mounted on a particular host for use only by a different host (for example, exported for diskless clients).

The `nodev` option in the **mount** command prevents the opening of devices using device special files that are accessed via the resulting mounted file system. This option is also used for any file system that is being mounted for use only by a different host (for example, exported for diskless clients).

Diskless Mounts

Although the file system of a diskless workstation is mounted from a server **/exports** directory, to the diskless machine, the file system looks just like the file system on a standalone machine.

The following shows the relationship between server exports, and the diskless workstation mount points:

Server Exports	Diskless Imports
/export/root/*HostName*	/ (root)
/export/exec/*SPOTName*	**/usr**
/export/home/*HostName*	**/home**
/export/share	/usr/share
/export/dump	Used by diskless client as dump space
/export/swap	Used by diskless clients as remote paging space.

For more information about the **/export** directory, see “Understanding the /export Directory” on page 40.

Securing Diskless Mounts

In general, users on a server do not have any access to the **/export** directory.

Exporting the /export/root Directory: The **/export/root** directory must be exported with read/write permissions, and the root user on the server must have access. However, you might want to mount this directory with the following options of the **mount** command:

nosuid Prevents a user on the server from running the **setuid** programs of the client
nodev Prevents a user from accessing the server devices using a device-special file of the client.

An alternative to mounting the **/export/root** directory with these options is to avoid giving users running on the server any access to the **/export/root** directory.

Exporting the /export/exec Directory: The **/export/exec** directory is exported with read-only permissions and must provide root access. However, you might want to mount this directory with the following options of the **mount** command:

nosuid Prevents a user on the server from running the **setuid** programs of the client. If you are exporting the server **/usr** directory, you cannot use the **nousid** option.
nodev Prevents a user from accessing the server devices using a device-special file of the client.

Exporting the /export/share Directory: The **/export/share** directory is exported with read-only permissions and must provide root access. Because this directory generally contains only data (no executables or devices), you do not need to use the mount security options.

Exporting the /export/home Directory: There are several ways to mount a user **/home** directory:

- You can mount the **/export/home/***Clienthostname* directory over the client **/home** directory. In this case, the client has read/write permissions and the root user has access. To ensure system security, mount the **/export/home** directory with the following options to the **mount** command:

nosuid Prevents a user on the server from running the **setuid** programs of the client.
nodev Prevents a user from accessing the server devices using a device-special file of the client.

- You can mount the **/home** directory on the server over the **/home** directory of the client. In this case, the **/home** directory is exported with read/write permissions and without root access. To ensure system security, mount the **/home** directory on both the server and client with the `nosuid` and `nodev` options of the **mount** command.
- Alternatively, you can mount on the client each **/home/***UserName* directory on the server over the **/home/***Username* directory on the client so users can log in to different machines and still have access to their home directories. In this case, the **/home/***Username* directories on the server and clients are both mounted with the `nousid` and `nodev` options of the **mount** command.

Exporting the /export/dump Directory: Export the **/export/dump/***Clienthostname* directory with read/write permissions and root access. Users on the server do not have any access to the **/export/dump/***Clienthostname* files.

Exporting the /export/swap Directory: Export the **/export/swap/***Clienthostname* file with read/write permissions and root access. No security measures are necessary. Users on the server do not have any access to the **/export/swap/***Clienthostname* files.

t

File System Types

AIX supports multiple file system types. These include the following:

- "Journaled File System (JFS) or Enhanced Journaled File System (JFS2)"
- "Network File System (NFS)"
- "CD-ROM File System (CDRFS)"
- "DVD-ROM File System (UDFS)" on page 47

Journaled File System (JFS) or Enhanced Journaled File System (JFS2)

Supports the entire set of file system semantics. These file systems use database journaling techniques to maintain structural consistency. This prevents damage to the file system when the system is halted abnormally.

Each JFS or JFS2 resides on a separate logical volume. The operating system mounts the file system during initialization. This multiple file system configuration is useful for system management functions such as backup, restore, and repair, because it isolates a part of the file tree so that you can work on it.

A difference between JFS and JFS2 is that JFS2 is designed to support large files and large file systems. These file system types are described more thoroughly in "Understanding JFS and JFS2" on page 47.

Network File System (NFS)

Is a distributed file system that allows users to access files and directories located on remote computers and use those files and directories as if they were local. For example, users can use operating system commands to create, remove, read, write, and set file attributes for remote files and directories. NFS is described more thoroughly in Chapter 6, Network File System in *AIX 5L Version 5.3 System Management Guide: Communications and Networks*.

CD-ROM File System (CDRFS)

Allows access to the contents of a CD-ROM through the normal file system interfaces. CDRFS is described more thoroughly in "Understanding CDRFS" on page 57.

DVD-ROM File System (UDFS)

Allows access to the contents of a DVD through the normal file system interfaces. UDFS is described more thoroughly in “Understanding UDFS” on page 58.

Understanding JFS and JFS2

The journaled file system (JFS) and the enhanced journaled file system (JFS2) are built into the base operating system. Both file system types link their file and directory data to the structure used by the AIX Logical Volume Manager for storage and retrieval. A difference is that JFS2 is designed to accommodate a 64-bit kernel and larger files.

The following sections describe these file systems. Unless otherwise noted, the following sections apply equally to JFS and JFS2.

JFS2

JFS2 (enhanced journaled file system) is a file system, introduced in AIX 5L™ for POWER™ Version 5.1, that provides the capability to store much larger files than the existing Journaled File System (JFS). Customers can choose to implement either JFS, which is the recommended file system for 32-bit environments, or JFS2, which offers 64-bit functionality.

Note: Unlike the JFS file system, the JFS2 file system will not allow the **link()** API to be used on files of type directory. This limitation may cause some applications that operate correctly on a JFS file system to fail on a JFS2 file system.

The following table provides a summary of JFS and JFS2 functions:

Functions	JFS2	JFS
Fragments and block size	Block sizes (bytes): 512, 1024, 2048, 4096 Maximum file system size in terabytes (TBs): 4, 8, 16, 32	Fragment sizes (bytes): 512, 1024, 2048, 4096 Maximum file system size in gigabytes (GBs): 128, 256, 512, 1024
Maximum file system size	32 TBs	1 TB
Maximum file size	16 TBs	Approximately 63.876 GBs
Number of i-nodes	Dynamic, limited by disk space	Fixed, set at file system creation
Directory organization	B-tree	Linear
Compression	No	Yes
Quotas	Yes	Yes
Error logging	Yes	Yes

Notes:

1. The maximum file size and maximum file system size is limited to 1 TB when used with the 32-bit kernel.
2. JFS2 supports the standard AIX error logging scheme beginning in AIX 5.2. For more information on AIX error logging, please see Error-Logging Overview in *AIX 5L Version 5.3 General Programming Concepts: Writing and Debugging Programs*.

Understanding Disk Space Segmentation

Many UNIX file systems only allocate contiguous disk space in units equal in size to the logical blocks used for the logical division of files and directories. These allocation units are typically referred to as *disk blocks* and a single disk block is used exclusively to store the data contained within a single logical block of a file or directory.

Using a relatively large logical block size (4096 bytes for example) and maintaining disk block allocations that are equal in size to the logical block are advantageous for reducing the number of disk I/O operations that must be performed by a single file system operation. A file or directory data is stored on disk in a small number of large disk blocks rather than in a large number of small disk blocks. For example, a file with a size of 4096 bytes or less is allocated a single 4096-byte disk block if the logical block size is 4096 bytes. A read or write operation therefore only has to perform a single disk I/O operation to access the data on the disk. If the logical block size is smaller requiring more than one allocation for the same amount of data, then more than one disk I/O operation is be required to access the data. A large logical block and equal disk block size are also advantageous for reducing the amount of disk space allocation activity that must be performed as new data is added to files and directories, because large disk blocks hold more data.

Restricting the disk space allocation unit to the logical block size can, however, lead to wasted disk space in a file system containing numerous files and directories of a small size. Wasted disk space occurs when a logical block worth of disk space is allocated to a partial logical block of a file or directory. Because partial logical blocks always contain less than a logical block worth of data, a partial logical block only consumes a portion of the disk space allocated to it. The remaining portion remains unused because no other file or directory can write its contents to disk space that has already been allocated. The total amount of wasted disk space can be large for file systems containing a large number of small files and directories.

The journaled file system (JFS) divides disk space into allocation units called *fragments*. The enhanced journaled file system (JFS2) segments disk space into *blocks*. The objective is the same: to efficiently store data.

JFS fragments are smaller than the default disk allocation size of 4096 bytes. Fragments minimize wasted disk space by more efficiently storing the data in a file or directory partial logical blocks. The functional behavior of JFS fragment support is based on that provided by Berkeley Software Distribution (BSD) fragment support.

JFS2 supports multiple file system block sizes of 512, 1024, 2048, and 4096. Smaller block sizes minimize wasted disk space by more efficiently storing the data in a file or directory's partial logical blocks. Smaller block sizes also result in additional operational overhead. The block size for a JFS2 is specified during its creation. Different file systems can have different block sizes, but only one block size can be used within a single file system.

JFS Fragments: In JFS, the disk space allocation unit is called a *fragment*, and it can be smaller than the logical block size of 4096 bytes. With the use of fragments smaller than 4096 bytes, the data contained within a partial logical block can be stored more efficiently by using only as many fragments as are required to hold the data. For example, a partial logical block that only has 500 bytes could be allocated a fragment of 512 bytes (assuming a fragment size of 512 bytes), thus greatly reducing the amount of wasted disk space. If the storage requirements of a partial logical block increase, one or more additional fragments are allocated.

The fragment size for a file system is specified during its creation. The allowable fragment sizes for journaled file systems (JFS) are 512, 1024, 2048, and 4096 bytes. Different file systems can have different fragment sizes, but only one fragment size can be used within a single file system. Different fragment sizes can also coexist on a single system (machine) so that users can select a fragment size most appropriate for each file system.

JFS fragment support provides a view of the file system as a contiguous series of fragments rather than as a contiguous series of disk blocks. To maintain the efficiency of disk operations, however, disk space is often allocated in units of 4096 bytes so that the disk blocks or allocation units remain equal in size to the logical blocks. A disk-block allocation in this case can be viewed as an allocation of 4096 bytes of contiguous fragments.

Both operational overhead (additional disk seeks, data transfers, and allocation activity) and better utilization of disk space increase as the fragment size for a file system decreases. To maintain the optimum balance between increased overhead and increased usable disk space, the following factors apply to JFS fragment support:

- Where possible, disk space allocations of 4096 bytes of fragments are maintained for a file or the logical blocks of a directory.
- Only partial logical blocks for files or directories less than 32KB in size can be allocated less than 4096 bytes of fragments.

As the files and directories within a file system grow beyond 32 KB in size, the benefit of maintaining disk space allocations of less than 4096 bytes for partial logical blocks diminishes. The disk space savings as a percentage of total file system space grows small while the extra performance cost of maintaining small disk space allocations remains constant. Since disk space allocations of less than 4096 bytes provide the most effective disk space utilization when used with small files and directories, the logical blocks of files and directories equal to or greater than 32 KB are always allocated 4096 bytes of fragments. Any partial logical block associated with such a large file or directory is also allocated 4096 bytes of fragments.

JFS2 Blocks: The enhanced journaled file system segments disk space into *blocks*. JFS2 supports multiple file system block sizes of 512, 1024, 2048, and 4096. Different file systems can have different block sizes, but only one block size can be used within a single file system.

Smaller block sizes minimize wasted disk space by more efficiently storing the data in a file or directory's partial logical blocks. Smaller block sizes can also result in additional operational overhead. Also, device drivers must provide disk block addressability that is the same or smaller than the file system block size.

Because disk space is allocated in smaller units for a file system with a block size other than 4096 bytes, allocation activity can occur more often when files or directories are repeatedly extended in size. For example, a write operation that extends the size of a zero-length file by 512 bytes results in the allocation of one block to the file, assuming a block size of 512 bytes. If the file size is extended further by another write of 512 bytes, an additional block must be allocated to the file. Applying this example to a file system with 4096-byte blocks, disk space allocation occurs only once, as part of the first write operation. No additional allocation activity is performed as part of the second write operation since the initial 4096-byte block allocation is large enough to hold the data added by the second write operation.

File system block size is specified during the file system's creation with Web-based System Manager, the System Management Interface Tool (SMIT), or the **crfs** and **mkfs** commands. The decision of which file system block size to choose should be based on the projected size of files contained by the file system.

The file system block size value can be identified through Web-based System Manager, the System Management Interface Tool (SMIT), or the **lsfs** command. For application programs, the **statfs** subroutine can be used to identify the file system block size.

Blocks serve as the basic unit of disk space allocation, and the allocation state of each block within a file system is recorded in the file system block allocation map. More virtual memory and file system disk space might be required to hold block allocation maps for file systems with a block size smaller than 4096 bytes.

Understanding the Variable Number of I-Nodes

Segmenting disk space at sizes smaller than 4096 bytes optimizes disk space utilization, but it increases the number of small files and directories that can be stored within a file system. However, disk space is only one of the file system resources required by files and directories: each file or directory also requires a disk i-node.

JFS and i-nodes: JFS allows you to specify the number of disk i-nodes created within a file system in case more or fewer than the default number of disk i-nodes is desired. The number of disk i-nodes at file system creation is specified in a value called as the *number of bytes per i-node* or *NBPI*. For example, an

NBPI value of 1024 causes a disk i-node to be created for every 1024 bytes of file system disk space. Another way to look at this is that a small NBPI value (512 for instance) results in a large number of i-nodes, while a large NBPI value (such as 16,384) results in a small number of i-nodes.

For JFS file systems, one i-node is created for every NBPI bytes of allocation group space allocated to the file system. The total number of i-nodes in a file system limits the total number of files and the total size of the file system. An allocation group can be partially allocated, though the full number of i-nodes per allocation group is still allocated. NBPI is inversely proportional to the total number of i-nodes in a file system.

The JFS restricts all file systems to 16M (2^{24}) i-nodes.

The set of allowable NBPI values vary according to the allocation group size (*agsize*). The default is 8 MB. The allowable NBPI values are 512, 1024, 2048, 4096, 8192, and 16,384 with an agsize of 8 MB. A larger agsize can be used. The allowable values for agsize are 8, 16, 32, and 64. The range of allowable NBPI values scales up as agsize increases. If the agsize is doubled to 16 MB, the range of NBPI values also double: 1024, 2048, 4096, 8193, 16384, and 32768.

Fragment size and the NBPI value are specified during the file system's creation with Web-based System Manager, the System Management Interface Tool (SMIT), or the **crfs** and **mkfs** commands. The decision of fragment size and how many i-nodes to create for the file system is based on the projected number and size of files contained by the file system.

You can identify fragment size and NBPI value using the Web-based System Manager, the System Management Interface Tool (SMIT), or the **lsfs** command. For application programs, use the **statfs** subroutine to identify the file system fragment size.

JFS2 and i-nodes: JFS2 allocates i-nodes as needed. If there is room in the file system for additional i-nodes, they are automatically allocated. Therefore, the number of i-nodes available is limited by the size of the file system itself.

Understanding Size Limitations

You define the maximum size for a JFS when you create the file system. The decision of what size to define for a JFS is based on several significant issues. The following section describes the key considerations.

The recommended maximum size for a JFS2 is 16 TBs. Key considerations for very large JFS2 file systems are described in "Log Size Issues" on page 51 and "JFS2 Size Limitations" on page 52.

Although file systems that use allocation units smaller than 4096 bytes require substantially less disk space than those using the default allocation unit of 4096 bytes, the use of smaller fragments might incur performance costs.

The allocation state of each fragment (JFS) or block (JFS2) within a file system is recorded in the file system allocation map. More virtual memory and file system disk space might be required to hold allocation maps for file systems with a fragment or block size smaller than 4096 bytes.

Because disk space is allocated in smaller units for a file system with a fragment (JFS) or block (JFS2) size other than 4096 bytes, allocation activity can occur more often when files or directories are repeatedly extended in size. For example, a write operation that extends the size of a zero-length file by 512 bytes results in the allocation of one 512-byte fragment or block to the file, depending on the file system type. If the file size is extended further by another write of 512 bytes, an additional fragment or block must be allocated to the file. Applying this example to a file system with 4096-byte fragments or blocks, disk space allocation occurs only once, as part of the first write operation. No additional allocation activity must be

performed as part of the second write operation since the initial 4096-byte allocation is large enough to hold the data added by the second write operation. Allocation activity can be minimized if the files are extended by 4096 bytes at a time.

Log Size Issues: One size-related issue is the size of the file system log.

For JFS, in most instances, multiple file systems use a common log configured to be 4 MB in size. For example, after initial installation, all file systems within the root volume group use logical volume hd8 as a common JFS log. The default logical volume partition size is 4 MB, and the default log size is one partition, therefore, the root volume group normally contains a 4 MB JFS log. When file systems exceed 2 GB or when the total amount of file system space using a single log exceeds 2 GB, the default log size might not be sufficient. In either case, the log sizes are scaled upward as the file system size increases. When the size of the log logical volume is changed, the **logform** command must be run to reinitialize the log before the new space can be used. The JFS log is limited to a maximum size of 256 MB.

There is a practical limit to the size of the combined file systems that a single JFS log can support. As a guideline, one trillion bytes of total file system capacity is the recommended limitation for a single JFS log. When this guideline is exceeded or is close to being exceeded, or when out-of-memory errors occur from the **logredo** command (which is called by the **fsck** command), add an additional JFS log and then share the load between the two JFS log files.

For JFS2, in most instances, multiple file systems also use a common log. When file systems exceed 2 GB or when the total amount of file system space using a single log exceeds 2 GB, the default log size might not be sufficient. In either case, you can scale log sizes upward as the file system size increases or you can add an additional JFS2 log and then share the load between the two JFS2 log files.

JFS Size Limits: The maximum JFS size is defined when the file system is created. The NBPI, fragment size, and allocation group size are contributing factors to the decision. The file system size limitation is the minimum of the following:

$NBPI * 2^{24}$

or

$FragmentSize * 2^{28}$

For example, if you select an NBPI ratio of 512, the file system size is limit to 8 GB ($512 * 2^{24} = 8$ GB). JFS supports NBPI values of 512, 1024, 2048, 4096, 8192, 16384, 32768, 65536, and 131072.

The JFS restricts all file systems to 16M (2^{24}) i-nodes.

One i-node is created for every *NBPI* bytes of allocation group space allocated to the file system. An allocation group can be partially allocated, though the full number of i-nodes per allocation group is still allocated. NBPI is inversely proportional to the total number of i-nodes in a file system.

The JFS segregates file system space into groupings of i-nodes and disk blocks for user data. These groupings are called allocation groups. The allocation group size can be specified when the file system is created. The allocation group sizes are 8M, 16M, 32M, and 64M. Each allocation group size has an associated NBPI range. The ranges are defined by the following table:

Allocation Group Size in Megabytes	Allowable NBPI Values
8	512, 1024, 2048, 4096, 8192, 16384
16	1024, 2048, 4096, 8192, 16384, 32768
32	2048, 4096, 8192, 16384, 32768, 65536
64	4096, 8192, 16384, 32768, 65536, 131072

The JFS supports four fragment sizes of 512, 1024, 2048, and 4096 byte units of contiguous disk space. The JFS maintains fragment addresses in i-nodes and indirect blocks as 28-bit numbers. Each fragment must be addressable by a number from 0 to (2^{28}).

JFS2 Size Limitations: Testing has shown that extremely large JFS2 file systems that contain very large files are more practical to maintain than those that contain a large number of small files. When a large file system contains many small files, the **fsck** command and other file system maintenance tasks take a long time to run. The following size limitations are recommended:

Maximum JFS2 file system size:	16TB
Maximum JFS2 file size:	16TB

JFS Free Space Fragmentation: For JFS file systems, using fragments smaller than 4096 bytes can cause greater fragmentation of the free space on the disk. For example, consider an area of the disk that is divided into eight fragments of 512 bytes each. Suppose that different files, requiring 512 bytes each, have written to the first, fourth, fifth, and seventh fragments in this area of the disk, leaving the second, third, sixth, and eighth fragments free. Although four fragments representing 2048 bytes of disk space are free, no partial logical block requiring four fragments (or 2048 bytes) is allocated for these free fragments, since the fragments in a single allocation must be contiguous.

Because the fragments allocated for a file or directory logical blocks must be contiguous, free space fragmentation can cause a file system operation that requests new disk space to fail even though the total amount of available free space is large enough to satisfy the operation. For example, a write operation that extends a zero-length file by one logical block requires 4096 bytes of contiguous disk space to be allocated. If the file system free space is fragmented and consists of 32 noncontiguous 512-byte fragments or a total of 16 KB of free disk space, the write operation will fail because eight contiguous fragments (or 4096 bytes of contiguous disk space) are not available to satisfy the write operation.

A JFS file system with an unmanageable amount of fragmented free space can be defragmented with the **defragfs** command. Running the **defrags** command has a positive impact on performance.

Understanding Sparse Files

A file is a sequence of indexed blocks. Blocks are mapped from the i-node to the logical offset of the file they represent.

A file that has one or more indexes that are not mapped to a data block is referred to as being *sparsely-allocated* or a *sparse file*. A sparse file will have a size associated with it, but it will not have all of the data blocks allocated to fulfill the size requirements. To identify if a file is sparsely-allocated, use the **fileplace** command. It will indicate all blocks in the file that are not currently allocated.

Note: In most circumstances, **du** can also be used to determine if the number of data blocks allocated to a file do not match those required to hold a file of its size. A compressed file system might show the same behavior for files that are not sparsely-allocated.

A sparse file is created when an application extends a file by seeking to a location outside the currently allocated indexes, but the data that is written does not occupy all of the newly assigned indexes. The new file size reflects the farthest write into the file.

A read to a section of a file that has unallocated data blocks results in a buffer of zeroes being returned. A write to a section of a file that has unallocated data blocks causes the necessary data blocks to be allocated and the data written.

This behavior can affect file manipulation or archival commands. For example, the following commands do not preserve the sparse allocation of a file:

cp

mv

tar

cpio

Note: In the case of **mv**, this only applies to moving a file to another file system. If the file is moved within the same file system, it will remain sparse.

The result of a file being copied or restored from the preceding commands has each data block allocated, and thus have no sparse characteristics. However, the following archival commands either preserve sparse characteristics or actively sparse a file:

backup

restore

pax

Because it is possible to overcommit the resources of a file system with sparse files, care should be taken in the use and maintenance of files of this type.

Understanding Large Files and JFS

This section contains information about creating large files and file allocation with the JFS file system type. All JFS2 file systems support large files.

File systems enabled for large files can be created with the **crfs** and **mkfs** commands. Both commands have an option (`bf=true`) to specify file systems enabled for large files. You can also use SMIT or Web-based System Manager to create these file systems also.

In file systems enabled for large files, file data stored before the 4 MB file offset is allocated in 4096-byte blocks. File data stored beyond the 4 MB file offset is allocated with large disk blocks of 128 KB in size. The large disk blocks are actually 32 contiguous 4096-byte blocks.

For example, in a regular file system, the 132-MB file requires 33K 4-KB disk blocks (33 single indirect blocks each filled with 1024 4 KB disk addresses). A 132-MB file in a file system enabled for large files has 1024 4-KB disk blocks and 1024 128-KB disk blocks. The large file geometry requires only two single indirect blocks for the 132 MB file. Both large and regular file types require one double indirect block.

Large disk blocks require 32 contiguous 4 KB blocks. If you write to large files beyond the 4 MB, file offset will fail with ENOSPC if the file system does not contain 32 unused contiguous 4 KB blocks.

Note: The file system may have thousands of free blocks, but if 32 of them are not contiguous, the allocation will fail.

The **defragfs** command reorganizes disk blocks to provide larger contiguous free block areas.

The JFS is required to initialize all new disk allocations. The JFS starts the kernel kproc procedure used to zero initial file allocations when mounting the first large file enabled file system on your system. The kproc procedure remains once the file system enabled for large files has successfully unmounted.

Understanding JFS Data Compression

JFS supports fragmented and compressed file systems, which save disk space by allowing a logical block to be stored on the disk in units or ″fragments″ smaller than the full block size of 4096 bytes. Data compression is not supported for JFS2.

In a fragmented file system, only the last logical block of files no larger than 32KB are stored in this manner, so that fragment support is only beneficial for file systems containing numerous small files. Data compression, however, allows all logical blocks of any-sized file to be stored as one or more contiguous fragments. On average, data compression saves disk space by about a factor of two.

The use of fragments and data compression does, however, increase the potential for fragmentation of the disk free space. Fragments allocated to a logical block must be contiguous on the disk. A file system experiencing free space fragmentation might have difficulty locating enough contiguous fragments for a logical block allocation, even though the total number of free fragments may exceed the logical block requirements. The JFS alleviates free space fragmentation by providing the **defragfs** program which ″defragments″ a file system by increasing the amount of contiguous free space. This utility can be used for fragmented and compressed file systems. The disk space savings gained from fragments and data compression can be substantial, while the problem of free space fragmentation remains manageable.

Data compression in the current JFS is compatible with previous versions of this operating system. The API comprised of all the system calls remains the same in both versions of the JFS.

For more information on fragment support, disk utilization, free space fragmentation, and the performance costs associated with fragments, refer to “Understanding Disk Space Segmentation” on page 47.

Data Compression Implementation:

Attention: The root file system (**/**) must not be compressed. Compressing the **/usr** file system is not recommended because **installp** must be able to accurately calculate its size for updates and new installs. See the “Implicit Behavior” section for more information on size and calculations.

Data compression is an attribute of a file system which is specified when the file system is created with the **crfs** or **mkfs** command. You can also use Web-based System Manager or SMIT to specify data compression. Compression only applies to regular files and long symbolic links in such file systems. Fragment support continues to apply to directories and metadata that are not compressed. Each logical block of a file is compressed by itself before being written to the disk. Compression in this manner facilitates random seeks and updates, while losing only a small amount of freed disk space in comparison to compressing data in larger units.

After compression, a logical block usually requires less than 4096 bytes of disk space. The compressed logical block is written to the disk and allocated only the number of contiguous fragments required for its storage. If a logical block does not compress, then it is written to disk in its uncompressed form and allocated 4096 bytes of contiguous fragments.

The **lsfs -q** command displays the current value for compression. You can also use Web-based System Manager or SMIT to identify data compression.

Implicit Behavior: Because a program that writes a file does not expect an out-of-space (ENOSPC) condition to occur after a successful write (or successful store for mapped files), it is necessary to guarantee that space be available when logical blocks are written to the disk. This is accomplished by allocating 4096 bytes to a logical block when it is first modified so that there is disk space available even if the block does not compress. If a 4096-byte allocation is not available, the system returns an ENOSPC or EDQUOT error condition even though there might be enough disk space to accommodate the compressed logical block. Premature reporting of an out-of-space condition is most likely when operating near disk quota limits or with a nearly full file system.

Compressed file systems might also exhibit the following behavior:

- Because 4096 bytes are initially allocated to a logical block, certain system calls might receive an ENOSPC or EDQUOT error. For example, an old file might be mapped using the **mmap** system call, and a store operation into a previously written location can result in an ENOSPC error.
- With data compression, a full disk block remains allocated to a modified block until it is written to disk. If the block had a previously committed allocation of less than a full block, the amount of disk space tied up by the block is the sum of the two, the previous allocation not being freed until the file (i-node) is committed. This is the case for normal fragments. The number of logical blocks in a file that can have previously committed allocations is, at most, one for normal fragments but can be as many as the number of blocks in a file with compression.

- None of the previously committed resources for a logical block are freed until the **fsync** or **sync** system call is run by the application program.
- The **stat** system call indicates the number of fragments allocated to a file. The number reported is based on 4096 bytes being allocated to modified but unwritten blocks and the compressed size of unmodified blocks. Previously committed resources are not counted by the **stat** system call. The **stat** system call reports the correct number of allocated fragments after an i-node commit operation if none of the modified blocks compressed. Similarly, disk quotas are charged for the current allocation. As the logical blocks of a file are written to the disk, the number of fragments allocated to them decrease if they compress, and thereby change disk quotas and the result from **stat**.

Compression Algorithm: The compression algorithm is an IBM® version of LZ. In general, LZ algorithms compress data by representing the second and later occurrences of a given string with a pointer that identifies the location of the first occurrence of the string and its length. At the beginning of the compression process, no strings have been identified, so at least the first byte of data must be represented as a ″raw″ character requiring 9-bits (0,byte). After a given amount of data is compressed, say *N* bytes, the compressor searches for the longest string in the *N* bytes that matches the string starting at the next unprocessed byte. If the longest match has length 0 or 1, the next byte is encoded as a ″raw″ character. Otherwise, the string is represented as a (pointer,length) pair and the number of bytes processed is incremented by length. Architecturally, IBM LZ supports values of *N* of 512, 1024, or 2048. IBM LZ specifies the encoding of (pointer,length) pairs and of raw characters. The pointer is a fixed-length field of size log2 *N*, while the length is encoded as a variable-length field.

Performance Costs: Because data compression is an extension of fragment support, the performance associated with fragments also applies to data compression. Compressed file systems also affect performance in the following ways:

- It can require a great deal of time to compress and decompress data so that the usability of a compressed file system might be limited for some user environments.
- Most UNIX regular files are written only once, but some are updated in place. For the latter, data compression has the additional performance cost of having to allocate 4096 bytes of disk space when a logical block is first modified, and then reallocate disk space after the logical block is written to the disk. This additional allocation activity is not necessary for regular files in a uncompressed file system.
- Data compression increases the number of processor cycles. For the software compressor, the number of cycles for compression is approximately 50 cycles per byte, and for decompression 10 cycles per byte.

Understanding JFS Online Backups and JFS2 Snapshots

You can make an point-in-time image of a JFS file system or of a JFS2 file system (AIX 5.2 and later), which you can then use for backup purposes. There are differences, however, in the requirements and behavior of this image for each file system type.

For a JFS file system, you can split off a read-only static copy of a mirrored copy of the file system. Typically, a mirrored copy is updated whenever the original file system is updated, but this point-in-time copy does not change. It remains a stable image of the point in time at which the copy was made. When this image is used for backing up, any modifications that begin after you begin the procedure to create the image might not be present in the backup copy. Therefore, it is recommended that file system activity be minimal while the split is taking place. Any changes that occur after the split is made will not be present in the backup copy.

For a JFS2 file system, the point-in-time image is called a *snapshot*. The snapshot remains static and it retains the same security permissions as the original file system (called the *snappedFS*) had when the snapshot was made. Also, you can create a JFS2 snapshot without unmounting or quiescing the file system. You can use a JFS2 snapshot to use as an online backup of the file system, to access the files or directories as they existed when the snapshot was taken, or to back up to removable media. Note the following about JFS2 snapshots:

- A snapshot image of the root (**/**) or **/usr** file system is overwritten when the system is rebooted. Snapshots of other file systems can be preserved by unmounting the file system before rebooting. Snapshots created in AIX 5.2 with 5200-01 are recoverable. When **fsck** or **logredo** runs on a JFS2 filesystem with a snapshot created on AIX 5.2 with 5200-01, the snapshot will be preserved. A cleanly unmounted filesystem with an AIX 5.2 created snapshot will also be recoverable once it is mounted on an AIX 5.2 with 5200-01 system.
- Running the **defragfs** command against a file system with snapshots is not recommended. Every block that is moved during defragmentation must also be copied to the snapshot, which is both time consuming and a waste of space in the snapshot logical volume.
- If a snapshot runs out of space, all snapshots for that snappedFS are deleted. This failure writes an entry to the error log.
- If a write to a snapshot fails, all snapshots for that snappedFS are deleted. This failure writes an entry to the error log.
- A snapshot that is created or accessed on a AIX 5.2 with 5200-01 system cannot be accessed on an AIX 5.2 system. These snapshots must be deleted before the filesystem can be mounted.
- A JFS2 file system that has a snapshot on AIX 5.3 cannot be accessed on any releases prior to AIX 5.2 with 5200-01. If the system is to be moved back, the snapshots must be deleted first to allow file system access.

Understanding Compatibility and Migration

JFS file systems are fully compatible within AIX 5.1 and AIX 5.2. Previous supported versions of the operating system are compatible with the current JFS, although file systems with a nondefault fragment size, NBPI value, or allocation group size might require special attention if migrated to a previous version.

JFS2 file systems, with the exception of snapshots, are compatible within AIX 5.1 and AIX 5.2, but not with earlier versions of the operating system. JFS2 file systems with snapshots are not supported in AIX 5.1. Always ensure a clean unmount of all JFS2 file systems before reverting to an earlier version of AIX because the **logredo** command does not necessarily run against a file system created for a later release.

Note: JFS2 file systems created or converted to v2 format cannot be accessed on prior releases of AIX.

The following sections describe aspects that might cause problems with file systems created under earlier versions of the operating system.

JFS File System Images: Any JFS file system image created with the default fragment size and NBPI value of 4096 bytes, and default allocation group size (agsize) of 8 can be interchanged with JFS file system images created under AIX 4.3 and later versions of this operating system without requiring any special migration activities.

Note: JFS2 Snapshots: JFS2 snapshots created or accessed in AIX 5.2 with 5200-01 are not accessible on earlier releases. These snapshots must be deleted before the filesystem can be mounted.

Backup and Restore between JFS File Systems: Backup and restore sequences can be performed between JFS file systems with different block sizes, however because of increased disk utilization, restore operations can fail due to a lack of free blocks if the block size of the source file system is smaller than the block size of the target file system. This is of particular interest for full file system backup and restore sequences and can even occur when the total file system size of the target file system is larger than that of the source file system.

While backup and restore sequences can be performed from compressed to uncompressed file systems or between compressed file systems with different fragment sizes, because of the enhanced disk utilization of compressed file systems, restore operations might fail due to a shortage of disk space. This is of particular interest for full file system backup and restore sequences and might even occur when the total file system size of the target file system is larger than that of the source file system.

JFS and JFS2 Device Driver Limitations: A device driver must provide disk block addressability that is the same or smaller than the JFS file system fragment size or the JFS2 block size. For example, if a JFS file system was made on a user supplied RAM disk device driver, the driver must allow 512 byte blocks to contain a file system that had 512 byte fragments. If the driver only allowed page level addressability, a JFS with a fragment size of 4096 bytes could only be used.

Understanding CDRFS

A CDRFS file system is a read-only local file system implementation under the logical file system (LFS) layer. It is stored on CD-ROM media. Beginning with AIX 5.2, CDs are automatically mounted by default but this feature can be disabled. If the feature has been disabled, use the **cdmount** command to mount the CDRFS file system.

For AIX 5.1 and earlier, mount the CD-ROM and its file system using the mount command and the CD must be write-protected. For example:

```
mount -r -v cdrfs /dev/cd0 /mnt
```

You must specify the following when mounting a CD-ROM (AIX 5.1 and earlier):

Device name
Defines the name of the device containing the media.

Mount point
Specifies the directory where the file system will be mounted.

Automatic mount
Specifies whether the file system will be mounted automatically at system restart.

AIX supports the following CDRFS volume and file structure formats:

Type	Description
The ISO 9660:1988(E) standard	The CDRFS supports ISO 9660 level 3 of interchange and level 1 of implementation.
The High Sierra Group Specification	Precedes the ISO 9660 and provides backward compatibility with previous CD-ROMs.
The Rock Ridge Group Protocol	Specifies extensions to the ISO 9660 that are fully compliant with the ISO 9660 standard, and that provide full POSIX file system semantics based on the System Use Sharing Protocol (SUSP) and the Rock Ridge Interchange Protocol (RRIP), enabling mount/access CD-ROM as with any other UNIX file system.
The CD-ROM eXtended Architecture File Format (in Mode 2 Form 1 sector format only)	The CD-ROM eXtended Architecture (XA) file format specifies extensions to the ISO 9660 that are used in CD-ROM-based multimedia applications for example, Photo CD.

For all volume and file structure formats, the following restrictions apply:

- Single-volume volume set only
- Non-interleaved files only

The CDRFS is dependent upon the underlying CD-ROM device driver to provide transparency of the physical sector format (CD-ROM Mode 1 and CD-ROM XA Mode 2 Form 1), and the multisession format of the disks (mapping the volume descriptor set from the volume recognition area of the last session).

Note: The CDRFS must be unmounted from the system before you can remove the CD-ROM media.

Understanding UDFS

Beginning with AIX 5.2, another supported file system type is UDFS, which is a read-only file system stored on DVD-ROM media. The UDFS must be unmounted from the system before you can remove the media. The operating system supports UDFS format versions 1.50, 2.00, and 2.01.

To use the cdmount command to automatically mount a read/write UDFS, edit the **cdromd.conf** file. You can also manually mount a read/write UDFS with the **mount** command.

Chapter 6. Backup and Restore

This chapter contains information about methods of backing up and restoring information as well as backup policies for specific needs.

Topics covered are:

Backup Overview

Backing up file systems, directories, and files represents a significant investment of time and effort. At the same time, all computer files are potentially easy to change or erase, either intentionally or by accident. When a hard disk crashes, the information contained on that disk is destroyed. The only way to recover the destroyed data is to retrieve the information from your backup copy. If you carefully and methodically back up your file systems, you can always restore recent versions of files or file systems with little difficulty.

Backup Methods

Several methods exist for backing up information. One of the most frequently used methods is called *backup by name* or *file name archive*. This method of backup is done when the **i** flag is specified and is used to make a backup copy of individual files and directories. It is a method commonly used by individual users to back up their accounts.

Another frequently used method is called *backup by i-node* or *file system archive*. This method of backup is done when the **i** flag is *not* specified. It is used to make a backup copy of an entire file system and is the method commonly used by system administrators to back up large groups of files, such as all of the user accounts in **/home**. A file system backup allows incremental backups to be performed easily. An incremental backup backs up all files that have been modified since a specified previous backup.

The **compress** and **pack** commands enable you to compress files for storage, and the **uncompress** and **unpack** commands unpack the files once they have been restored. The process of packing and unpacking files takes time, but once packed, the data uses less space on the backup medium.

Several commands create backups and archives. Because of this, data that has been backed up needs to be labeled as to which command was used to initiate the backup, and how the backup was made (by name or by file system).

backup	Backs up files by name or by file system.
mksysb	Creates an installable image of the rootvg.
cpio	Copies files into and out of archive storage.
dd	Converts and copies a file. Commonly used to convert and copy data to and from systems running other operating systems, for example, mainframes. **dd** does not group multiple files into one archive; it is used to manipulate and move data.
tar	Creates or manipulates **tar** format archives.
rdump	Backs up files by file system onto a remote machine's device.
pax	(POSIX-conformant archive utility) Reads and writes **tar** and **cpio** archives.

Deciding on a Backup Policy

No single backup policy can meet the needs of all users. A policy that works well for a system with one user, for example, could be inadequate for a system that serves one hundred users. Likewise, a policy developed for a system on which many files are changed daily would be inefficient for a system on which data changes infrequently. Whatever the appropriate backup strategy for your site, it is very important that one exist and that backups be done frequently and regularly. It is difficult to recover from data loss if a good backup strategy has not been implemented.

Only you can determine the best backup policy for your system, but the following general guidelines might be helpful:

- Make sure you can recover from major losses.

  Can your system continue to run after any single fixed disk failure? Can you recover your system if all the fixed disks should fail? Could you recover your system if you lost your backup diskettes or tape to fire or theft? If data were lost, how difficult would it be to re-create it? Think through possible, even unlikely, major losses, and design a backup policy that enables you to recover your system after any of them.
- Check your backups periodically.

  Backup media and their hardware can be unreliable. A large library of backup tapes or diskettes is useless if data cannot be read back onto a fixed disk. To make certain that your backups are usable, display the table of contents from the backup tape periodically (using **restore -T** or **tar -t** for archive tapes). If you use diskettes for your backups and have more than one diskette drive, read diskettes from a drive other than the one on which they were created. You might want the security of repeating each level 0 backup with a second set of media. If you use a streaming tape device for backups, you can use the **tapechk** command to perform rudimentary consistency checks on the tape.
- Keep old backups.

  Develop a regular cycle for reusing your backup media; however, do not reuse all of your backup media. Sometimes it is months before you or some other user of your system notices that an important file is damaged or missing. Save old backups for such possibilities. For example, you could have the following three cycles of backup tapes or diskettes:
  - Once per week, recycle all daily diskettes except the Friday diskette.
  - 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, preferably in a different building.
- Check file systems before backing up.

  A backup made from a damaged file system might be useless. Before making your backups, it is good policy to check the integrity of the file system with the **fsck** command.
- Ensure files are not in use during a backup.

  Do not use the system when you make your backups. If the system is in use, files can change while they are being backed up, and the backup copy will not be accurate.
- Back up your system before major changes are made to the system.

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

Note: For the backup of named pipes (FIFO special files) the pipes can be either closed or open. However, the restoration fails when the backup is done on open named pipes. When restoring a FIFO special file, its i-node is all that is required to recreate it because it contains all its characteristic information. The content of the named pipe is not relevant for restored. Therefore, the file size during backup is zero (all the FIFOs closed) before the backup is made.

Attention: System backup and restoration procedures require that the system be restored on the same type of platform from which the backup was made. In particular, the CPU and I/O planar boards must be of the same type.

Understanding Backup Media

Several types of backup media are available. The types of backup media available to your specific system configuration depend upon your software and hardware. The types most frequently used are tapes, diskettes, remote archives, and alternate local hard disks. Unless you specify a different device using the **backup -f** command, the **backup** command automatically writes its output to **/dev/rfd0**, which is the diskette drive.

Attention: Running the **backup** command results in the loss of all material previously stored on the selected output medium.

Restoring Data

After the data has been correctly backed up, there are several different methods of restoring the data based upon the type of backup command you used.

You need to know how your backup or archive was created to restore it correctly. Each backup procedure gives information about restoring data. For example, if you use the **backup** command, you can specify a backup either by file system or by name. That backup must be restored the way it was done, by file system or by name.

Several commands restore backed up data, such as:

restore	Copies files created by the **backup** command.
rrestore	Copies file systems backed up on a remote machine to the local machine.
cpio	Copies files into and out of archive storage.
tar	Creates or manipulates **tar** archives.
pax	(POSIX-conformant archive utility) Reads and writes **tar** and **cpio** archives.

Developing a Backup Strategy

There are two methods of backing up large amounts of data:

- Complete system backup
- Incremental backup

To understand these two types of backups and which one is right for a site or system, it is important to have an understanding of file system structure and data placement. After you have decided on a strategy for data placement, you can develop a backup strategy for that data. See Implementing Scheduled Backups for an example of a backup strategy that includes weekly complete system backups and daily incremental backups.

File System Structure

It is important to understand the difference between a file system and a directory. A file system is a section of hard disk that has been allocated to contain files. This section of hard disk is accessed by mounting the file system over a directory. After the file system is mounted, it looks just like any other directory to the end user. However, because of the structural differences between the file systems and directories, the data within these entities can be managed separately.

When the operating system is installed for the first time, it is loaded into a directory structure, as shown in the following illustration.

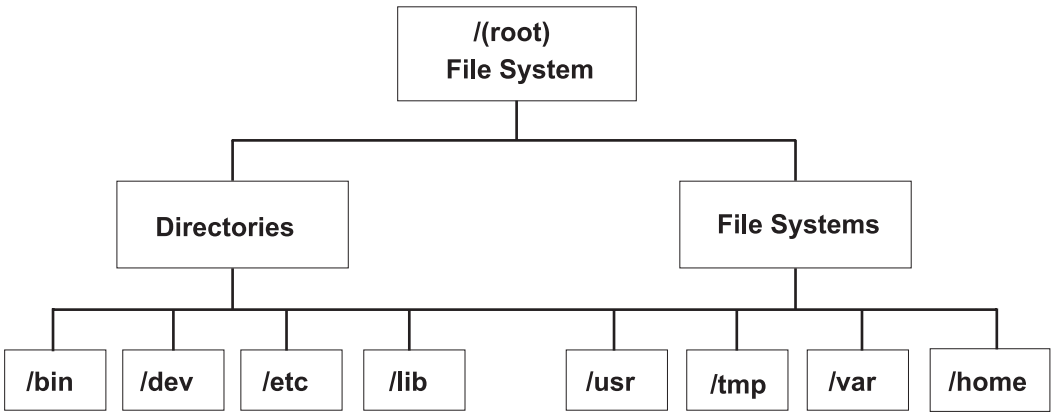

*Figure 11. / (root) File System Tree. This tree chart shows a directory structure with the **/** (root) file system at the top, branching downward to directories and file systems. Directories branch to **/bin**, **/dev**, **/etc**, and **/lib**. File systems branch to **/usr**, **/tmp**, **/var**, and **/home**.*

The directories on the right (**/usr**, **/tmp**, **/var**, and **/home**) are all file systems so they have separate sections of the hard disk allocated for their use. These file systems are mounted automatically when the system is started, so the end user does not see the difference between these file systems and the directories listed on the left (**/bin**, **/dev**, **/etc**, and **/lib**).

System Data Versus User Data

Data is defined as programs or text and for this discussion is broken down into two classes:

- System data, which makes up the operating system and its extensions. This data is always to be kept in the system file systems, namely **/** (root), **/usr**, **/tmp**, **/var**, and so on.
- User data is typically local data that individuals need to complete their specific tasks. This data is to be kept in the **/home** file system or in file systems that are created specifically for user data.

User programs and text are not to be placed in file systems designed to contain system data. For example, a system manager might create a new file system and mount it over**/local**. An exception is **/tmp**, which is used for temporary storage of system and user data.

Backing Up

In general, backups of user and system data are kept in case data is accidentally removed or in case of a disk failure. It is easier to manage backups when user data is kept separate from system data. The following are reasons for keeping system data separate from user data:

- User data tends to change much more often than operating system data. Backup images are much smaller if the system data is not backed up into the same image as the user data. The number of users affects the storage media and frequency required for backup.
- It is quicker and easier to restore user data when it is kept separate. Restoring the operating system along with the user data requires extra time and effort. The reason is that the method used to recover the operating system data involves booting the system from removable media (tape or CD-ROM) and installing the system backup.

To back up the system data, unmount all user file systems, including **/home** with the **umount** command. If these file systems are in use, you cannot unmount them. Schedule the backups at low usage times so they can be unmounted; if the user data file systems remain mounted, they are backed up along with the operating system data. Use the **mount** command to ensure that only the operating system file systems are mounted.

The only mounted file systems are **/**, **/usr**, **/var**, and **/tmp**, and the output of the **mount** command should be similar to the following:

```
node  mounted  mounted over vfs       date         options

      /dev/hd4     /        jfs   Jun 11 10:36  rw,log=/dev/hd8

      /dev/hd2     /usr     jfs   Jun 11 10:36  rw,log=/dev/hd8

      /dev/hd9var  /var     jfs   Jun 11 10:36  rw,log=/dev/hd8

      /dev/hd      /tmp     jfs   Jun 11 10:36  rw,log=/dev/hd8
```

After you are certain that all user file systems are unmounted, See “Backing Up the System Image and User-Defined Volume Groups” on page 64 for information on backing up the operating system data.

When you finish backing up the operating system, mount the user file system using the **smit mount** command. Next, you can back up files, file systems, or other volume groups, depending on your needs. Procedures on these backups are covered later in the chapter.

Replicating a System (Cloning)

Cloning allows you to save configuration data along with user or system data. For example, you might want to replicate a system or volume group; this is sometimes called cloning. You can then install this image onto another system and can use it just like the first system. The **mksysb** command is used to clone the rootvg volume group, which contains the operating system, while the **savevg** command is used to clone a volume group. Procedures for backing up both your system and user volume groups are discussed later in the chapter.

Backing Up User Files or File Systems

To back up user files and file systems, you can use the Web-based System Manager, the SMIT fast paths **smit backfile** or **smit backfilesys**, or the commands listed in “Backup Methods” on page 59.

You can use the SMIT interface for backing up single and small file systems by name, such as **/home** on your local system. Note that SMIT cannot make archives in any other format than that provided by the **backup** command. Also, not every flag of the **backup** command is available through SMIT. SMIT might hang if multiple tapes or disks are needed during the backup. For more information, see the **backup** command description in *AIX 5L Version 5.3 Commands Reference*.

Use the **backup** command when you want to back up large and multiple file systems. You can specify a level number to control how much data is backed up (full, 0; incremental, 1-9). Using the **backup** command is the only way you can specify the level number on backups.

The **backup** command creates copies in one of the two following backup formats:

- Specific files backed up by name using the **-i** flag.
- Entire file systems backed up by i-node using the *-Level* and *FileSystem* parameters. The file system is defragmented when it is restored from backup.

Attention: Backing up by i-node does not work correctly for files that have a user ID (UID) or a group ID (GID) greater than 65535. These files are backed up with UID or GID truncated and will, therefore, have the wrong UID or GID attributes when restored. For these cases, you must back up by name.

Backing Up the System Image and User-Defined Volume Groups

The rootvg is stored on a hard disk, or group of disks, and contains start up files, the BOS, configuration information, and any optional software products. A *user-defined volume group* (also called *nonrootvg volume group*) typically contains data files and application software. You can backup an image of the system and volume groups using Web-based System Manager, SMIT, or command procedures.

A backup image serves two purposes. One is to restore a corrupted system using the system backup image. The other is to transfer installed and configured software from one system to others.

The Web-based System Manager and SMIT procedures use the **mksysb** command to create a backup image that can be stored either on tape or in a file. If you choose tape, the backup program writes a *boot image* to the tape, which makes it suitable for installing.

Notes:

1. Startup tapes cannot be made on or used to start a PowerPC-based personal computer.
2. If you choose the SMIT method for backup, you must first install the **sysbr** fileset in the **bos.sysmgt** software package. See Installing optional software products and service updates in the *AIX 5L Version 5.3 Installation Guide and Reference* for information on how to install software packages and options.

Configuring before the Backup

Configure the source system before creating a backup image of it. If, however, you plan to use a backup image for installing other, differently configured target systems, create the image *before* configuring the source system.

The *source* system is the system from which you created the backup copy. The *target* system is the system on which you are installing the backup copy.

The installation program automatically installs only the device support required for the hardware configuration of the installed machine. Therefore, if you are using a system backup to install other machines, you might need to install additional devices on the source system before making the backup image and using it to install one or more target systems.

Use Web-based System Manager or the SMIT fast path, **smit devinst**, to install additional device support on the source system.

- If there is sufficient disk space on the source and target systems, install all device support.
- If there is limited disk space on the source and target systems, selectively install device support.

For information on installing optional software, see Installing optional software and service updates in the *AIX 5L Version 5.3 Installation Guide and Reference*.

A backup transfers the following configurations from the source system to the target system:

- Paging space information
- Logical volume information
- rootvg information
- Placement of logical partitions (if you have selected the map option).

See Customizing your installation in the *AIX 5L Version 5.3 Installation Guide and Reference* for information about how to set installation parameters to enable you to bypass menu prompts when you install the target machine from a system backup.

Mounting and Unmounting File Systems

The Backing Up Your System procedure backs up only mounted file systems in the rootvg. You must, therefore, mount all file systems you want to back up before starting. Similarly, you must unmount file systems you do *not* want backed up.

This backup procedure backs up files twice if a local directory is mounted over another local directory in the same file system. For example, if you mount **/tmp** over **/usr/tmp**, the files in the **/tmp** directory are backed up twice. This duplication might exceed the number of files a file system can hold, which can cause a future installation of the backup image to fail.

Security Considerations

If you install the backup image on other systems, you might not, for security reasons, want passwords and network addresses copied to the target systems. Also, copying network addresses to a target system creates duplicate addresses that can disrupt network communications.

Restoring a Backup Image

When installing the backup image, the system checks whether the target system has enough disk space to create all the logical volumes stored on the backup. If there is enough space, the entire backup is recovered. Otherwise, the installation halts and the system prompts you to choose more destination hard disks.

File systems created on the target system are the same size as they were on the source system, unless the **SHRINK** variable was set to `yes` in the **image.data** file before the backup image was made. An exception is the **/tmp** directory, which can be increased to allocate enough space for the **bosboot** command. For information about setting variables, see the **image.data** file in *AIX 5L Version 5.3 Files Reference*.

When the system finishes installing the backup image, the installation program reconfigures the ODM on the target system. If the target system does not have exactly the same hardware configuration as the source system, the program might modify device attributes in the following target system files:

- All files in **/etc/objrepos** beginning with ″Cu″
- All files in the **/dev** directory.

For more information about installing (or restoring) a backup image, see Installing system backups in the *AIX 5L Version 5.3 Installation Guide and Reference*.

Backing up Files on a DMAPI-Managed JFS2 File System

Beginning with AIX 5L Version 5.3 with the 5300-03 Recommended Maintenance package, there are options in the **tar** and **backbyinode** commands that allow you to back up the extended attributes (EAs).

With the **backbyinode** command on a DMAPI file system, only the data resident in the file system at the time the command is issued is backed up. The **backbyinode** command examines the current state of metadata to do its work. This can be advantageous with DMAPI, because it backs up the state of the managed file system. However, any offline data will not be backed up.

To back up all of the data in a DMAPI file system, use a command that reads entire files, such as the **tar** command. This can cause a DMAPI-enabled application to restore data for every file accessed by the **tar** command, moving data back and forth between secondary and tertiary storage, so there can be performance implications.

Chapter 7. System Environment

The system environment is primarily the set of variables that define or control certain aspects of process execution. They are set or reset each time a shell is started. From the system-management point of view, it is important to ensure the user is set up with the correct values at log in. Most of these variables are set during system initialization. Their definitions are read from the **/etc/profile** file or set by default.

Topics discussed in this chapter are:

Profiles Overview

The shell uses two types of profile files when you log in to the operating system. It evaluates the commands contained in the files and then runs the commands to set up your system environment. The files have similar functions except that the **/etc/profile** file controls profile variables for all users on a system whereas the **.profile** file allows you to customize your own environment.

The following profile and system environment information is provided:

- **/etc/profile** File
- **.profile** File
- Changing the System Date and Time
- Changing the Message of the Day
- "List of Time Data Manipulation Services" on page 68.

/etc/profile File

The first file that the operating system uses at login time is the **/etc/profile** file. This file controls system-wide default variables such as:

- Export variables
- File creation mask (umask)
- Terminal types
- Mail messages to indicate when new mail has arrived.

The system administrator configures the **profile** file for all users on the system. Only the system administrator can change this file.

.profile File

The second file that the operating system uses at login time is the **.profile** file. The **.profile** file is present in your home (**$HOME**) directory and enables you to customize your individual working environment. The **.profile** file also overrides commands and variables set in the **/etc/profile** file. Since the **.profile** file is hidden, use the **ls -a** command to list it. Use the **.profile** file to control the following defaults:

- Shells to open
- Prompt appearance
- Environment variables (for example, search path variables)
- Keyboard sound

The following example shows a typical **.profile** file:

```
PATH=/usr/bin:/etc:/home/bin1:/usr/lpp/tps4.0/user:/home/gsc/bin::
epath=/home/gsc/e3:
export PATH epath
csh
```

This example has defined two paths (`PATH` and `epath`), exported them, and opened a C shell (`csh`).

You can also use the **.profile** file (or if it is not present, the **profile** file) to determine login shell variables. You can also customize other shell environments. For example, use the **.chsrc** and **.kshrc** files to tailor a C shell and a Korn shell, respectively, when each type shell is started.

List of Time Data Manipulation Services

The time functions access and reformat the current system date and time. You do not need to specify any special flag to the compiler to use the time functions.

Include the header file for these functions in the program. To include a header file, use the following statement:

```
#include <time.h>
```

The time services are the following:

adjtime	Corrects the time to allow synchronization of the system clock.
ctime, **localtime**, **gmtime**, **mktime**, **difftime**, **asctime**, **tzset**	Converts date and time to string representation.
getinterval, **incinterval**, **absinterval**, **resinc**, **resabs**, **alarm**, **ualarm**, **getitimer**, **setitimer**	Manipulates the expiration time of interval timers.
gettimer, **settimer**, **restimer**, **stime**, **time**	Gets or sets the current value for the specified systemwide timer.
gettimerid	Allocates a per-process interval timer.
gettimeofday, **settimeofday**, **ftime**	Gets and sets date and time.
nsleep, **usleep**, **sleep**	Suspends a current process from running.
reltimerid	Releases a previously allocated interval timer.

Dynamic Processor Deallocation

Starting with machine type 7044 model 270, the hardware of all systems with two or more processors is able to detect correctable errors, which are gathered by the firmware. These errors are not fatal and, as long as they remain rare occurrences, can be safely ignored. However, when a pattern of failures seems to be developing on a specific processor, this pattern might indicate that this component is likely to exhibit a fatal failure in the near future. This prediction is made by the firmware based on the failure rates and threshold analysis.

On these systems, AIX implements continuous hardware surveillance and regularly polls the firmware for hardware errors. When the number of processor errors hits a threshold and the firmware recognizes that there is a distinct probability that this system component will fail, the firmware returns an error report. In all cases, the error is logged in the system error log. In addition, on multiprocessor systems, depending on the type of failure, AIX attempts to stop using the untrustworthy processor and deallocate it. This feature is called *Dynamic Processor Deallocation*.

At this point, the processor is also flagged by the firmware for persistent deallocation for subsequent reboots, until maintenance personnel replaces the processor.

Potential Impact to Applications

Processor deallocation is transparent for the vast majority of applications, including drivers and kernel extensions. However, you can use the published interfaces to determine whether an application or kernel extension is running on a multiprocessor machine, find out how many processors there are, and bind threads to specific processors.

The bindprocessor interface for binding processes or threads to processors uses bind CPU numbers. The bind CPU numbers are in the range [0..*N*-1] where *N* is the total number of CPUs. To avoid breaking applications or kernel extensions that assume no ″holes″ in the CPU numbering, AIX always makes it appear for applications as if it is the ″last″ (highest numbered) bind CPU to be deallocated. For instance, on an 8-way SMP, the bind CPU numbers are [0..7]. If one processor is deallocated, the total number of available CPUs becomes 7, and they are numbered [0..6]. Externally, it looks like CPU 7 has disappeared, regardless of which physical processor failed.

Note: In the rest of this description, the term *CPU* is used for the logical entity and the term *processor* for the physical entity.

Potentially, applications or kernel extensions that are binding processes or threads could be broken if AIX silently terminated their bound threads or forcefully moved them to another CPU when one of the processors needs to be deallocated. Dynamic Processor Deallocation provides programming interfaces so that such applications and kernel extensions can be notified that a processor deallocation is about to happen. When these applications and kernel extensions receive notification, they are responsible for moving their bound threads and associated resources (such as timer request blocks) away from the last bind CPU ID and for adapting themselves to the new CPU configuration.

After notification, if some threads remain bound to the last bind CPU ID, the deallocation is aborted, the aborted deallocation is logged in the error log, and AIX continues using the ailing processor. When the processor ultimately fails, it causes a total system failure. Therefore, it is important that applications or kernel extensions receive notification of an impending processor deallocation and act on this notice.

Even in the rare cases that the deallocation cannot go through, Dynamic Processor Deallocation still gives advanced warning to system administrators. By recording the error in the error log, it gives them a chance to schedule a maintenance operation on the system to replace the ailing component before a global system failure occurs.

Processor Deallocation

The typical flow of events for processor deallocation is as follows:

1. The firmware detects that a recoverable error threshold has been reached by one of the processors.
2. The firmware error report is logged in the system error log, and, when AIX is executing on a machine that supports processor deallocation, AIX starts the deallocation process.
3. AIX notifies non-kernel processes and threads bound to the last bind CPU.
4. AIX waits up to ten minutes for all the bound threads to move away from the last bind CPU. If threads remain bound, AIX aborts the deallocation.
5. If all processes or threads are unbound from the ailing processor, the previously registered High Availability Event Handlers (HAEHs) are invoked. An HAEH might return an error that aborts the deallocation.
6. Unless aborted, the deallocation process ultimately stops the failing processor.

If there is a failure at any point of the deallocation, the failure and its cause are logged. The system administrator can look at the error log, take corrective action (when possible) and restart the deallocation. For instance, if the deallocation was aborted because an application did not unbind its bound threads, the system administrator can stop the application, restart the deallocation, and then restart the application.

Turning Processor Deallocation On and Off

Dynamic Processor Deallocation can be enabled or disabled by changing the value of the **cpuguard** attribute of the ODM object ***sys0***. The possible values for the attribute are **enable** and **disable**.

Beginning with AIX 5.2, the default is **enabled** (the attribute **cpuguard** has a value of **enable**). System administrators who want to disable this feature must use either the Web-based System Manager system menus, the SMIT System Environments menu, or the **chdev** command. (In previous AIX versions, the default was **disabled**.)

Note: If processor deallocation is turned off (disabled), the errors are still logged. The error log will contain an error such as CPU_FAILURE_PREDICTED, indicating that AIX was notified of a problem with a CPU.

Restarting an Aborted Processor Deallocation

Sometimes the processor deallocation fails because an application did not move its bound threads away from the last logical CPU. Once this problem has been fixed, either by unbinding (when it is safe to do so) or by stopping the application, the system administrator can restart the processor deallocation process using the **ha_star** command.

The syntax for this command is:

```
ha_star -C
```

where **-C** is for a CPU predictive failure event.

Processor State Considerations

Physical processors are represented in the ODM database by objects named **proc***n* where *n* is a decimal number that represents the physical processor number. Like any other device represented in the ODM database, processor objects have a state, such as Defined/Available, and attributes.

The state of a **proc** object is always Available as long as the corresponding processor is present, regardless of whether it is usable. The **state** attribute of a **proc** object indicates if the processor is used and, if not, the reason. This attribute can have three values:

enable	The processor is used.
disable	The processor has been dynamically deallocated.
faulty	The processor was declared defective by the firmware at startup time.

If an ailing processor is successfully deallocated, its state goes from **enable** to **disable**. Independently of AIX, this processor is also flagged in the firmware as defective. Upon reboot, the deallocated processor will not be available and will have its state set to **faulty**. The ODM **proc** object, however, is still marked Available. You must physically remove the defective CPU from the system board or remove the CPU board (if possible) for the **proc** object to change to Defined.

Example

In the following scenario, processor **proc4** is working correctly and is being used by the operating system, as shown in the following output:

```
# lsattr -EH -l proc4
attribute value   description  user_settable

state  enable   Processor state  False
type  PowerPC_RS64-III Processor type  False
 #
```

When processor **proc4** gets a predictive failure, it gets deallocated by the operating system, as shown in the following:

```
# lsattr -EH -l proc4
attribute value   description  user_settable

state  disable   Processor state  False
type  PowerPC_RS64-III Processor type  False
 #
```

At the next system restart, processor **proc4** is reported by firmware as defective, as shown in the following:

```
# lsattr -EH -l proc4
attribute value   description  user_settable

state  faulty   Processor state  False
type  PowerPC_RS64-III Processor type  False
 #
```

But the status of processor **proc4** remains Available, as shown in the following:

```
# lsdev -CH -l proc4
name  status   location  description

proc4  Available  00-04   Processor
#
```

Error Log Entries

Three different error log messages are associated with CPU deallocation. The following are examples.

errpt short format - summary

The following is an example of entries displayed by the **errpt** command (without options):

```
# errpt
IDENTIFIER      TIMESTAMP      T    C    RESOURCE_NAME    DESCRIPTION
804E987A        1008161399     I    O    proc4            CPU DEALLOCATED
8470267F        1008161299     T    S    proc4            CPU DEALLOCATION ABORTED
1B963892        1008160299     P    H    proc4            CPU FAILURE PREDICTED
#
```

- If processor deallocation is enabled, a `CPU FAILURE PREDICTED` message is always followed by either a `CPU DEALLOCATED` message or a `CPU DEALLOCATION ABORTED` message.
- If processor deallocation is not enabled, only the `CPU FAILURE PREDICTED` message is logged. Enabling processor deallocation any time after one or more `CPU FAILURE PREDICTED` messages have been logged initiates the deallocation process and results in a success or failure error log entry, as described above, for each processor reported failing.

errpt long format - detailed description

The following is the form of output obtained with **errpt -a**:

- **CPU_FAIL_PREDICTED**

  Error description: Predictive Processor Failure

  This error indicates that the hardware detected that a processor has a high probability to fail in a near future. It is always logged whether or not processor deallocation is enabled.

  DETAIL DATA: *Physical processor number*, *location*

  Example error log entry - long form

```
LABEL:   CPU_FAIL_PREDICTED
IDENTIFIER:  1655419A

Date/Time:  Thu Sep 30 13:42:11
Sequence Number: 53
Machine Id:  00002F0E4C00
Node Id:  auntbea
Class:   H
```

```
Type:   PEND
Resource Name:  proc25
Resource Class:  processor
Resource Type:  proc_rspc
Location:  00-25

Description
CPU FAILURE PREDICTED

Probable Causes
CPU FAILURE

Failure Causes
CPU FAILURE

 Recommended Actions
 ENSURE CPU GARD MODE IS ENABLED
 RUN SYSTEM DIAGNOSTICS.

Detail Data
PROBLEM DATA
0144 1000 0000 003A 8E00 9100 1842 1100 1999 0930 4019
0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 4942 4D00 5531
2E31 2D50 312D 4332 0000
0002 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000
... ... ... ... ...
```

- **CPU_DEALLOC_SUCCESS**

  Error Description: A processor has been successfully deallocated after detection of a predictive processor failure. This message is logged when processor deallocation is enabled, and when the CPU has been successfully deallocated.

  DETAIL DATA: *Logical CPU number of deallocated processor*.

  Example: error log entry - long form:

```
LABEL:   CPU_DEALLOC_SUCCESS
IDENTIFIER:  804E987A

Date/Time:  Thu Sep 30 13:44:13
Sequence Number: 63
Machine Id:  00002F0E4C00
Node Id:  auntbea
Class:   O
Type:   INFO
Resource Name:  proc24

Description
CPU DEALLOCATED


 Recommended Actions
 MAINTENANCE IS REQUIRED BECAUSE OF CPU FAILURE

Detail Data
LOGICAL DEALLOCATED CPU NUMBER

 0
```

  In this example, **proc24** was successfully deallocated and was logical CPU **0** when the failure occurred.

- **CPU_DEALLOC_FAIL**

Error Description: A processor deallocation, due to a predictive processor failure, was not successful. This message is logged when CPU deallocation is enabled, and when the CPU has not been successfully deallocated.

DETAIL DATA: *Reason code, logical CPU number, additional information depending of the type of failure.*

The reason code is a numeric hexadecimal value. The possible reason codes are:

2 One or more processes/threads remain bound to the last logical CPU. In this case, the detailed data give the PIDs of the offending processes.

3 A registered driver or kernel extension returned an error when notified. In this case, the detailed data field contains the name of the offending driver or kernel extension (ASCII encoded).

4 Deallocating a processor causes the machine to have less than two available CPUs. This operating system does not deallocate more than *N*-2 processors on an *N*-way machine to avoid confusing applications or kernel extensions using the total number of available processors to determine whether they are running on a Uni Processor (UP) system where it is safe to skip the use of multiprocessor locks, or a Symmetric Multi Processor (SMP).

200 (0xC8) Processor deallocation is disabled (the ODM attribute **cpuguard** has a value of **disable**). You normally do not see this error unless you start **ha_star** manually.

Examples: error log entries - long format

Example 1:

```
 LABEL:   CPU_DEALLOC_ABORTED
 IDENTIFIER:  8470267F
 Date/Time:  Thu Sep 30 13:41:10
 Sequence Number: 50
 Machine Id:  00002F0E4C00
 Node Id:  auntbea
 Class:   S
 Type:   TEMP
 Resource Name:  proc26

Description
CPU DEALLOCATION ABORTED

Probable Causes
SOFTWARE PROGRAM

Failure Causes
SOFTWARE PROGRAM

 Recommended Actions
 MAINTENANCE IS REQUIRED BECAUSE OF CPU FAILURE
 SEE USER DOCUMENTATION FOR CPU GARD

Detail Data
DEALLOCATION ABORTED CAUSE
0000 0003
DEALLOCATION ABORTED DATA
6676 6861 6568 3200
```

In this example, the deallocation for **proc26** failed. The reason code **3** means that a kernel extension returned an error to the kernel notification routine. The `DEALLOCATION ABORTED DATA` above spells **fvhaeh2**, which is the name the extension used when registering with the kernel.

Example 2:

```
 LABEL:   CPU_DEALLOC_ABORTED
 IDENTIFIER:  8470267F
 Date/Time:  Thu Sep 30 14:00:22
 Sequence Number: 71
 Machine Id:  00002F0E4C00
 Node Id:  auntbea
 Class:   S
```

```
 Type:   TEMP
 Resource Name:  proc19

Description
CPU DEALLOCATION ABORTED

Probable Causes
SOFTWARE PROGRAM

Failure Causes
SOFTWARE PROGRAM

 Recommended Actions
 MAINTENANCE IS REQUIRED BECAUSE OF CPU FAILURE;
 SEE USER DOCUMENTATION FOR CPU GARD

Detail Data
DEALLOCATION ABORTED CAUSE
0000 0002
DEALLOCATION ABORTED DATA
0000 0000 0000 4F4A
```

In this example, the deallocation for **proc19** failed. The reason code **2** indicates thread(s) were bound to the last logical processor and did not unbind after receiving the SIGCPUFAIL signal. The DEALLOCATION ABORTED DATA shows that these threads belonged to process **0x4F4A**.

Options of the **ps** command (-o THREAD, -o BND) allow you to list all threads or processes along with the number of the CPU they are bound to, when applicable.

Example 3:

```
 LABEL:   CPU_DEALLOC_ABORTED
 IDENTIFIER:  8470267F

 Date/Time:  Thu Sep 30 14:37:34
 Sequence Number: 106
 Machine Id:  00002F0E4C00
 Node Id:  auntbea
 Class:   S
 Type:   TEMP
 Resource Name:  proc2

Description
CPU DEALLOCATION ABORTED

Probable Causes
SOFTWARE PROGRAM

Failure Causes
SOFTWARE PROGRAM

 Recommended Actions
 MAINTENANCE IS REQUIRED BECAUSE OF CPU FAILURE
 SEE USER DOCUMENTATION FOR CPU GARD

Detail Data
DEALLOCATION ABORTED CAUSE
0000 0004
DEALLOCATION ABORTED DATA
0000 0000 0000 0000
```

In this example, the deallocation of **proc2** failed because there were two or fewer active processors at the time of failure (reason code **4**).

64-Bit Mode

64-bit mode allows for fast access to large amounts of data and efficient handling of 64-bit data types. This section provides general information about 64-bit mode and information about how to run 64-bit mode on AIX. This section presents information about the following:

- Filesets Needed for 64-Bit Mode
- Hardware Required for 64-Bit Mode
- Kernel Extensions vs. 64-bit Kernel
- Changing from 32-Bit to 64-Bit
- Changing from 64-Bit to 32-Bit
- 32-Bit and 64-Bit Performance Comparisons

Filesets Needed for 64-Bit Mode

The base operating system 64-bit runtime fileset is **bos.64bit**. Installing **bos.64bit** also installs the **/etc/methods/cfg64** file. The **/etc/methods/cfg64** file provides the option of enabling or disabling the 64-bit environment via SMIT, which updates the **/etc/inittab** file with the load64bit line. (Simply adding the load64bit line does not enable the 64-bit environment).

The command **lslpp -l bos.64bit** reveals if this fileset is installed. The **bos.64bit** fileset is on the AIX media; however, installing the **bos.64bit** fileset does not ensure that you will be able to run 64-bit software. If the **bos.64bit** fileset is installed on 32-bit hardware, you should be able to compile 64-bit software, but you cannot run 64-bit programs on 32-bit hardware.

The **syscalls64** extension must be loaded in order to run a 64-bit executable. This is done from the `load64bit` entry in the **inittab** file. You must load the **syscalls64** extension even when running a 64-bit kernel on 64-bit hardware.

Hardware Required for 64-Bit Mode

You must have 64-bit hardware to run 64-bit applications. To determine whether your system has 32-bit or 64-bit hardware architecture:

1. Log in as a root user.
2. At the command line, enter `bootinfo -y`.

This produces the output of either `32` or `64`, depending on whether the hardware architecture is 32-bit or 64-bit. In addition, if you enter **lsattr -El proc0** at any version of AIX, the type of processor for your server displays.

Kernel Extensions vs. 64-bit Kernel

To determine if the 64-bit kernel extension is loaded, at the command line, enter `genkex |grep 64`.

Information similar to the following displays:

```
149bf58 a3ec /usr/lib/drivers/syscalls64.ext
```

Note: Having the driver extensions does not mean that the kernel is a 64-bit kernel. A 64-bit kernel became available at the AIX 5.1 level. The driver extensions simply allow the 64-bit application to be compiled by a 32-bit kernel. If the 32-bit kernel has a 64-bit processor, **syscalls64.ext** allows the 64-bit application to execute. Yet at the AIX 5L level, a 64-bit kernel and a 64-bit processor has better performance with 64-bit applications.

Changing from 32-Bit to 64-Bit

To truly change the kernel to 64-bit, the system must be at the AIX 5.1 or AIX 5.2 levels. To change to a 64-bit kernel, enter the following commands:

```
ln -sf /usr/lib/boot/unix_64 /unix
ln -sf /usr/lib/boot/unix_64 /usr/lib/boot/unix
lslv -m hd5
bosboot -ad /dev/ipldevice
shutdown -Fr
```

Changing from 64-Bit to 32-Bit

To change the kernel back to 32-bit, enter the following commands:

```
ln -sf /usr/lib/boot/unix_mp /unix
ln -sf /usr/lib/boot/unix_mp /usr/lib/boot/unix
lslv -m hd5
bosboot -ad /dev/ipldevice
shutdown -Fr
```

32-Bit and 64-Bit Performance Comparisons

In most cases, running 32-bit applications on 64-bit hardware is not a problem, because 64-bit hardware can run both 64-bit and 32-bit software. However, 32-bit hardware cannot run 64-bit software. To find out if any performance issues exist for applications that are running on the system, refer to those application's user guides for their recommended running environment.

Chapter 8. Process Management

The process is the entity that the operating system uses to control the use of system resources. *Threads* can control processor-time consumption, but most system management tools still require you to refer to the process in which a thread is running, rather than to the thread itself.

See the *AIX 5L Version 5.3 System User's Guide: Operating System and Devices* for basic information on managing your own processes; for example, restarting or stopping a process that you started or scheduling a process for a later time.

For task procedures, see Monitoring and Managing Processes in the *AIX 5L Version 5.3 System Management Guide: Operating System and Devices*.

Tools are available to:

- Observe the creation, cancellation, identity, and resource consumption of processes
  - The **ps** command is used to report process IDs, users, CPU-time consumption, and other attributes.
  - The **who -u** command reports the shell process ID of logged-on users.
  - The **svmon** command is used to report process real-memory consumption. (See *Performance Toolbox Version 2 and 3 for AIX: Guide and Reference* for information on the **svmon** command.)
  - The **acct** command mechanism writes records at process termination summarizing the process's resource use. (See how to set up an accounting system in "Accounting Overview" on page 111.)
- Control the priority level at which a process contends for the CPU.
  - The **nice** command causes a command to be run with a specified process priority. (See *AIX 5L Version 5.3 System User's Guide: Operating System and Devices*.)
  - The **renice** command changes the priority of a given process.
- Terminate processes that are out of control.
  - The **kill** command sends a termination signal to one or more processes.

Chapter 9. Workload Management

Workload Manager (WLM) is designed to provide the system administrator with increased control over how the scheduler virtual memory manager (VMM) and the disk I/O subsystem allocate resources to processes. You can use WLM to prevent different classes of jobs from interfering with each other and to allocate resources based on the requirements of different groups of users.

Attention: Efficient use of WLM requires extensive knowledge of existing system processes and performance. If the system administrator configures WLM with extreme or inaccurate values, performance will be significantly degraded.

WLM is primarily intended for use with large systems. Large systems are often used for server consolidation, in which workloads from many different server systems (such as printer, database, general user, and transaction processing systems) are combined into a single large system to reduce the cost of system maintenance. These workloads often interfere with each other and have different goals and service agreements.

WLM also provides isolation between user communities with very different system behaviors. This can prevent effective starvation of workloads with certain behaviors (for example, interactive or low CPU usage jobs) by workloads with other behaviors (for example, batch or high memory usage jobs).

Also, WLM ties into the accounting subsystem (see “Accounting Overview” on page 111) allowing users to do resource usage accounting per WLM class in addition to the standard accounting per user or group.

WLM Concepts

With WLM, you can create different classes of service for jobs, as well as specify attributes for those classes. These attributes specify minimum and maximum amounts of CPU, physical memory, and disk I/O throughput to be allocated to a class. WLM then assigns jobs automatically to classes using class assignment rules provided by a system administrator. These assignment rules are based on the values of a set of attributes for a process. Either the system administrator or a privileged user can also manually assign jobs to classes, overriding the automatic assignment.

Terminology

class	A *class* is a collection of processes and their associated threads. A class has a single set of resource-limitation values and target shares. **class** is used to describe both subclasses and superclasses.
superclass	A *superclass* is a class that has subclasses associated with it. No processes can belong to a superclass without also belonging to a subclass. A superclass has a set of class-assignment rules that determines which processes are assigned to the superclass. A superclass also has a set of resource-limitation values and resource target shares that determines the amount of resources that can be used by processes which belong to the superclass. These resources are divided among the subclasses based on the resources limitation values and resource target shares of the subclasses.

subclasses	A *subclass* is a class associated with exactly one superclass. Every process in a subclass is also a member of its superclass. Subclasses have access only to resources that are available to the superclass. A subclass has a set of class assignment rules that determines which of the processes assigned to the superclass belong to the subclass. A subclass also has a set of resource-limitation values and resource target shares that determines the resources that can be used by processes in the subclass. These resource-limitation values and resource target shares indicate how much of the resources available to the superclass (the target for the superclass) can be used by processes in the subclass. WLM administration can be done using either the Web-based System Manager, SMIT, or the WLM command-line interface.
classification mechanism	A *classification mechanism* is a set of class assignment rules that determines which processes are assigned to which classes (superclasses or subclasses within superclasses).
class assignment rule	A *class assignment rule* indicates which values within a set of process attributes result in a process being assigned to a particular class (superclass or subclass within a superclass).
process attribute value	A *process attribute value* is the value that a process has for a process attribute. The process attributes can include attributes such as user ID, group ID, and application path name.
resource-limitation values	*Resource-limitation values* are a set of values that WLM maintains for a set of resource utilization values. These limits are completely independent of the resource limits specified with the **setrlimit** subroutine.
resource target share	*Resource target shares* are the shares of a resource that are available to a class (subclass or superclass). These shares are used with other class shares (subclass or superclass) at the same level and tier to determine the desired distribution of the resources between classes at that level and tier.
resource-utilization value	A *resource-utilization value* is the amount of a resource that a process or set of processes is currently using in a system. Whether it is one process or a set of processes is determined by the scope of process resource collection.
scope-of-resource collection	The *scope-of-resource collection* is the level at which resource utilization is collected and the level at which resource-limitation values are applied. This might be at the level of each process in a class, the level of the sum across every process in a class owned by each user, or the level of the sum across every process in a class. The only scope currently supported is the latter.
process class properties	The *process class properties* are the set of properties that are given to a process based on the classes (subclass and superclass) to which it is assigned.
class authorizations	The *class authorizations* are a set of rules that indicates which users and groups are allowed to perform operations on a class or processes and threads in a class. This includes the authorization to manually assign processes to a class or to create subclasses of a superclass.
class tier	The *class tier* value is the position of the class within the hierarchy of resource limitation desirability for all classes. The resource limits (including the resource targets) for all classes in a tier are satisfied before any resource is provided to lower tier classes. Tiers are provided at both the superclass and subclass levels. Resources are provided to superclasses based on their tiers. Within a superclass, resources are given to subclasses based on their tier values within the superclass. Thus, superclass tier is the major differentiator in resource distribution; the subclass tier provides an additional smaller differentiator within a superclass.

Classes

WLM allows system administrators to define classes and define for each class a set of attributes and resource limits. The processes are assigned to classes based on criteria provided by the system administrator. The resource entitlements and limits are enforced at the class level. This method of defining classes of service and regulating the resource utilization of each class of applications prevents applications with very different resource use patterns from interfering with each other when they share a single server.

WLM supports a hierarchy of classes with two levels:

- The resources of the system are distributed among superclasses according to the resource entitlements for each superclass. The system administrator defines resource entitlements.
- In turn, each superclass can have subclasses. The resources allocated to the superclass are distributed among the subclasses according to the resource entitlements given to each subclass.
- The system administrator can delegate the administration of the subclasses of each superclass to a *superclass administrator* or to a group of superclass administrators.
- In AIX 5.2 and later, WLM supports up to 69 superclasses (64 user-defined) and 64 subclasses per superclass (61 user-defined).
- Depending on the needs of the organization, a system administrator can decide to use only superclasses or to use superclasses and subclasses.

Note: Throughout this discussion of WLM, the term *class* applies to both superclasses and subclasses. If discussion applies only to a specific class type, that type is explicitly mentioned.

Process Assignment to Classes

The processes are assigned to a class, using class-assignment rules provided by the system administrator. The classification criteria are based on the value of a set of attributes of the process such as user ID, group ID, name of the application file, type of process, and application tag.

A defined set of rules is used to determine the superclass a process is assigned to. If this superclass has subclasses defined, there is another set of rules for this superclass to determine which subclass is assigned to which process. This automatic assignment process also takes into account the **inheritance** attributes of both the superclass and the subclass. (For information on class attributes, see “WLM Class Attributes” on page 86.)

The automatic class assignment is done when a process calls the **exec** subroutine. The class assignment is reevaluated when a process uses a subroutine that can alter a process attribute used for classification purposes. Examples are the **setuid**, **setgid**, **setpri**, and **plock** subroutines.

In addition to this automatic class assignment, a user with the proper authority can manually assign processes or groups of processes to a specific superclass or subclass.

Resource Control

WLM allows management of resources in two ways: as a percentage of available resources or as total resource usage. Resources that can be controlled on a percentage basis include the following:

- Processor use of the threads of type SCHED_OTHER in a class. This is the sum of all of the processor cycles consumed by every thread in the class. Fixed-priority threads are non-adjustable. Therefore, they cannot be altered, and they can exceed the processor usage target.
- Physical memory utilization of the processes in a class. This is the sum of all the memory pages that belong to the processes in the class.
- Disk I/O bandwidth of the class. This is the bandwidth (in 512–byte blocks per second) of all the I/Os started by threads in the class on each disk device accessed by the class.

Resources that can be controlled on a total usage basis fall into one of two categories: class totals or process totals. The class totals category includes:

Number of processes in a class
This is the number of processes that are active in a class at one time.

Number of threads in a class
This is the number of threads that are active in a class at one time.

Number of logins in a class
This is the number of login sessions that are active in a class at one time.

The process totals category includes:

Total CPU time
This is the total accumulated CPU time for a single process.

Total disk I/O
This is the total accumulated blocks of disk I/O for a single process.

Total connect time
This is total amount of time that a login session can be active.

Resource Entitlements

WLM allows system administrators to specify per-class resource entitlements independently for each resource type. These entitlements can be specified by indicating the following:

- The target for usage of different types of resources. This target is specified with shares. The shares are specified as relative amounts of usage between different classes. For instance, if two classes have respectively 1 and 3 shares of CPU and are the only classes active at this time, their percentage goal used by WLM for its CPU regulation will be 25% and 75%, respectively. The target percentages are calculated for classes in each tier based on the number of active shares in the tier and the amount of resource available to the tier.
- Minimum and maximum limits. These limits are specified as percentages of the total resource available. WLM supports two kinds of maximum limits:
  - A soft maximum limit indicates the maximum amount of the resource that can be made available when there is contention for the resource. This maximum can be exceeded if there is no contention; that is, if no one else requires the resource.
  - A hard maximum limit indicates the maximum amount of the resource that can be made available regardless of whether there is contention on the resource. Fixed-priority threads, however, are not subject to these same rules and therefore can exceed the limit.
- Total limits. The total limits are strictly enforced. If a process exceeds one of its total consumption limits, it will be terminated. If a class is at one of its total limits, any operation that would result in the creation of another instance of that resource will fail.

In most cases, soft maximum limits are sufficient to ensure that resource entitlements are met and enforced. Using hard maximum limits may result in unused system resources since these are strictly enforced, even when there is no contention for the resource. Careful consideration must be made when using hard maximum limits since these can greatly affect system or application performance if set too low. Total limits should also be used with caution, since these could result in process termination or failure to function as intended.

In active mode, WLM attempts to keep active classes close to their targets. Since there are few constraints on the values of the various limits, the sum of any of the limits across all classes could far exceed 100%. In this case, if all of the classes are active, the limit cannot be reached by all classes. WLM regulates the processor consumption by adjusting the scheduling priorities of the non-fixed priority threads in the system according to how the class they belong to is performing, relative to its limits and target. This approach guarantees a processor consumption averaged over a given period of time, rather than the processor consumption over very short intervals (for example, 10 ms).

For example, if class A is the only active class, with a processor minimum of 0% and a processor target of 60 shares, then it gets 100% of the processor. If class B, with a processor minimum limit of 0% and a processor target of 40 shares, becomes active, then the class A processor utilization progressively decreases to 60% and the class B processor utilization increases from 0% to 40%. The system stabilizes at 60% and 40% processor utilization, respectively, in a matter of seconds.

This example supposes that there is no memory contention between the classes. Under regular working conditions, the limits you set for processor and memory are interdependent. For example, a class may be unable to reach its target or even its minimum processor allocation if the maximum limit on its memory usage is too low compared to its working set.

To help refine the class definition and class limits for a given set of applications, WLM provides the **wlmstat** reporting tool, which shows the amount of resource currently being used by each class. A graphical display tool, **wlmmon**, is also provided for system monitoring.

WLM Modes of Operation

WLM can be used to regulate resource consumption as per-class percentages, per-class totals, or per-process totals. Regulation for all resource types can be enabled by running WLM in active mode.

Optionally, you can start a mode of WLM that classifies new and existing processes and monitors the resource usage of the various classes, without attempting to regulate this usage. This mode is called the *passive mode*.

The passive mode can be used when configuring WLM on a new system to verify the classification and assignment rules, and to establish a base line of resource utilization for the various classes when WLM does *not* regulate the processor and memory allocation. This should give a basis for system administrators to decide how to apply the resource shares and resource limits (if needed) to favor critical applications and restrict less important work in order to meet their business goals.

If processor time is the only resource that you are interested in regulating, you can choose to run WLM in active mode for processor and passive mode for all other resources. This mode is called *cpu only* mode. If you want to regulate per-class percentages, but neither of the total resource types, the total resource accounting and regulation can be disabled for per-class totals, per-process totals, or both. In all modes, you have the option of disabling resource set binding.

Dynamic Control

When WLM is active, any parameter of the current configuration can be modified at any time, including the attributes of a class, its resource shares and limits, the assignment rules, and adding new classes or deleting existing classes. This can be done in several ways, such as:

- Modifying the property files for the currently active configuration (directory pointed to by the symbolic link **/etc/wlm/current**) and refreshing WLM by using the **wlmcntrl** command to use the new parameters.
- Creating another configuration with a different set of parameters and updating WLM to load the parameters of the new configuration, thus making it the current configuration.
- Modifying some of the parameters of the currently active configuration using the WLM command line interface (the **mkclass**, **chclass**, and **rmclass** commands).
- Modifying some of the parameters of the currently active configuration from an application using the WLM APIs.

Automatic switches to a new configuration at specified times of day can be accomplished using *configuration sets*. Configuration sets allow the administrator to specify a set of configurations to be used, and a time range for which each will be active.

Monitoring Tools

Use the following WLM commands to display WLM statistics and monitor the operation of WLM:

- The **wlmstat** command is text oriented and displays statistics as text (percentage of resource utilization per class for all the resource types managed by WLM).
- The **wlmmon** command gives a graphical view of per-class resource utilization and WLM regulation.
- The **wlmperf** command is an optional tool available with the Performance Toolbox and provides more capabilities, such as long-term record and replay.

WLM API

An Application Programming Interface (API) allows applications to perform any operation that can be done using the WLM command-line interface such as:

- Create, modify, or delete classes
- Change class attributes or resource shares and limits
- Manually assign processes to classes.

In addition, the API allows applications to set an application-defined classification attribute called *tag*. Setting this tag using a set of values provided by the system administrator (through the application user documentation) allows discrimination between several instances of the same application. The different classes can therefore be classified with different resource entitlements.

Per Class Accounting

The AIX accounting system utility lets you collect and report the use of various system resources by user, group, or WLM class. When process accounting is turned on, the operating system records statistics about the process resource usage in an accounting file when the process exits. Beginning with AIX 5.1, this accounting record includes a 64-bit numeric key representing the name of the WLM class that the process belonged to. (See “Accounting Overview” on page 111 for more information about the accounting system utility.)

The accounting subsystem uses a 64-bit key instead of the full 34-character class name to save space (otherwise the change would practically double the size of the accounting record). When the accounting command is run to extract the per-process data, the key is translated back into a class name using the above-mentioned routine. This translation uses the class names currently in the WLM configuration files. So, if a class has been deleted between the time the accounting record was written, when the process terminated, and the time the accounting report is run, the class name corresponding to the key cannot be found, and the class displays as `Unknown`.

To keep accurate records of the resource usage of classes deleted during an accounting period, do one of the following:

- Instead of deleting the class, keep the class name in the classes file and remove the class from the rules file so that no process can be assigned to it. Then you can delete the class after the accounting report has been generated at the end of the accounting period.
- Or, delete the class from the configuration it belongs to, and keep the class name in the classes file in a ″dummy″ configuration (one that is never activated) until after the accounting records for the period have been generated.

Overview of WLM Classes

Workload Manager helps you control the allocation of system resources by defining classes of service and allocating resources to each of these classes. Each class has a set of attributes that determine what its resource entitlements are, as well as other behaviors. Every process on the system is classified into a service class, and is thus subject to enforcement of the resource entitlements and behaviors for that class. Processes are assigned to a class either manually using manual assignment, or automatically according to user-defined classification rules.

WLM supports two levels of classes: *superclasses* and *subclasses*. Superclasses are given resource entitlements based on available system resources, and subclasses are given resource entitlements relative to the entitlements of their associated superclass. Optionally, you can define subclasses to allow for more granular control of the processes in a superclass. You can also delegate the responsibility of defining subclasses by specifying an adminuser or admingroup for a superclass.

For both the superclass and subclass levels, you can define classes, resource shares and limits, and rules using SMIT, Web-based System Manager, or the command-line interface. Applications can use the WLM APIs. Configuration definitions are kept in a set of text files called the WLM *property files*.

A class name is up to 16 characters in length and can contain only uppercase and lowercase letters, numbers and underscores (_). For a given WLM configuration, each superclass name must be unique. Each subclass name must be unique within that superclasses, but it can match subclass names in other superclasses. To uniquely identify every subclass, the full name of a subclass is composed of the superclass name and the subclass name separated by a dot; for example: *Super.Sub*.

Superclasses

The system administrator can define up to 64 superclasses. In addition, the following five superclasses are automatically created:

Default **superclass**
: Is the default superclass and is always defined. All non-root processes that are not automatically assigned to a specific superclass are assigned to the Default superclass. Other processes can also be assigned to the *Default* superclass by providing specific assignment rules.

System **superclass**
: Has all privileged (root) processes assigned to it if those processes are not assigned by rules to a specific class. This superclass also collects the memory pages belonging to kernel memory segments and kernel processes. Other processes can also be assigned to the System superclass by providing specific assignment rules for this superclass. This superclass has a memory minimum limit of 1% as the default.

Shared **superclass**
: Receives the memory pages that are shared by processes in more than one superclass. This includes pages in shared memory regions and pages in files that are used by processes in more than one superclass (or in subclasses of different superclasses). Shared memory and files that are used by multiple processes that all belong to a single superclass (or subclasses of the same superclass) are associated with that superclass. Only when a process from a different superclass accesses the shared memory region or file are the pages placed in the Shared superclass. This superclass can have only physical memory shares and limits applied to it. It cannot have shares or limits for the other resource types, subclasses, or assignment rules specified. Whether a memory segment shared by processes in different subclasses of the same superclass is classified into the *Shared* subclass or remains in its original subclass depends on the value of the **localshm** attribute of the original subclass.

Unclassified **superclass**
: Is a memory allocation for unclassified processes. The processes in existence at the time that WLM is started are classified according to the assignment rules of the WLM configuration being loaded. During this initial classification, all the memory pages attached to each process are ″charged″ either to the superclass that the process belongs to (when not shared, or shared by processes in the same superclass), or to the *Shared* superclass when shared by processes in different superclasses.

  However, a few pages cannot be directly tied to any processes (and thus to any class) at the time of this classification, and this memory is charged to the *Unclassified* superclass. Most of this memory is correctly reclassified over time, when it is either accessed by a process, or released and reallocated to a process after WLM is started. There are no processes in the *Unclassified*

superclass. This superclass can have physical memory shares and limits applied to it. It cannot have shares or limits for the other resource types, subclasses, or assignment rules specified.

Unmanaged **superclass**
Is always defined. No processes will be assigned to this class. This class will be used to accumulate the memory usage for all pinned pages in the system. The CPU utilization for the wait processes is intentionally not accumulated in any class. The memory in this class is not managed by WLM and is not considered to part of the available system resources. Otherwise, the system would always seem to be at 100% CPU utilization, which could be misleading for users when looking at the WLM statistics.

Subclasses

The system administrator or a superclass administrator can define up to 61 subclasses. In addition, two special subclasses, *Default* and *Shared*, are always defined.

Default **subclass**
Is the default subclass and is always defined. All processes that are not automatically assigned to a specific subclass of the superclass are assigned to the *Default* subclass. You can also assign other processes to the *Default* subclass by providing specific assignment rules.

Shared **subclass**
Receives all the memory pages that are used by processes in more than one subclass of the superclass. Included are pages in shared memory regions and pages in files that are used by processes in more than one subclass of the same superclass. Shared memory and files that are used by multiple processes that all belong to a single subclass are associated with that subclass. Only when a process from a different subclass of the same superclass accesses the shared memory region or file are the pages placed in the *Shared* subclass of the superclass. There are no processes in the *Shared* subclass. This subclass can have only physical memory shares and limits applied to it, and it cannot have shares or limits for the other resource types or assignment rules specified. Whether a memory segment shared by processes in different subclasses of the same superclass is classified into the *Shared* subclass or remains in its original subclass depends on the value of the **localshm** attribute of the original subclass.

WLM Class Attributes

The attributes of a WLM class are:

Class Name
Can be up to 16 characters in length and can only contain uppercase and lowercase letters, numbers and underscores (_).

Tier A number between 0 and 9 used to prioritize resource allocation between classes.

Inheritance
Specifies whether a child process inherits the class assignment from its parent.

localshm
Prevents memory segments belonging to one class from migrating to Shared class.

Administrator (adminuser, admingroup, authgroup) (superclass only)
Delegates the administration of a superclass.

Authorization (authuser, authgroup)
Delegates the right to manually assign a process to a class.

Resource Set (rset)
Limits the set of resources a given class has access to in terms of CPUs (processor set).

Tier Attribute

Tiers represent the order in which system resources are allocated to WLM classes. The administrator can define classes in up to 10 tiers, numbered 0 through 9, with 0 being the highest or most important tier. The

amount of resources available to tier 0 is all available system resources. The amount of resources available to the lower (higher number) tiers, is the amount of resources that is unused by all higher tiers. Target consumption percentages for classes are based on the number of active shares in its tier, and the amount of resource available to the tier. Since tier 0 is the only tier that is guaranteed to always have resources available to it, it is recommended that processes that are essential to system operation be classified in a class in this tier. If no tier value is specified for a class, it will be put in tier 0.

A tier can be specified at both the superclass and the subclass levels. Superclass tiers are used to specify resource allocation priority between superclasses. Subclass tiers are used to specify resource allocation priority between subclasses of the same superclass. There is no relationship among sub-tiers of different superclasses.

Inheritance Attribute

The inheritance attribute of a class indicates whether processes in the class should be automatically reclassified when one of the classification attributes of the process changes. When a new process is created with the **fork** subroutine, it automatically inherits its parent's class, whether or not inheritance is enabled. One exception is when the parent process has a tag, has its **inherit tag at fork** set to off, and class inheritance is off for the parent's class. In this case, the child process is reclassified according to the classification rules.

When inheritance is not enabled for a class, any process in the class is automatically classified according to the classification rules after calling any service that changes a process attribute that is used in the rule. The most common of these calls is the **exec** subroutine, but other subroutines that can change classification include **setuid**, **setgid**, **plock**, **setpri**, and **wlm_set_tag**. When inheritance is enabled, the process is not subject to reclassification based on the classification rules, and will remain in its current class. Manual assignment has priority over inheritance and can be used to reclassify processes that are in a class with inheritance enabled.

The specified value for this attribute can be either yes or no. If unspecified, inheritance will not be enabled for a class.

This attribute can be specified at both superclass and subclass level. For a subclass of a given superclass:

- If the inheritance attribute is set to yes at both the superclass and the subclass levels, a child of a process in the subclass will remain in the same subclass.
- If the inheritance attribute is set to yes for the superclass and no (or unspecified) for the subclass, a child of a process in the subclass will remain in the same superclass and will be classified in one of its subclasses according to the assignment rules for the superclass.
- If the inheritance attribute is no (or is unspecified) for the superclass and is set to yes for the subclass, a child of a process in the subclass will be submitted to the automatic assignment rules for the superclasses.
  - If the process is classified by the rules in the same superclass, then it will remain in the subclass (it will not be submitted to the subclass's assignment rules).
  - If the process is classified by the superclass's rules in a different superclass, then the subclass assignment rules of the new superclass are applied to determine the subclass of the new superclass the process will be assigned to.
- If both superclass and subclass inheritance attributes are set to no (or are unspecified), then a child of a process in the subclass will be submitted to the standard automatic assignment.

localshm Attribute

The localshm attribute can be specified at the superclass and the subclass levels. It is used to prevent memory segments belonging to one class from migrating to the *Shared* superclass or subclass when accessed by processes in other classes. The possible values for the attribute are yes or no. A value of *yes* means that shared memory segments in this class must remain local to the class and not migrate to the appropriate *Shared* class. A value of *no* is the default when the attribute is not specified.

Memory segments are classified on page faults. When a segment is created, it is marked as belonging to the *Unclassified* superclass. On the first page fault on the segment, this segment is classified into the same class as the faulting process. If, later on, a process belonging to a different class than the segment page faults on this segment, WLM considers whether the segment needs to be reclassified into the appropriate *Shared* class (superclass or subclass). If the faulting process and the segment belong to different superclasses, one of the following occurs:

- If the segment's superclass has the localshm attribute set to *yes*, the segment remains in its current superclass. If the segment's subclass has the localshm attribute set to *yes*, the segment remains in its current subclass. If the superclass localshm attribute is set to *yes* but its subclass attribute is set to *no*, it goes into the *Shared* subclass of the current superclass.
- If the segment's superclass has the localshm attribute set to *no*, the segment goes to *Shared* superclass. This is the default action.

If the faulting process and the segment belong to different subclasses of the same superclass, and the segment's subclass has the localshm attribute set to *yes*, the segment remains in the current class (superclass and subclass). Otherwise, the segment goes to the *Shared* subclass of the superclass.

Of course, if the faulting process and the segment belong to the same class (same superclass and same subclass), the segment is not reclassified regardless of the values of the localshm attributes.

Administrator Attributes

Note: These attributes are valid only for superclasses.

The adminuser and admingroup attributes are used to delegate the superclass administration to a user or group of users:

- adminuser specifies the name of the user (as listed in **/etc/passwd**) authorized to perform administration tasks on the superclass.
- admingroup specifies the name of the group of users (as listed in **/etc/group**) authorized to perform administration tasks on the superclass.

Only one value (user or group) is allowed for each attribute. Either of them, none, or both can be specified. The user or group will have authority to do the following:

- Create and delete subclasses
- Change the attributes and resource shares and limits for the subclasses
- Define, remove or modify subclass assignment rules
- Refresh (update) the active WLM configuration for the superclass.

Authorization Attributes

The authuser and authgroup attributes are valid for all classes. They are used to specify the user or group authorized to manually assign processes to the class (superclass or subclass). When manually assigning a process (or a group of processes) to a superclass, the assignment rules for the superclass are used to determine to which subclass of the superclass each process will be assigned.

Only one value (user or group) is allowed for each attribute. Either of them, none, or both can be specified.

Resource Set Attribute

The resource set attribute (called *rset*) can be specified for any class. Its value is the name of a resource set defined by the system administrator. The *rset* attribute represents a subset of the CPU resource available on the system (processor set). The default is ″system,″ which gives access to all the CPU resources available on the system. The only restriction is that if an *rset* is specified for a subclass, the set of CPUs in the the set must be a subset of the CPUs available to the superclass. (For detailed information, see the **mkrset** command description in the *AIX 5L Version 5.3 Commands Reference*.)

Note: Carefully consider assigning resource sets to any class that is not in tier 0. Because lower tiers only have access to the resources that are unused by the higher tiers, restricting a non-tier-0 class to a subset of the CPUs on the system could result in starvation if there is no CPU time available on those CPUs.

Process to Class Assignment in WLM

In WLM, processes can be classified in either of two ways:

- A process is automatically assigned using assignment rules when process classification attributes change. When WLM is running in active mode, this automatic assignment is always in effect (it cannot be turned off). This is the most common way that processes are classified.
- A selected process or group of processes can be manually assigned to a class by a user with the required authority on both the processes and the target class. Manual assignment can be done using by a WLM command, which could be invoked directly or through SMIT or Web-based System Manager, or by an application using a function of the WLM Application Programming Interface. This manual assignment overrides the automatic assignment and inheritance.

Automatic Assignment

The automatic assignment of processes to classes uses a set of class-assignment rules specified by a WLM administrator. There are two levels of assignment rules:

- A set of assignment rules at the WLM configuration level used to determine which superclass a given process is assigned to.
- Each superclass with subclasses defined, in turn has a set of assignment rules used to determine which subclass of the superclass the process is assigned to.

The assignment rules at both levels are based on the values of a set of process attributes. These attributes are as follows:

- Process user ID
- Process group ID
- Path name of the application (program) executed
- Type of the process (`32bit` or `64bit`, for example)
- Process tag.

The tag is a process attribute, defined as a character string, that an application can set by program, using the WLM API.

The classification is done whenever an attribute changes by comparing the value of these process attributes against lists of possible values given in the class assignment rules file (called **rules**). The comparison determines which rule is a match for the current value of the process attributes.

A class assignment rule is a text string that includes the following fields, separated by one or more spaces:

Name	Must contain the name of a class which is defined in the class file corresponding to the level of the **rules** file (superclass or subclass). Class names can contain only uppercase and lowercase letters, numbers, and underscores and can be up to 16 characters in length. No assignment rule can be specified for the system defined classes Unclassified, Unmanaged and Shared.
Reserved	Reserved for future extension. Its value must be a hyphen (-), and it must be present.
User	Can contain either a hyphen (-) or at least one valid user name (as defined in the **/etc/passwd** file). The list is composed of one or more names, separated by a comma (,). An exclamation mark (!) can be used before a name to exclude a given user from the class. Patterns can be specified to match a set of user names, using full Korn shell pattern-matching syntax. If there are no valid user names, the rule is ignored.

Group
: Can contain either a hyphen (-) or at least one valid group name (as defined in the **/etc/group** file). The list is composed of one or more names, separated by a comma (,). An exclamation mark (!) can be used before a name to exclude a given group from the class. Patterns can be specified to match a set of user names using full Korn shell pattern matching syntax. If there are no valid group names, the rule is ignored.

Application
: Can contain either a hyphen (-) or a list of application path names. This is the path name of the applications (programs) executed by processes included in the class. The application names will be either full path names or Korn shell patterns that match path names. The list is composed of one or more path names, separated by a comma (,). An exclamation mark (!) can be used before a name to exclude a given application.

  At least one application in the list must be found at load time or the rule is ignored. Rules that are initially ignored for this reason might become effective later on if a file system is mounted that contains one or more applications in the list.

Type
: Can contain either a hyphen (-) or a list of process attributes. The possible values for these attributes are:

  - **32bit**: the process is a 32-bit process
  - **64bit**: the process is a 64-bit process
  - **plock**: the process called the **plock** subroutine to pin memory
  - **fixed**: the process is a fixed priority process (SCHED_FIFO or SCHED_RR)

  The **fixed** type is for classification purposes only. WLM does not regulate the processor use of fixed priority processes or threads. Because fixed priority processes have the potential to cause deprivation among other processes in a class, this classification attribute is provided to allow isolation of these jobs. This attribute can also be used to report consumption of such processes.

  The value of the **type** field can be a combination of one or more of the above attributes separated by a plus (+). The **32bit** and **64bit** values are mutually exclusive.

Tag
: May contain either a hyphen (-) or a list of application tags. An application tag is a string of up to 30 alphanumeric characters. The list is composed of one or more application tag values separated by commas.

The User, Group, Application, and Tag attributes can be an attribute value grouping.

When a process is created (fork), it remains in the same class as its parent. Reclassification happens when the new process issues a system call which can modify one of the attributes of the process used for classification; for example, *exec*, *setuid* (and related calls), *setgid* (and related calls), *setpri* and *plock*.

To classify the process, WLM examines the top-level **rules** file for the active configuration to determine which superclass the process belongs. For each rule in the file, WLM checks the current values for the process attributes against the values and lists of values specified in the rule. Rules are checked in the order that they appear in the file. When a match is found, the process is assigned to the superclass named in the first field of the rule. Then the rules file for the superclass is examined in the same way to determine to which subclass the process should be assigned.

For a process to match one of the rules, each of its attributes must match the corresponding field in the rule. The following is a list of the criteria used to determine whether the value of an attribute matches the values in the field of the **rules** file:

- If the field in the rules file has a value of hyphen (-), then any value of the corresponding process attribute is a match.
- For all the attributes except *type*, if the value of the process attribute matches one of the values in the list in the rules file that is not excluded (prefaced by a ″!″), then a match has occurred.
- For the *type* attribute, if one of the values in the rule is comprised of two or more values separated by a plus (+), then a process is a match only if its characteristics match all the values.

At both superclass and subclass levels, WLM goes through the rules in the order in which they appear in the **rules** file, and classifies the process in the class corresponding to the first rule for which the process is a match. The order of the rules in the rules file is therefore extremely important. Use caution when you create or modify the rules file.

Manual Assignment in WLM

A process or a group of processes can be manually assigned to a superclass and/or subclass by using Web-based System Manager, SMIT, or the **wlmassign** command. An application can assign processes through the **wlm_assign** API function.

To manually assign processes to a class or to cancel an existing manual assignment, a user must have the appropriate level of privilege. (See “Security Considerations” on page 92 for further details.) A manual assignment can be made or canceled separately at the superclass level, the subclass level, or both. This assignment is specified by flags for the programming interface and a set of options for the command line interface used by the WLM administration tools. So, a process can be manually assigned to a superclass only, a subclass only, or to a superclass and a subclass of that superclass. In the latter case, the dual assignment can be done simultaneously (with a single command or API call) or at different times, by different users.

Assignment is very flexible, but can be confusing. Following are two examples of the possible cases.

Example 1: First Assignment of Processes

A system administrator manually assigns *Process1* from *superclassA* to *superclassB* (superclass-level-only assignment). The automatic assignment rules for the subclasses of *superclassB* are used by WLM to determine to which subclass the process is ultimately assigned. *Process1* is assigned to *superclassB.subclassA* and is flagged as having a ″superclass only″ assignment.

A user with the appropriate privileges assigns *Process2* from its current class *superclassA.subclassA* to a new subclass of the same superclass, *superclassA.subclassB*. *Process2* is assigned to its new subclass and flagged as having a ″subclass only″ assignment.

A WLM administrator of the subclasses of *superclassB* manually reassigns *Process1* to *subclassC*, which is another subclass of *superclassB*. *Process1* is reclassified into *superclassB.subclassC* and is now flagged as having both superclass and subclass level assignment.

Example 2: Reassignment or Cancelation of Manual Assignment

The reassignment and cancelation of a manual assignment at the subclass level is less complex and affects only the subclass level assignment.

Suppose that the system administrator wants *Process2* to be in a superclass with more resources and decides to manually assign *Process2* to *superclassC*. In Example 1, *Process2* was manually assigned to *subclassB* of *superclassA*, with a ″subclass only″ assignment. Because *Process2* is assigned to a different superclass, the previous manual assignment becomes meaningless and is canceled. *Process2* now has a ″superclass only″ manual assignment to *superclassC*, and in the absence of inheritance, is assigned to a subclass of *superclassC* using the automatic assignment rules.

Now, the system administrator decides to terminate the manual assignment from *Process1* to *superclassB*. The ″superclass level″ manual assignment of *Process1* is canceled, and in the absence of inheritance, *Process1* is assigned a superclass using the top level automatic assignment rules.

If the rules have not changed, *Process1* is assigned to *superclassA*, and its subclass level manual assignment to *superclassB.subclassC* becomes meaningless and is canceled.

If for some reason the top level rules assign *Process1* to *superclassB*, then the subclass level assignment to *superclassB.subclassC* is still valid and remains in effect. *Process1* now has a ″subclass only″ manual assignment.

Updating WLM

When WLM is updated (with the **wlmcntrl -u** command), the updated configuration can load a new set of classification rules. When this happens, processes are often reclassified using the new rules. WLM does not reclassify processes that have been manually assigned or that are in a class with inheritance enabled, unless their class does not exist in the new configuration.

Security Considerations

To assign a process to a class or to cancel a prior manual assignment, the user must have authority both on the process and on the target class. These constraints translate into the following rules:

- The root user can assign any process to any class.
- A user with administration privileges on the subclasses of a given superclass (that is, the user or group name matches the user or group names specified in the attributes **adminuser** and **admingroup** of the superclass) can manually reassign any process from one of the subclasses of this superclass to another subclass of the superclass.
- Users can manually assign their own processes (ones associated with the same real or effective user ID) to a subclass for which they have manual assignment privileges (that is, the user or group name matches the user or group names specified in the attributes **authuser** and **authgroup** of the superclass or subclass).

To modify or terminate a manual assignment, users must have at least the same level of privilege as the person who issued the last manual assignment.

Managing Resources with WLM

WLM monitors and regulates the resource utilization, on a per-class basis, of the threads and processes active on the system. You can set minimum or maximum limits per class for each resource type managed by WLM, as well as a target value per class for each resource. This target is representative of the amount of the resource that is optimal for the jobs in the class.

The shares and limits at the superclass level refer to the total amount of each resource available on the system. At the subclass level, shares and limits refer to the amount of each resource made available to the superclass that the subclass is in (superclass target). The hierarchy of classes is a way to divide the system resources between groups of users (superclasses) and delegate the administration of this share of the resources to superclass administrators. Each superclass administrator can then redistribute this amount of resources between the users in the group by creating subclasses and defining resource entitlements for these subclasses.

Resource Types

WLM manages three types of resources on a percentage consumption basis:

CPU utilization of the threads in a class	This is the sum of all the CPU cycles consumed by every thread in the class.
Physical memory utilization for the processes in a class	This is the sum of all the memory pages which belong to the processes in the class.
Disk I/O bandwidth for the class	This is the bandwidth (in 512-byte blocks per second) of all the I/Os started by threads in the class on each disk device accessed by the class.

Every second, WLM calculates the per-class utilization for each resource during the last second, as a percentage of the total resource available, as follows:

- For the CPU, the total amount of CPU time available every second is equal to 1 second times the number of CPUs on the system. For instance, on an eight-way SMP, if all the threads of a class combined consumed 2 seconds of CPU time during the last second, this represents a percentage of 2/8 = 25%. The percentage used by WLM for regulation is a decayed average over a few seconds of this ″instantaneous″ per-second resource utilization.

- For physical memory, the total amount of physical memory available for processes at any given time is equal to the total number of memory pages physically present on the system minus the number of pinned pages. Pinned pages are not managed by WLM because these pages cannot be stolen from a class and given to another class to regulate memory utilization. The memory utilization of a class is the ratio of the number of non-pinned memory pages owned by all the processes in the class to the number of pages available on the system, expressed as a percentage.
- For disk I/O, the main problem is determining a meaningful available bandwidth for a device. When a disk is 100% busy, its throughput, in blocks per second, is very different if one application is doing sequential I/Os than if several applications are generating random I/Os. If you only used the maximum throughput measured for the sequential I/O case (as a value of the I/O bandwidth available for the device) to compute the percentage of utilization of the device under random I/Os, you might be misled to think that the device is at 20% utilization, when it is in fact at 100% utilization.

To get more accurate and reliable percentages of per-class disk utilization, WLM uses the statistics provided by the disk drivers (displayed using the **iostat** command) giving for each disk device the percentage of time that the device has been busy during the last second. WLM counts the number of total blocks which have been read or written on a device during the last second by all the classes accessing the device, and how many blocks have been read or written by each class and what was the percentage of utilization of the device. WLM then calculates the percentage of the disk throughput consumed by each class.

For instance, if the total number of blocks read or written during the last second was 1000 and the device had been 70% busy, this means that a class reading or writing 100 blocks used 7% of the disk bandwidth. Similarly to the CPU time (another renewable resource), the values used by WLM for its disk I/O regulation are also a decayed average over a few seconds of these per second percentages.

For the disk I/O resource, the shares and limits apply to each disk device individually accessed by the class. The regulation is done independently for each device. This means that a class could be over its entitlement on one device, and the I/Os to this device will be regulated while it is under its entitlement on another disk and the I/Os to this other device will not be constrained.

In AIX 5.2 and later, WLM supports accounting and regulation of resources on the basis of total consumption. There are two types of resources that can be regulated in this manner: class totals and process totals.

class totals
: Per-class limits can be specified for the number of processes, threads, and login sessions in the class. These limits are specified as the absolute number of each resource that can exist in the class at any instant. These limits are strictly enforced; when a class has reached its limit for one of these resources, any attempt to create another instance of the resource will fail. The operation will continue to fail for any process in the class until the class is below its specified limit for the resource.

process totals
: Per-process limits can be specified for the total amount of CPU time, blocks of disk I/O, and connect time for a login session. These limits are specified at the class level, but apply to each process in the class individually (each process can consume this amount). These consumption values are cumulative and thus represent the total amount of each particular resource that has been consumed by the process during its lifetime. Once a process has exceeded its total limit for any resource, the process will be terminated. The process will be sent a SIGTERM signal, and if it catches this signal and does not exit after a 5 second grace period, will be sent a SIGKILL signal. When a login session has reached 90% of its connect time limit, a warning message will be written to the controlling terminal to warn the user that the session will soon be terminated.

Target Shares

The target (or desired) resource consumption percentage for a class is determined by the number of shares it has for a particular resource. The shares represent how much of a particular resource a class should get, relative to the other classes in its tier. A class's target percentage for a particular resource is simply its number of shares divided by the number of active shares in its tier. If limits are also being used, the target is limited to the range [minimum, soft maximum]. If the calculated target outside this range, it is set to the appropriate upper/lower bound (see Resource Limits). The number of active shares is the total number of shares of all classes that have at least one active process in them. Since the number of active shares is dynamic, so is the target. If a class is the only active class in a tier, its target will be 100% of the amount of resource available to the tier.

For example, assume three active superclasses classes in tier 0—A, B, and C—with shares for a particular resource of 15, 10, and 5, respectively, the targets would be:

target(A) = 15/30 = 50%
target(B) = 10/30 = 33%
target(C) = 5/30 = 17%

If some time later, class B becomes inactive (no active processes), the targets for class A and C will be automatically adjusted:

target(A) = 15/20 = 75%
target(C) = 5/20 = 25%

As you can see, the shares represent a self-adapting percentage which allow the resources allocated to a class to be evenly distributed to or taken from the other classes when it becomes active/inactive.

To allow a high degree of flexibility, the number of shares for a class can be any number between 1 and 65535. Shares can be specified for superclasses and subclasses. For superclasses, the shares are relative to all other active superclasses in the same tier. For subclasses, the shares are relative to all other active subclasses in the same superclass, in the same tier. The shares for a subclass one superclass have no relationship to the shares for a subclass of another superclass.

In some cases, it might be desirable to make the target for a class independent from the number of active shares. To accomplish this, a value of ″-″ can be specified for the number of shares. In this case, the class will be unregulated for that resource, meaning that it has no shares, and its target is not dependent on the number of active shares. Its target will be set to (resource available to the tier - the sum of mins for all other classes in the tier). This target, or actual consumption (whichever is lower) is then taken out of what is available to other classes in the same tier.

For example, assume classes A, B, C, and D have shares for a particular resource of ″-″, 200, 150, and 100, respectively. All classes are active, and class A is consuming 50% of the resource:

target(A) = unregulated = 100%
target(B) = 200/450 * available = 44% * 50% = 22%
target(C) = 150/450 * available = 33% * 50% = 17%
target(D) = 100/450 * available = 22% * 50% = 11%

Because class A is unregulated and is consuming 50% of the available resource, the other classes only have 50% available to them, and their targets are calculated based on this percentage. Since class A will always be below its target (100%), it will always have a higher priority than all other classes in the same tier that are at or above their targets (see “Understanding Class Priority” on page 97 for more information).

Note: Making a class unregulated for a resource is not the same as putting it in a higher tier. The following behaviors, listed here, are true for an unregulated class (in the same tier), and are not true if the class is put in a higher tier:

- Since the shares are defined on a per-resource basis, a class can be unregulated for one or more resources, and regulated for others.
- The minimum limits for other classes in the same tier are honored. Higher tiers do not honor minimums specified in the lower tiers.
- Even in the absence of minimum limits for the classes with shares, the consumption of unregulated classes is somewhat dependent on classes with shares since they are competing for some of the resource available to the tier. Some experimentation is required to see what the behavior is with a given workload.

If the number of shares is unspecified, a default value of ″-″ will be used, and the class will be unregulated for that resource. Note that in the first version of WLM, the default share value, if unspecified, was 1.

The shares are specified per class for all the resource types. Shares are specified in stanzas of the **shares** file. For example:

```
shares
classname:
   CPU      =   2
   memory   =   4
   diskIO   =   3
```

Resource Limits

In addition to using shares to define relative resource entitlements, WLM provides the ability to specify resource limits for a class. Resource limits allow the administrator to have more control over resource allocation. These limits are specified as percentages and are relative to the amount of resource available to the tier that the class is in.

There are three type of limits for percentage-based regulation:

Minimum
This is the minimum amount of a resource that should be made available to the class. If the actual class consumption is below this value, the class will be given highest priority access to the resource. The possible values are 0 to 100, with 0 being the default (if unspecified).

Soft Maximum
This is the maximum amount of a resource that a class can consume when there is contention for that resource. If the class consumption exceeds this value, the class will be given the lowest priority priority in its tier. If there is no contention for the resource (from other classes in the same tier), the class will be allowed to consume as much as it wants. The possible values are 1 to 100, with 100 being the default (if unspecified).

Hard Maximum
This is the maximum amount of a resource that a class can consume, even when there is no contention. If the class reaches this limit, it will not be allowed to consume any more of the resource until its consumption percentage falls below the limit. The possible values are 1 to 100, with 100 being the default (if unspecified).

Resource limit values are specified in the resource limit file by resource type within stanzas for each class. The limits are specified as a minimum to soft maximum range, separated by a hyphen (-) with white space ignored. The hard maximum when specified follows the soft maximum, and is separated by a semicolon (;). Each limit value is followed by a percent sign (%).

The following are examples using the rules files:

- If user joe from group acct3 executes **/bin/vi**, then the process will be put in superclass acctg.

- If user sue from group dev executes **/bin/emacs**, then the process will be in the superclass devlt (group ID match), but will not be classified into the editors subclass, because user sue is excluded from that class. The process will go to devlt. Default.
- When a DB administrator starts **/usr/sbin/oracle** with a user ID of oracle and a group ID of dbm, to serve the data base DB1, the process will be classified in the Default superclass. Only when the process sets its tag to _DB1 will it be assigned to superclass db1.

Limits are specified for all resource types, per class, in stanzas of the **limits** file. For example:

```
shares
classname:
   CPU     =   0%-50%;80%
   memory  =   10%-30%;50%
```

In this example, no limits are set for disk I/O. Using the system defaults, this translates to the following:

```
diskIO   =   0%-100%;100%
```

All of the preceding examples assume that the superclasses and subclasses described do not have the inheritance attribute set to yes. Otherwise, the new processes would simply inherit the superclass or subclass from their parent.

The following are the only constraints that WLM places on resource limit values:

- The minimum limit must be less than or equal to the soft maximum limit.
- The soft maximum limit must be less than or equal to the hard maximum limit.
- The sum of the minimum of all the superclasses within a tier cannot exceed 100.
- The sum of the minimum of all the subclasses of a given superclass within a tier cannot exceed 100.

When a class with a hard memory limit has reached this limit and requests more pages, the VMM page replacement algorithm (LRU) is initiated and ″steals″ pages from the limited class, thereby lowering its number of pages below the hard maximum, before handing out new pages. This behavior is correct, but extra paging activity, which can take place even where there are plenty of free pages available, impacts the general performance of the system. Minimum memory limits for other classes are recommended before imposing a hard memory maximum for any class.

Since classes under their minimum have the highest priority in their tier, the sum of the minimums should be kept to a reasonable level, based on the resource requirements of the other classes in the same tier.

The constraint of the sum of the minimum limits within a tier being less than or equal to 100 means that a class in the highest priority tier is always allowed to get resources up to its minimum limit. WLM does not guarantee that the class will actually reach its minimum limit. This depends on how the processes in the class use their resources and on other limits that are in effect. For example, a class might not reach its minimum CPU entitlement because it cannot get enough memory.

For physical memory, setting a minimum memory limit provides some protection for the memory pages of the class's processes (at least for those in the highest priority tier). A class should not have pages stolen when it is below its minimum limit unless all the active classes are below their minimum limit and one of them requests more pages. A class in the highest tier should never have pages stolen when it is below its minimum. Setting a memory minimum limits for a class of interactive jobs helps make sure that their pages will not all have been stolen between consecutive activations (even when memory is tight) and improves response time.

Attention: Using hard maximum limits can have a significant impact on system or application performance if not used appropriately. Since imposing hard limits can result in unused system resources, in most cases, soft maximum limits are more appropriate. One use for hard maximum limits might be to limit the consumption of a higher tier to make some resource available to a lower tier, though putting applications that need resources in the higher tier may be a better solution.

Beginning with AIX 5.2, the total limits can be specified in the limits file with values and units summarized in the following table:

Table 2. Resource Limits for Workload Manager

Resource	Allowed Units	Default Unit	Maximum Value	Minimum Value
totalCPU	s, m, h, d, w	s	2^{30} – 1s	10 s
totalDiskIO	KB, MB, TB, PB, EB	KB	(2^{63} – 1) * 512/1024 KB	1 MB
totalConnectTime	s, m, h, d, w	s	2^{63} – 1 s	5 m
totalProcesses	–	–	2^{63} – 1	2
totalThreads	–	–	2^{63} – 1	2
totalLogins	–	–	2^{63} – 1	1

Note: The unit specifiers are not case sensitive. s = seconds, m = minutes, h = hours, d = days, w = weeks, KB = kilobytes, MK = megabytes, ... etc.

An example limits stanza follows:

```
BadUserClass:
     totalCPU = 1m
     totalConnectTime = 1h
```

The total limits can be specified using any value in the above table with the following restrictions:

- If specified, the value for totalThreads must be at least the value of totalProcesses.
- If totalThreads is specified and totalProcesses is not, the limit for totalProcesses will be set to the value of totalThreads.

The total limits can be specified at the superclass and subclass level. When checking the limits, the subclass limit is checked before the superclass limit. If both limits are specified, the lower of the two is enforced. If the specified subclass limit is greater than its associated superclass limit, a warning will be issued when the configuration is loaded, but it will be loaded. This is significant for the class total limits since the limit is absolute (not relative to the superclass) and one subclass could consume all resources available to the superclass. If unspecified, the default value for all total limits is ″-″ which means no limit. By default, class and process total accounting and regulation will be enabled when WLM is running. The **-T [class|proc]** option of the **wlmcntrl** command can be used to disable total accounting and regulation.

Understanding Class Priority

WLM allocates resources to classes by giving a priority to each class for each resource. The priority of a class is dynamic and is based on the tier, shares, limits, and the current consumption of the class. At any given time, the highest priority class or classes will be given preferential access to resources. At the highest level, tiers represent non-overlapping ranges of class priority. Classes in tier 0 will always be guaranteed to have a higher priority than classes in tier 1 (unless over hard maximum).

When determining class priority, WLM enforces its constraints with the following precedence (highest to lowest):

hard limits
: If class consumption exceeds the hard maximum for a resource, the class is given the lowest-possible priority for the resource and will be denied access until its consumption falls below this limit.

tier
: In the absence of hard limits, a class's priority will be bounded by the minimum and maximum allowed priority for its tier.

soft limits

If class consumption is below the minimum of the soft maximum limits for a resource, the class is given the highest-priority in the tier. If class consumption is above the soft maximum, the class is given the lowest-priority in the tier.

shares

These are used to calculated the class consumption targets for each resource. The class priority is increased as class consumption falls below the target, and is decreased as it rises above the target. Note that soft limits have higher precedence and the class's priority will be determined based on them, when applicable.

Even though both shares and limits can be used for each class and for each resource, results are more predictable if only one or the other is used per class.

Resource Sets

WLM uses resource sets (or *rsets*) to restrict the processes in a given class to a subset of the system's physical resources. In WLM, the physical resources managed are the memory and the processors. A valid resource set is composed of memory and at least one processor.

Using SMIT or Web-based System Manager, a system administrator can define and name resource sets containing a subset of the resources available on the system. Then, using the WLM administration interfaces, root user or a designated superclass administrator can use the name of the resource set as the **rset** attribute of a WLM class. From then on, every process assigned to this WLM class is dispatched only on one of the processors in the resource set, effectively separating workloads for the CPU resource.

All of the current systems have only one memory domain shared by all the resource sets, so this method does not physically separate workloads in memory.

Rset Registry

The **rset** registry services enable system administrators to define and name resource sets so that they can then be used by other users or applications. To alleviate the risks of name collisions, the registry supports a two-level naming scheme. The name of a resource set is in the form *name_space*/*rset_name*. Both the *namespace* and *rset_name* can each be 255 characters in length, are case-sensitive, and may contain only uppercase and lowercase letters, numbers, underscores, and periods (**.**). The *namespace* of **sys** is reserved by the operating system and used for **rset** definitions that represent the resources of the system.

The **rset** definition names are unique within the registry name space. Adding a new **rset** definition to the registry using the same name as an existing **rset** definition causes the existing definition to be replaced with the new definition, given the proper permission and privilege. Only root can create, modify, and delete resource sets and update the in-core **rset** database, using SMIT or Web-based System Manager.

Each **rset** definition has an owner (user ID), group (group ID), and access permissions associated with it. These are specified at the time the **rset** definition is created and exist for the purpose of access control. As is the case for files, separate access permissions exist for the owner, group and others that define whether read and/or write permission has been granted. Read permission allows an **rset** definition to be retrieved while write permission allows an rset definition to be modified or removed.

System Administrator defined **rset** definitions are kept in the **/etc/rsets** stanza file. The format of this file is not described, and users must manipulate **rsets** through the SMIT or Web-based System Manager interfaces to prevent future potential compatibility problems if the file format is modified. As is the case for WLM class definitions, the **rset** definitions must be loaded into kernel data structures before they can be used by WLM.

Rather than giving an extensive definition of resource sets and how to create them from basic building blocks called *system RADs* (Resource Access Domains). For an example that details what is in a resource set and how to create new one, see Creating a Resource Set in *AIX 5L Version 5.3 System Management Guide: Operating System and Devices*.

Setting Up WLM

Class definitions, class attributes, the shares and limits, and the automatic class assignment rules, can be entered using Web-based System Manager, SMIT, or the WLM command-line interface. These definitions and rules are kept in plain text files, that can also be created or modified using a text editor.

These files (called *WLM property files*) are kept in the subdirectories of **/etc/wlm**. A set of files describing superclasses and their associated subclasses define a WLM configuration. The files for the WLM **Config** configuration are in **/etc/wlm/Config**. This directory contains the definitions of the WLM parameters for the superclasses. The files are named **description**, **classes**, **shares**, **limits**, and **rules**. This directory might also contain subdirectories with the name of the superclasses where the subclass definitions are kept. For example, for the superclass *Super* of the WLM configuration **Config**, the directory **/etc/wlm/Config/Super** contains the property files for the subclasses of the superclass *Super*. The files are named **description**, **classes**, **shares**, **limits**, and **rules**.

After a WLM configuration has been defined by the system administrator, it can be made the active configuration using Web-based System Manager, the **smit wlmmanage** fast path, or the **wlmcntrl** command.

You can define multiple sets of property files, defining different configurations of workload management. These configurations are usually located in subdirectories of **/etc/wlm**. A symbolic link **/etc/wlm/current** points to the directory containing the current configuration files. This link is updated by the **wlmcntrl** command when WLM starts with a specified set of configuration files.

Instructions for creating a WLM configuration are described in Configure Workload Manager (WLM) to Consolidate Workloads in the *AIX 5L Version 5.3 System Management Guide: Operating System and Devices*. This section describes the concepts and requirements of each phase of the procedure.

Identifying Application Requirements

The first phase of defining a configuration requires an understanding of your users and their computing needs, as well as the applications on your system, their resource needs and the requirements of your business (for example, which tasks are critical and which can be given a lower priority). Based on this understanding, you define your superclasses and then your subclasses.

Setting priorities is dependent on what function WLM serves in your organization. In the case of server consolidation, you might already know the applications, the users and their resource requirements, and you might be able to skip or shorten some of the steps.

WLM allows you to classify processes by user or group, application, type, tag, or a combination of these attributes. Because WLM regulates the resource utilization among classes, system administrators should group applications and users with the same resource utilization patterns into the same classes. For instance, you might want to separate the interactive jobs that typically consume very little CPU time but require quick response time from batch-type jobs that typically are very CPU- and memory-intensive. This is the same in a database environment in which you need to separate the OLTP-type traffic from the heavy-duty queries of data mining.

This step is done using Web-based System Manager, SMIT, or command-line interface. For the first few times, it is probably a good idea to use Web-based System Manager or SMIT to take you through the steps of creating your first WLM configuration, including defining the superclasses and setting their

attributes. For the first pass, you can set up some of the attributes and leave the others at their default value. This is the same for the resource shares and limits. All these class characteristics can be dynamically modified at a later time.

You can then start WLM in passive mode, check your classification, and start reviewing the resource utilization patterns of your applications.

Verify your configuration, using the **wlmcheck** command or the corresponding SMIT or Web-based System Manager menus. Then start WLM in passive mode on the newly defined configuration. WLM will classify all the existing processes (and all processes created from that point on) and start compiling statistics on the CPU, memory, and disk I/O utilization of the various classes. WLM will not try to regulate this resource usage.

Verify that the various processes are classified in the appropriate class as expected by the system administrator (using the **-o** flag of the **ps** command). If some of the processes are not classified as you expect, you can modify your assignment rules or set the inheritance bit for some of the classes (if you want the new processes to remain in the same class as their parent) and update WLM. You can repeat the process until you are satisfied with this first level of classification (superclasses).

Running WLM in passive mode and refreshing WLM (always in passive mode) involves low risk, has low overhead operation, and can be done safely on a production system without disturbing normal system operation. To activate and refresh WLM, use the **wlmcntrl** command, invoked either from the command line or from SMIT or Web-based System Manager.

Run WLM in passive mode to gather statistics by using the **wlmstat** command. The **wlmstat** command can be used at regular time intervals to display the per-class resource utilization as a percentage of the total resource available, for superclasses). This allows you to monitor your system for extended periods of time to review the resource utilization of your main applications.

Defining Tiers, Shares, and Limits

Using the data gathered from running WLM in passive mode and your business goals, decide which tier number will be given to every superclass and what share of each resource should be given to the various classes. For some classes, you might want to define minimum or maximum limits. Adjust the shares and tier numbers to reach your resource-allocation goals. Reserve limits for cases that cannot be solved only with shares. Also, you might decide you need to add subclasses.

- Use minimum limits for applications that typically have low resource usage but need a quick response time when activated by an external event. One of the problems faced by interactive jobs in situations where memory becomes tight is that their pages get stolen during the periods of inactivity. A memory minimum limit can be used to protect some of the pages of interactive jobs if the class is in tier 0.
- Use maximum limits to contain some resource-intensive, low-priority jobs. Unless you partition your system resources for other reasons, a hard maximum will make sense mostly for a nonrenewable resource such as memory. This is because of the time it takes to write data out to the paging space if a higher priority class needs pages that the first class would have used. For CPU usage, you can use tiers or soft maximum to make sure that if a higher priority class is immediately assigned CPU time.

When creating and adjusting the parameters of subclasses, you can refresh WLM only for the subclasses of a given superclass that do not affect users and applications in the other superclasses, until you are satisfied with the system behavior.

You can also define other configurations with different parameters, according to the needs of the business. When doing so, you can save time by copying and modifying existing configurations.

Fine-Tuning the Configuration

You are now ready to start WLM in active mode. Monitor the system using the **wlmstat** command and verify that the regulation done by WLM aligns with your goals and does not unduly deprive some applications of resources while others get more than they should. If this is the case, adjust the shares and refresh WLM.

As you monitor and adjust the shares, limits, and tier numbers, decide whether you want to delegate the administration for the subclasses for some or all of the superclasses. The administrator can then monitor and set up the subclass shares, limits, and tier number.

The administrator of each superclass can repeat this process for the subclasses of each superclass. The only difference is that WLM cannot run in passive mode at the subclass level only. The subclass configuration and tuning has to be done with WLM in active mode. One way not to impact users and applications in the superclass is to start the tier number, and the shares and limits for the subclasses at their default value ('-' (hyphen) for shares, 0% for minimum, and 100% for soft and hard maximum). With these settings, WLM will not regulate the resource allocation between the subclasses.

WLM Application Programming Interface

Applications can use the WLM APIs, a set of routines in the **/usr/lib/libwlm.a** library, to perform all the tasks that a WLM administrator can perform using the WLM commands, including:

- Creating classes
- Changing classes
- Removing classes
- Manually assigning processes to specific classes
- Retrieving WLM statistics

In addition, the **wlm_set_tag** routine allows an application to set up an application tag and specify whether this tag should be inherited by child processes at **fork** or **exec**. The library provides support for multi-threaded 32-bit or 64-bit applications.

Application Tag

The application tag is a string of characters and is used as one of the classification criteria for the automatic classification of processes (using the **rules** file). This tag basically provides an application-defined classification criteria in addition to the system-defined criteria such as *user*, *group*, *application* and *type*.

When an application process sets its tag, it is immediately reclassified using the superclass and subclass rules in effect for the currently active WLM configuration. WLM then reviews the assignment rules looking for a match, using all the process attributes, including the new tag.

To be effective, this tag must appear in one or more of the assignment rules. The format and the use of the various tags by each application must be clearly specified in the application's administration documentation and well-known to the WLM administrators so that they can use the various values of the tags in their assignment rules to distinguish between different instances of the same application.

Because different users might have different requirements regarding what set of characteristics of the application processes they want to use to classify those processes, it is recommended that the application provide a set of configuration or run time attributes that can be used to build the tag. The application administrator can specify the format of this tag to the application. The attributes that can be used for the tag and the syntax to specify the format of the WLM tag are application-dependent and are the responsibility of the application provider.

For example, an instance of a database server is able to determine which database it is working on (*db_name*) and through which TCP port a given user is connected (*port_num*). Administrators might have any of the following priorities:

- To create different classes for processes accessing different databases to give each class different resource entitlement
- To separate the processes serving remote requests from different origins and to use the port number as a classification attribute
- To create one superclass for each database and subclass per port number in each superclass.

One way to accommodate these different needs is to specify the content and format of the tag. In this example, assume that the tag can be passed to the application in a configuration file or run time parameter such as `WLM_TAG=$db_name` or `WLM_TAG=$db_name_$port_num`.

When setting its tag, an application can specify whether this tag will be inherited by its children so that all the processes spawned by a specific instance of an application can be classified in the same class. Setting the tag inheritance is how the application tag is most commonly used.

The following is an example of how application tags could be used. In this example, the tag of the database is the same as the database name. Then two instances of the server working on two different databases would set up two different tags, for instance *db1* and *db2*.

A system administrator could create two different classes *dbserv1* and *dbserv2* and classify the two database servers (and all their children if tag inheritance is used) in these classes using the tags. It would then be possible to give each class different resource entitlement according to specific business goals.

The corresponding assignment rules could look similar to the following:

```
* class   resvd  user  group     application     type  tag
*
dbserv1     -    -     dbadm     /usr/sbin/dbserv  -   db1
dbserv2     -    -     dbadm     /usr/sbin/dbserv  -   db2
```

Class Management APIs

The WLM API provides applications with the ability to:

- Query the names and characteristics of the existing classes of a given WLM configuration (**wlm_read_classes**).
- Create a new class for a given WLM configuration, define the values of the various attributes of the class (such as tier and inheritance) and the shares and limits for the resources managed by WLM, such as CPU, physical memory, and block I/O (**wlm_create_class**).
- Change the characteristics of an existing class of a given WLM configuration, including the class attributes and resource shares and limits (**wlm_change_class**).
- Delete an existing class of a given configuration (**wlm_delete_class**).

The changes will be applied only to the property files of the specified WLM configuration. Optionally, by specifying an empty string as the configuration name, it is possible to apply the change only to the in-core classes, resulting in an immediate update of the active configuration state.

The API calls require from the caller the same level of privilege that would be required for the command line or for the SMIT or Web-based System Manager interfaces, as follows:

- Any user can read the class names and characteristics
- Only the root user can create, modify, or delete superclasses
- Only the root user or designated superclass administrators (superclass attributes *adminuser* or *admingroup*) can create, modify, or delete subclasses of a given superclass.

In cases where WLM administration is accomplished, both through the command line and administration tools by WLM administrators, and by application(s) through the API, some caution must be applied. Both interfaces share the same name space for the superclass and subclass names, and the total number of superclasses and subclasses.

In addition, when the API directly modifies the in-core WLM data (create new classes, for example), WLM administrators are not aware of this until classes that they did not create appear on the output of commands such as the **wlmstat** command. To avoid conflicts that would confuse the applications using this API when the system administrator updates WLM, the classes created through the API that are not defined in the WLM property files are not automatically removed from the in-core data. They remain in effect until explicitly removed through the **wlm_delete_class** routine or through an invocation of the **rmclass** command (invoked directly or through SMIT or Web-based System Manager by the system administrator).

The WLM API also provides applications with the ability to:

- Query or change the mode of operation of WLM using the **wlm_set** function
- Query the current status of WLM
- Stop WLM
- Toggle between active to passive mode
- Turn the **rset** binding on and off
- Start or update WLM with the current or an alternate configuration by using the **wlm_load** routine
- Assign a process or a group of processes to a class by using the **wlm_assign** routine.

The API requires the same levels of privilege as the corresponding **wlmcntrl** and **wlmassign** commands:

- Any user can query the state of WLM
- Only the root user can change the mode of operation of WLM
- Only the root user can update or refresh a whole configuration
- root or an authorized superclass administrator (*adminuser* or *admingroup*) can update WLM for the subclasses of a given superclass
- root, an authorized user (specified by *authuser* or **authgroup**), or an authorized superclass administrator (*adminuser* or *admingroup*) can assign processes to a superclass or subclass. See the **wlmassign** command for details.

WLM Statistics API

The WLM API routines and **wlm_get_bio_stats** provide application access to the WLM statistics displayed by the **wlmstat** commands.

WLM Classification API

The **wlm_check** routine allows the user to verify the class definitions and the assignment rules for a given WLM configuration. The API routine **wlm_classify** allows an application to determine to which class a process with a specified set of attributes would be classified.

Binary Compatibility

To provide binary compatibility if there are future changes to the data structures, each API call is passed a version number as a parameter, which will allow the library to determine with which version of the data structures the application has been built.

WLM Commands

WLM offers commands that allow system administrators to:

- Create, modify, and delete superclasses and subclasses, using the **mkclass**, **chclass**, and **rmclass** commands. These commands update the file's *classes*, *shares* and *limits*.
- Start, stop, and update WLM, using the **wlmcntrl** command.
- Check the WLM property files for a given configuration and determine to which class (superclass and subclass) a process with a given set of attributes is assigned using the **wlmcheck** command.
- Monitor the per-class resource utilization using the **wlmstat** (ASCII) command. Most of the performance tools, such as those started by the **svmon** and **topas** commands, have extensions to take into account the WLM classes and provide per-class and per-tier statistics using new command-line options.
- Flags in the **ps** command allow the user to display which class a process and its application tag is in. The **ps** command also allows the user to list all the processes belonging to a given superclass or subclass.
- There is no command line interface to manage the assignment rules. You must use the SMIT or Web-based System Manager administration tools, or a text editor.

Examples of WLM Classification, Rules, and Limits

Several methods exist for classifying a process, and all operate concurrently. A top-down strictest first-matching algorithm is used to provide for maximum configuration flexibility. You can organize process groupings by user with special cases for programs with certain names, by pathname with special cases for certain users, or any other arrangement.

Example of Assignment Rules

The following is an example of a top-level **rules** file for the configuration *Config* (**/etc/wlm/Config/rules** file):

```
* This file contains the rules used by WLM to
* assign a process to a superclass
*
* class resvd user    group   application     type  tag
db1      -      -       -    /usr/bin/oracle*      _DB1
db2      -      -       -    /usr/bin/oracle*      _DB2
devlt    -      -      dev        -            -    -
VPs      -   bob,ted    -         -            -    -
acctg    -      -     acct*      -            -    -
System   -    root      -         -            -    -
Default  -      -       -         -            -    -
```

The following is an example of the **rules** file for the **devlt** superclass in the **/etc/wlm/Config/devlt/rules** file:

```
* This file contains the rules used by WLM to
* assign a process to a subclass of the
* superclass devlt
*
* class  resvd   user    group  application         type       tag
hackers   -    jim,liz     -       -                 -          -
hogs      -       -        -       -             64bit+plock    -
editors   -     !sue       -   /bin/vi,/bin/emacs    -          -
build     -       -        -   /bin/make,/bin/cc     -          -
Default   -       -        -       -                 -          -
```

Note: The asterisk (*) is the comment character used in the **rules** file.

The following are examples using this rules file. The following examples assume that the superclasses and subclasses described do not have the inheritance attribute set to yes. If inheritance were enabled, new processes would inherit the superclass or subclass from their parent processes.

- If user joe from group acct3 executes **/bin/vi**, then the process will be put in superclass acctg.

- If user sue from group dev executes **/bin/emacs**, then the process will be put in the superclass devlt (group ID match), but will not be classified into the editors subclass, because the user is excluded from that class. The process will go to devlt by default.
- When a database administrator with a user ID of oracle and a group ID of dbm starts **/usr/sbin/oracle** to serve the DB1 database, the process will be classified in the Default superclass. Only when the process sets its tag to _DB1 will it be assigned to the superclass db1.

Example with Shares and Limits

Assume classes A, B, C, and D have shares of 3, 2, 1, and 1 respectively. If classes A, C, and D are active, the calculated targets would be:

target(A) = 3/5 = 60%
target(C) = 1/5 = 20%
target(D) = 1/5 = 20%

If during testing it was found that the applications in class A perform adequately when allowed to use 50% of the resource, it might be desirable to make the other 50% of the resource available to the other classes. This can be accomplished by giving class A a soft maximum of 50% for this resource. Since the current calculated target of 60% is over this limit, it will be adjusted down to the soft maximum value. When this happens, the target or actual consumption (whichever is lower) of class A is subtracted from the amount of resource available. Since this class now has a target constrained by its limit (and not its shares), the shares for the class are also subtracted from the number of active shares. Assuming class A has a current consumption of 48%, the targets will now be:

target(A) = 3/5 = 60%, softmax = 50, = 50%
target(C) = 1/2 * (100 - 48) = 26%
target(D) = 1/2 * (100 - 48) = 26%

Some time later, all classes may become active, and the targets will again be automatically adjusted:

target(A) = 3/7 = 42%
target(B) = 2/7 = 28%
target(C) = 1/7 = 14%
target(D) = 1/7 = 14%

Example with CPU Limits

The following example examines CPU allocation, assuming that each class can consume all the CPU that it is given.

Two classes, A and B, are in the same tier. CPU limits for A are [30% - 100%]. CPU limits for B are [20% - 100%]. When both classes are running and are using sufficient CPU, WLM first makes sure that they both get their minimum percentages of each second (averaged over several seconds). Then WLM distributes the remaining CPU cycles according to any CPU target share values.

If the CPU target shares for A and B are 60% and 40% respectively, then the CPU utilization for A and B stabilize at 60% and 40% respectively.

A third class, C, is added. This class is a group of CPU-bound jobs and should run with about half (or more) of the available CPU. Class C has limits of [20% - 100%] and CPU target shares of 100%. If C is in the same tier as A and B, then when C is starting, A and B see their CPU allocation decrease steeply and the three classes stabilize at 30%, 20% and 50%, respectively. Their targets in this case are also the minimum for A and B.

A system administrator might not want batch jobs to consume 50% of the CPU when other jobs, possibly of higher priority, are also running. In a situation like the previous example, C is placed in a lower priority

tier. C then receives whatever CPU remains after A and B receive their needs. In the above example, C receives no CPU time, because A and B are each capable of absorbing 100% of the CPU. In most situations, however, A and B, in a high-priority tier, are composed of interactive or transaction-oriented jobs, which do not use all of the CPU all of the time. C then receives some share of the CPU, for which it competes with other classes in the same or lower tiers.

Example with Memory Limits

The following example examines memory allocation to groups of processes with varying memory targets. Three groups of processes must run: a group of interactive processes that need to run whenever they are used (PEOPLE), a batch job that always runs in the background (BATCH1), and a second, more important batch job, that runs every night (BATCH0).

PEOPLE has a specified memory minimum of 20%, a memory target of 50 shares, and a class tier value of 1. A 20% minimum limit ensures that the desktop applications in this class resume fairly quickly when users touch their keyboards.

BATCH1 has a memory minimum of 50%, a memory target of 50 shares, and a tier value of 3.

BATCH0 has a memory minimum of 80%, a memory target of 50 shares, and a tier value of 2.

Classes PEOPLE and BATCH1 have a total memory minimum limit of 70. Under normal operation (when BATCH0 is not running), both of these classes are allowed to get all of their reserved memory. They share the rest of the memory in the machine about half and half, even though they are in different tiers. At midnight when BATCH0 is started, the memory minimum total reaches 150. WLM ignores the minimum requirements for the lowest tiers until processes in the upper tiers exit. BATCH0 takes memory from the BATCH1 50% reserve, but not from the PEOPLE 20% reserve. After BATCH0 is done, memory reserves for tier 3 processes are honored again and the system returns to its normal memory balance.

Backward Compatibility

When starting WLM with configurations created with versions earlier than AIX 5.1, only superclasses are used. The default output of the **wlmstat** command lists only the superclasses and is similar to those of previous versions. For example:

```
# wlmstat
               CLASS  CPU  MEM  DKIO
        Unclassified    0    0   0
           Unmanaged    0    0   0
             Default    0    0   0
              Shared    0    2   0
              System    2   12   0
              class1    0    0   0
              class2    0    0   0
#
```

If some of the superclasses have subclasses defined by a WLM administrator, then the subclasses are listed. For example:

```
# wlmstat
               CLASS  CPU  MEM  DKIO
        Unclassified    0    0   0
           Unmanaged    0    0   0
             Default    0    0   0
              Shared    0    2   0
              System    3   11   7
              class1   46    0   0
    class1.Default     28    0   0
     class1.Shared      0    0   0
        class1.sub1    18    0   0
              class2   48    0   0
#
```

The output is the same when you use the **ps** command. For processes in a superclass without any subclasses, the **ps** command lists the superclass name as the process class name.

Exclusive Use Processor Resource Sets

Exclusive use processor resource sets (XRSETs) allow administrators to guarantee resources for important work. An XRSET is a named resource set that changes the behavior of all the CPUs that it includes. Once a CPU is exclusive, it only runs programs explicitly directed to it.

Creating an XRSET

You must be a root user to create an XRSET. Use the **mkrset** command to create a resource set in the **sysxrset** namespace. For example, the command **mkrset -c 1-3 sysxrset/set1** creates an XRSET for CPUs 1, 2, and 3. The **rs_registername()** subroutine can also be used to create an XRSET.

Determining if XRSETs have been Defined on a System

The **lsrset -v -n sysxrset** command displays all XRSETs defined on a system. (There is currently no programming API to do this.)

Deleting an XRSET

You must be a root user to delete an XRSET. The **rmrset** command deletes an XRSET. The **rs_discardname()** subroutine may also be used to delete an XRSET.

Rebooting the System

When you reboot the system, any XRSETs that were set are removed from the registry and are no longer in effect.

Specifying Work for XRSETs

There are multiple ways for work to be marked as eligible to use exclusive use processors. The **attachrset** and **execrset** commands can be used to specify resource sets that contain exclusive use processors. Resource sets containing exclusive use processors can be associated with WLM classes. Work classified into such WLM classes will use the exclusive use processors specified in the resource set.

Using XRSETs with Bindprocessor and _system_configuration.ncpus

You cannot use **bindprocessor** to cause work to run on exclusive use processors. Only resource set-based attachments can cause work to run on exclusive use processors.

The number of CPUs in the system configuration (the **_system_configuration.ncpus** field) is not changed when XRSETs are created. There are still NCPUs in the system.

When programs use the **bindprocessor** system call to NCPUs, CPUs in XRSETs will fail with the **EINVAL** error. You can bind to any ID returned by the query option of the **bindprocessor** command. The query option (**bindprocessor -q**) will only return valid bind IDs, excluding those that are associated with exclusive CPUs.

For example, if there are 10 CPUs online in a system and three of them are in XRSETs, a **bindprocessor** to CPUs with bind IDs in the range of 0 to 6 will succeed. A **bindprocessor** to CPUs with bind IDs in the range of 7 to 9 will receive an **EINVAL** error.

Using XRSETs with Dynamic CPU Reconfiguration Operations

In general, dynamic CPU reconfiguration is not affected by exclusive use processors. However, the creation of XRSETs and assignment of work to those processors may prevent removal of a CPU. CPUs

that are dynamically added to the system may enter the system as either general use or exclusive use processors. They will enter the system as exclusive use if there is an XRSET containing the logical CPU ID when it enters the system.

Chapter 10. System Resource Controller and Subsystems

This chapter provides an overview of the System Resource Controller (SRC) and the various subsystems it controls. For task procedures, see System Resource Controller and Subsystemsin *AIX 5L Version 5.3 System Management Guide: Operating System and Devices*.

System Resource Controller Overview

The System Resource Controller (SRC) provides a set of commands and subroutines to make it easier for the system manager and programmer to create and control subsystems. A *subsystem* is any program or process or set of programs or processes that is usually capable of operating independently or with a controlling system. A subsystem is designed as a unit to provide a designated function.

The SRC was designed to minimize the need for operator intervention. It provides a mechanism to control subsystem processes using a common command line and the C interface. This mechanism includes the following:

- Consistent user interface for start, stop, and status inquiries
- Logging of the abnormal termination of subsystems
- Notification program called at the abnormal system termination of related processes
- Tracing of a subsystem, a group of subsystems, or a subserver
- Support for control of operations on a remote system
- Refreshing of a subsystem (such as after a configuration data change).

The SRC is useful if you want a common way to start, stop, and collect status information on processes.

Subsystem Components

A subsystem can have one or more of the following properties:

- Is known to the system by name
- Requires a more complex execution environment than a subroutine or nonprivileged program
- Includes application programs and libraries as well as subsystem code
- Controls resources that can be started and stopped by name
- Requires notification if a related process is unsuccessful to perform cleanup or to recover resources
- Requires more operational control than a simple daemon process
- Needs to be controlled by a remote operator
- Implements subservers to manage specific resources
- Does not put itself in the background.

A few subsystem examples are ypserv, ntsd, qdaemon, inetd, syslogd, and sendmail.

Note: See each specific subsystem for details of its SRC capabilities.

Use the **lssrc -a** command to list active and inactive subsystems on your system.

Subsystem Group

A *subsystem group* is a group of any specified subsystems. Grouping subsystems together allows the control of several subsystems at one time. A few subsystem group examples are TCP/IP, SNA Services, Network Information System (NIS), and Network File Systems (NFS).

Subserver

A *subserver* is a program or process that belongs to a subsystem. A subsystem can have multiple subservers and is responsible for starting, stopping, and providing status of subservers. Subservers can be defined only for a subsystem with a communication type of IPC message queues and sockets. Subsystems using signal communications do not support subservers.

Subservers are started when their parent subsystems are started. If you try to start a subserver and its parent subsystem is not active, the **startsrc** command starts the subsystem as well.

SRC Hierarchy

The System Resource Controller hierarchy begins with the operating system followed by a subsystem group (such as **tcpip**), which contains a subsystem (such as the **inetd** daemon), which in turn can own several subservers (such as the **ftp** daemon and the **finger** command).

List of SRC Administration Commands

The following is a list of SRC Administration commands:

srcmstr daemon	Starts the System Resource Controller
startsrc command	Starts a subsystem, subsystem group, or subserver
stopsrc command	Stops a subsystem, subsystem group, or subserver
refresh command	Refreshes a subsystem
traceson command	Turns on tracing of a subsystem, a group of subsystems, or a subserver
tracesoff command	Turns off tracing of a subsystem, a group of subsystems, or a subserver
lssrc command	Gets status on a subsystem.

Chapter 11. System Accounting

This chapter provides an overview of the system accounting utility, which allows you to collect and report on individual and group use of various system resources.

Note: A new Advanced Accounting subsystem is available beginning with AIX 5.3. For more information, see *AIX 5L Version 5.3 Understanding the Advanced Accounting subsystem*

The topics covered are:

Accounting Overview

This accounting information can be used to bill users for the system resources they utilize, and to monitor selected aspects of the system operation. To assist with billing, the accounting system provides the resource-usage totals defined by members of the adm group, and, if the **chargefee** command is included, factors in the billing fee.

The accounting system also provides data to assess the adequacy of current resource assignments, set resource limits and quotas, forecast future needs, and order supplies for printers and other devices.

The following information should help you understand how to implement the accounting utility in your system:

Collecting and Reporting System Data

For data to be collected automatically, a member of the adm group needs to follow the procedures described in ″Setting Up an Accounting System″. These procedures enable the **cron** daemon to run the commands that generate data on:

- The amount of time each user spends logged in to the system
- Usage of the processing unit, memory, and I/O resources
- The amount of disk space occupied by each user's files
- Usage of printers and plotters
- The number of times a specific command is given.

The system writes a record of each session and process after they are completed. These records are converted into total accounting (tacct) records arranged by user and merged into a daily report. Periodically, the daily reports are combined to produce totals for the defined fiscal period. Methods for collecting and reporting the data and the various accounting commands and files are discussed in the following sections.

Although most of the accounting data is collected and processed automatically, a member of the adm group can enter certain commands from the keyboard to obtain specific information. These commands are discussed in “Keyboard Commands” on page 116.

Collecting Accounting Data

There are several types of accounting data:

- “Connect-Time Accounting”
- “Process Accounting”
- “Disk-Usage Accounting”
- “Printer-Usage Accounting” on page 113
- “Fee Accounting” on page 113

They are described in the following sections.

Connect-Time Accounting

Connect-time data is collected by the **init** command and the **login** command. When you log in, the **login** program writes a record in the **/etc/utmp** file. 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 into the various display stations. If the **/var/adm/wtmp** connect-time accounting file exists, the **login** command adds a copy of this login record to it.

When your login program ends (normally when you log out), the **init** command records the end of the session by writing another record in the **/var/adm/wtmp** file. Logout records differ from login records in that they have a blank user name. Both the login and logout records have the form described in the **utmp.h** file.

The **acctwtmp** command also writes special entries in the **/var/adm/wtmp** file concerning system shutdowns and startups.

For more information, see “Connect-Time Reports” on page 113.

Process Accounting

The system collects data on resource usage for each process as it runs, including:

- The user and group numbers under which the process runs
- The first eight characters of the name of the command
- A 64-bit numeric key representing the Workload Manager class that the process belongs to
- The elapsed time and processor time used by the process
- Memory use
- The number of characters transferred
- The number of disk blocks read or written on behalf of the process

The **accton** command records these data in a specified file, usually the **/var/adm/pacct** file.

Related commands are the **startup** command, the **shutacct** command, the **dodisk** command, the **ckpacct** command, and the **turnacct** command.

For more information, see “Reporting Accounting Data” on page 113.

Disk-Usage Accounting

Much accounting information is collected as the resources are consumed. The **dodisk** command, run as specified by the **cron** daemon, periodically writes disk-usage records for each user to the **/var/adm/acct/nite(x)/dacct** file. To accomplish this, the **dodisk** command calls other commands. Depending upon the thoroughness of the accounting search, the **diskusg** command or the **acctdusg**

command can be used to collect data. The **acctdisk** command is used to write a total accounting record. The total accounting record, in turn, is used by the **acctmerg** command to prepare the daily accounting report.

The **dodisk** command charges a user for the links to files found in the user's login directory and evenly divides the charge for each file between the links. This distributes the cost of using a file over all who use it and removes the charges from users when they relinquish access to a file.

For more information, see "Disk-Usage Accounting Report" on page 114.

Printer-Usage Accounting

The collection of printer-usage data is a cooperative effort between the **enq** command and the queuing daemon. The **enq** command enqueues the user name, job number, and the name of the file to be printed. After the file is printed, the **qdaemon** command writes an ASCII record to a file, usually the **/var/adm/qacct** file, containing the user name, user number, and the number of pages printed. You can sort these records and convert them to total accounting records.

For more information, see "Printer-Usage Accounting Report" on page 114.

Fee Accounting

You can enter the **chargefee** command to produce an ASCII total accounting record in the **/var/adm/fee** file. This file will be added to daily reports by the **acctmerg** command.

For more information, see "Fee Accounting Report" on page 114.

Reporting Accounting Data

After the various types of accounting data are collected, the records are processed and converted into reports.

Accounting commands automatically convert records into scientific notation when numbers become large. A number is represented in scientific notation in the following format:

*Base***e+***Exp*

*Base***e-***Exp*

which is the number equal to the *Base* number multiplied by 10 to the **+***Exp* or **-***Exp* power. For example, the scientific notation 1.345e+9 is equal to 1.345×10^{9}, or 1,345,000,000. And the scientific notation 1.345e-9 is equal to 1.345×10^{-9} or, 0.000000001345.

Connect-Time Reports

The **runacct** command calls two commands, **acctcon1** and **acctcon2**, to process the login, logout, and system-shutdown records that collect in the**/var/adm/wtmp** file. The **acctcon1** command converts these records into session records and writes them to the **/var/adm/acct/nite(x)/lineuse** file. The **acctcon2** command then converts the session records into a total accounting record, **/var/adm/logacct**, that the **acctmerg** command adds to daily reports.

If you run the **acctcon1** command from the command line, you must include the **-l** flag to produce the line-use report, **/var/adm/acct/nite(x)/lineuse**. To produce an overall session report for the accounting period, **/var/adm/acct/nite(x)/reboots**, use the **acctcon1** command with the **-o** flag.

The **lastlogin** command produces a report that gives the last date on which each user logged in.

Process Accounting Reports

Two commands process the billing-related data that was collected in the **/var/adm/pacct** or other specified file. The **acctprc1** command translates the user ID into a user name and writes ASCII records containing

the chargeable items (prime and non-prime CPU time, mean memory size, and I/O data). The **acctprc2** command transforms these records into total accounting records that are added to daily reports by the **acctmerg** command.

Process accounting data also provides information that you can use to monitor system resource usage. The **acctcms** command summarizes resource use by command name. This provides information on how many times each command was run, how much processor time and memory was used, and how intensely the resources were used (also known as the *hog factor*). The **acctcms** command produces long-term statistics on system utilization, providing information on total system usage and the frequency with which commands are used.

The **acctcom** command handles the same data as the **acctcms** command, but provides detailed information about each process. 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 when the process ended, the name of the command, the user or group that invoked the process, the name of the WLM class the proces belonged to, and the port at which the process ran. Unlike other accounting commands, **acctcom** can be run by all users.

Disk-Usage Accounting Report

The disk-usage records collected in the **/var/adm/acct/nite(x)/dacct** file are merged into the daily accounting reports by the **acctmerg** command.

Printer-Usage Accounting Report

The ASCII record in the **/var/adm/qacct** file can be converted to a total accounting record to be added to the daily report by the **acctmerg** command.

Fee Accounting Report

If you used the **chargefee** command to charge users for services such as file restores, consulting, or materials, an ASCII total accounting record is written in the **/var/adm/fee** file. This file is added to the daily reports by the **acctmerg** command.

Daily Reports

Raw accounting data on connect-time, processes, disk usage, printer usage, and fees to charge are merged into daily reports by the **acctmerg** command. Called by the **runacct** command as part of its daily operation, the **acctmerg** command produces the following:

/var/adm/acct/nite(x)/dacct
: An intermediate report that is produced when one of the input files is full.

/var/adm/acct/sum(x)/tacct
: A cumulative total report in **tacct** format. This file is used by the **monacct** command to produce the ASCII monthly summary.

The **acctmerg** command can convert records between ASCII and binary formats and merge records from different sources into a single record for each user.

Monthly Report

Called by the **cron** daemon, the **monacct** command produces the following:

/var/adm/acct/fiscal	A periodic summary report produced from the **/var/adm/acct/sum/tacct** report by the **monacct** command. The **monacct** command can be configured to run monthly or at the end of a fiscal period.

Greater than eight character username support

In order to maintain backwards compatibility with all scripts long username support is not enabled by default within accounting. Instead all user ids are truncated to the first eight characters. In order to enable long username support most commands have been given the additional -X flag which allows them to

accept and output greater than eight character user ids (in both ASCII and Binary formats). In addition when long username support is enabled commands and scripts will process files in the **/var/adm/acct/sumx**, **/var/adm/acct/nitex**, and **/var/adm/acct/fiscalx** instead of using **/var/adm/acct/sum**, **/var/adm/acct/nite**, and **/var/adm/acct/fiscal**.

Accounting Commands

The accounting commands function several different ways. Some commands:

- Collect data or produce reports for a specific type of accounting: connect-time, process, disk usage, printer usage, or command usage.
- Call other commands. For example, the **runacct** command, which is usually run automatically by the **cron** daemon, calls many of the commands that collect and process accounting data and prepare reports. To obtain automatic accounting, you must first configure the **cron** daemon to run the **runacct** command. See the **crontab** command for more information about how to configure the **cron** daemon to submit commands at regularly scheduled intervals.
- Perform maintenance functions and ensure the integrity of active data files.
- Enable members of the adm group to perform occasional tasks, such as displaying specific records, by entering a command at the keyboard.
- Enable a user to display specific information. There is only one user command, the **acctcom** command, which displays process accounting summaries.

Commands That Run Automatically

Several commands usually run by the **cron** daemon automatically collect accounting data.

runacct
: Handles the main daily accounting procedure. Normally initiated by the **cron** daemon during non-prime hours, the **runacct** command calls several other accounting commands to process the active data files and produce command and resource usage summaries, sorted by user name. It also calls the **acctmerg** command to produce daily summary report files, and the **ckpacct** command to maintain the integrity of the active data files.

ckpacct
: Handles **pacct** file size. It is advantageous to have several smaller **pacct** files if you must restart the **runacct** procedure after a failure in processing these records. The **ckpacct** command checks the size of the **/var/adm/pacct** active data file, and if the file is larger than 500 blocks, the command invokes the **turnacct switch** command to turn off process accounting temporarily. The data is transferred to a new **pacct** file, **/var/adm/pacct** *x*. (*x* is an integer that increases each time a new **pacct** file is created.) If the number of free disk blocks falls below 500, the **ckpacct** command calls the **turnacct off** command to turn off process accounting.

dodisk
: Calls the **acctdisk** command and either the **diskusg** command or the **acctdusg** command to write disk-usage records to the **/var/adm/acct/nite/dacct** file. This data is later merged into the daily reports.

dodisk
: Calls the **acctdisk** command and either the **diskusg** command or the **acctdusg** command to write disk-usage records to the **/var/adm/acct/nite/dacct** file. This data is later merged into the daily reports.

monacct
: Produces a periodic summary from daily reports.

sa1 Collects and stores binary data in the **/var/adm/sa/sa** *dd* file, where *dd* is the day of the month.

sa2 Writes a daily report in the **/var/adm/sa/sa***dd* file, where *dd* is the day of the month. The command removes reports from the **/var/adm/sa/sa***dd* file that have been there longer than one week.

Other commands are run automatically by procedures other than the **cron** daemon:

startup
When added to the **/etc/rc** file, the **startup** command initiates startup procedures for the accounting system.

shutacct
Records the time accounting was turned off by calling the **acctwtmp** command to write a line to **/var/adm/wtmp** file. It then calls the **turnacct off** command to turn off process accounting.

Keyboard Commands

A member of the adm group can enter the following commands from the keyboard:

ac Prints connect-time records. This command is provided for compatibility with Berkeley Software Distribution (BSD) systems.

acctcom
Displays process accounting summaries. This command is also available to users.

acctcon1
Displays connect-time summaries. Either the **-l** flag or the **-o** flag must be used.

accton
Turns process accounting on and off.

chargefee
Charges the user a predetermined fee for units of work performed. The charges are added to the daily report by the **acctmerg** command.

fwtmp Converts files between binary and ASCII formats.

last Displays information about previous logins. This command is provided for compatibility with BSD systems.

lastcomm
Displays information about the last commands that were executed. This command is provided for compatibility with BSD systems.

lastlogin
Displays the time each user last logged in.

pac Prepares printer/plotter accounting records. This command is provided for compatibility with Berkeley Software Distribution (BSD) systems.

prctmp
Displays a session record.

prtacct
Displays total accounting files.

sa Summarizes raw accounting information to help manage large volumes of accounting information. This command is provided for compatibility with BSD systems.

sadc Reports on various local system actions, such as buffer usage, disk and tape I/O activity, TTY device activity counters, and file access counters.

sar Writes to standard output the contents of selected cumulative activity counters in the operating system. The **sar** command reports only on local activities.

time Prints real time, user time, and system time required to run a command.

timex Reports in seconds the elapsed time, user time, and run time.

Accounting Files

The two main accounting directories are the **/usr/sbin/acct** directory, where all the C language programs and shell procedures needed to run the accounting system are stored, and the **/var/adm** directory, which contains the data, report and summary files.

The accounting data files belong to members of the adm group, and all active data files (such as **wtmp** and **pacct**) reside in the adm home directory **/var/adm**.

Data Files

Files in the **/var/adm** directory are:

/var/adm/diskdiag	Diagnostic output during the running of disk accounting programs
/var/adm/dtmp	Output from the **acctdusg** command
/var/adm/fee	Output from the **chargefee** command, in ASCII **tacct** records
/var/adm/pacct	Active process accounting file
/var/adm/wtmp	Active process accounting file
/var/adm/Spacct *.mmdd*	Process accounting files for *mmdd* during the execution of the **runacct** command.

Report and Summary Files

Report and summary files reside in a **/var/adm/acct** subdirectory. You must create the following subdirectories before the accounting system is enabled. See ″Setting Up an Accounting System″ for more information.

/var/adm/acct/nite(x)
Contains files that the **runacct** command reuses daily

/var/adm/acct/sum(x)
Contains the cumulative summary files that the **runacct** command updates daily

/var/adm/acct/fiscal(x)
Contains the monthly summary files that the **monacct** command creates.

runacct Command Files

The following report and summary files, produced by the **runacct** command, are of particular interest:

/var/adm/acct/nite(x)/lineuse	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.
/var/adm/acct/nite(x)/daytacct	Contains the total accounting file for the previous day.
/var/adm/acct/sum(x)/tacct	Contains the accumulation of each day's **nite/daytacct** file and can be used for billing purposes. The **monacct** command restarts the file each month or fiscal period.
/var/adm/acct/sum(x)/cms	Contains the accumulation of each day's command summaries. The **monacct** command reads this binary version of the file and purges it. The ASCII version is **nite/cms**.
/var/adm/acct/sum(x)/daycms	Contains the daily command summary. An ASCII version is stored in **nite/daycms**.
/var/adm/acct/sum(x)/loginlog	Contains a record of the last time each user ID was used.
/var/adm/acct/sum(x)/rprt *mmdd*	This file contains a copy of the daily report saved by the **runacct** command.

Files in the /var/adm/acct/nite(x) Directory

active	Used by the **runacct** command to record progress and print warning and error messages. The file **active.** *mmdd* is a copy of the **active** file made by the **runacct** program 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 the **dodisk** command.

accterr	Diagnostic output produced during the execution of the **runacct** command.
lastdate	Last day the **runacct** executed, in **date +%m%d** format.
lock1	Used to control serial use of the **runacct** command.
lineuse	tty line usage report used by the **prdaily** command.
log	Diagnostic output from the **acctcon1** command.
log*mmdd*	Same as **log** after the **runacct** command 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 the **runacct** command.
tmpwtmp	**wtmp** file corrected by the **wtmpfix** command.
wtmperror	Contains **wtmpfix** error messages.
wtmperr*mmdd*	Same as **wtmperror** after the **runacct** command detects an error.
wtmp.*mmdd*	Contains previous day's **wtmp** file. Removed during the cleanup of **runacct** command.

Files in the /var/adm/acct/sum(x) Directory

cms	Total command summary file for the current fiscal period, in binary format.
cmsprev	Command summary file without the latest update.
daycms	Command summary file for the previous day, in binary format.
lastlogin	File created by the **lastlogin** command.
pacct.*mmdd*	Concatenated version of all **pacct** files for *mmdd*. This file is removed after system startup by the **remove** command.
rprt*mmdd*	Saved output of the **prdaily** command.
tacct	Cumulative total accounting file for the current fiscal period.
tacctprev	Same as **tacct** without the latest update.
tacct*mmdd*	Total accounting file for *mmdd*.

Files in the /var/adm/acct/fiscal(x) Directory

cms*?*	Total command summary file for the fiscal period, specified by *?*, in binary format
fiscrpt*?*	A report similar to that of the **prdaily** command for fiscal period, specified by *?*, in binary format
tacct*?*	Total accounting file for fiscal period, specified by *?*, in binary format.

Accounting File Formats

Accounting file output and formats are summarized in the following.

wtmp	Produces the active process accounting file. The format of the **wtmp** file is defined in the **utmp.h** file.
ctmp	Produces connect session records. The format is described in the **ctmp.h** file.
pacct*	Produces active process accounting records. The format of the output is defined in the **/usr/include/sys/acct.h** file.
Spacct*	Produces process accounting files for *mmdd* during the running of the **runacct** command. The format of these files is defined in the **sys/acct.h** file.
daytacct	Produces total accounting records for one day. The format of the file is defined in the **tacct** file format.
sum/tacct	Produces binary file that accumulates each day's command summaries. The format of this file is defined in the **/usr/include/sys/acct.h** header file.
ptacct	Produces concatenated versions of **pacct** files. The format of these files are defined in the **tacct** file.
ctacct	Produces connect total accounting records. The output of this file is defined in the **tacct** file.
cms	Produces total accounting command summary used by the **prdaily** command, in binary format. The ASCII version is **nite/cms**.
daycms	Daily command summary used by the **prdaily** command, in binary format. The ASCII version is **nite/daycms**.

Chapter 12. Web-based System Manager

Web-based System Manager is a client-server application that gives the user a powerful graphical user interface to access and manage multiple hosts. Through Web-based System Manager, you can view users and groups, install software, and printers and devices; manage logical volumes, users and groups, and resources; mount and unmount file systems; configuring the network; and do many other system administration tasks. A plug-in architecture makes it easier to extend the suite. In addition, Web-based System Manager supports dynamic monitoring and administrator notification of system events.

The interface provides point-and-click control of objects, which provides an alternative to learning and using commands or SMIT. You can select a machine to manage, and as you navigate through your desired operation, the panels refresh to display your currently allowable choices.

For a detailed explanation and procedures for using Web-based System Manager, *AIX 5L Version 5.3 Web-based System Manager Administration Guide*.

Chapter 13. System Management Interface Tool

Although Chapter 12, “Web-based System Manager,” on page 119 is the primary interface for system management, the System Management Interface Tool (SMIT) provides an alternative, natural-language, task-oriented interface. The SMIT facility runs in two interfaces, ASCII (nongraphical) or AIXwindows (graphical).

SMIT steps you through the desired task with the use of menus, selectors, and dialogs, thereby freeing you from the details of complex command syntax, valid parameter values, and system command spelling. In addition, SMIT creates log files that you can use to duplicate system configuration or to learn how to use specific commands.

In the SMIT interface, main menu selections lead to submenus, helping to narrow the scope of choice to a particular task. To skip the main menu and directly access a submenu or dialog, you can use the **smit** command with a *Fast Path* parameter.

To learn more about SMIT, you can:

- Start SMIT, then select **Using SMIT (Information Only)** from the SMIT Main Menu.
- In the SMIT dialogs, select **On Context (Ctrl+F1)** from the Help menu and move the cursor over the particular menu item or field about which you want more information.

The following table lists some basic SMIT tasks:

Basic SMIT Tasks

Task	*SMIT Fast Path*	*Selection (ASCII)*	*Selection (AIXwindows)*
Enter SMIT	**smit**		
Exit SMIT		F12	F12 or Exit SMIT option from Exit menu
Show command		F6	F6 or Command option from Show menu
Show fast path		F8	F8 or FastPath option from Show menu

Chapter 14. AIX Documentation

IBM AIX Information Center

The AIX Information Center provides navigation and search capabilities for system documentation. For instructions on how to use the information center, select **Information Center Help** from the navigation bar. For instructions on installing and configuring the information center on AIX, see the *AIX 5L Version 5.3 Installation Guide and Reference*.

Chapter 15. Devices

Devices include hardware components such as, printers, drives, adapters, buses, and enclosures, as well as pseudo-devices, such as the error special file and null special file. Device drivers are located in the **/usr/lib/drivers** directory.

This chapter provides an overview of the following methods used by the operating system to manage these devices:

Configuring a Large Number of Devices

The number of devices that AIX can support can vary from system to system, depending on several important factors. The following factors have an impact on the file systems that support the devices:

- Configuring a large number of devices requires storage of more information in the ODM device-configuration database. It can also require more device special files. As a result, more space and more i-nodes are required of the file system.
- Some devices require more space than others in the ODM device-configuration database. The number of special files or i-nodes used also varies from device to device. As a result, the amount of space and i-nodes required of the file system depends on the types of devices on the system.
- Multipath I/O (MPIO) devices require more space than non-MPIO devices because information is stored in the ODM for the device itself as well as for each path to the device. As a rough guideline, assume that each path takes up the space of one-fifth of a device. For example, an MPIO device with five paths will have the space equivalent to two non-MPIO devices.
- AIX includes both logical devices and physical devices in the ODM device-configuration database. Logical devices include volume groups, logical volumes, network interfaces, and so on. In some cases, the relationship between logical and physical devices can greatly affect the total number of devices supported. For example, if you define a volume group with two logical volumes for each physical disk that is attached to a system, this will result in four AIX devices for each disk. On the other hand, if you define a volume group with six logical volumes for each physical disk, there will be eight AIX devices for each disk. Therefore, only half as many disks could be attached.
- Changing device attributes from their default settings results in a larger ODM device-configuration database and could lead to fewer devices that can be supported.
- More devices require more real memory.

Two file systems are used by AIX to support devices:

- The RAM file system is used during boot in an environment that has no paging space and no disk file systems mounted. The size of the RAM file system is 25% of the system memory size up to a maximum of 128 MB. One i-node is allocated for every KB in the RAM file system. The minimum system memory requirement for AIX 5.2 is 128 MB, which translates into a minimum RAM file system size of 32 MB with 32768 i-nodes. If the system memory size is 512 MB or larger, then the RAM file system will be at its maximum size of 128 MB with 131072 i-nodes. If either the amount of RAM file system space or number of i-nodes needed to support the attached devices exceeds what has been allocated to the RAM disk, the system might not boot. If this is the case, you must remove some of the devices.
- The space and i-nodes of the root file system (rootvg) on the disk can be increased as long as there are unallocated partitions in the rootvg. With AIX 5.2 and the minimum RAM file system size, it is likely

that up to 5000 AIX devices can be configured. With the maximum RAM file system size, it is likely that up to 25,000 AIX devices could be configured. These numbers include both physical and logical devices. Depending on the various factors mentioned in this section, your system might be able to configure more or fewer devices than this number.

Notes:

1. With a large number of devices in the system, the longer configuration time contributes to a longer boot time.
2. AIX 5L Version 5.2 with the 5200-03 Recommended Maintenance package and newer support IDE DVD-RAM devices.

Device Nodes

Devices are organized into clusters known as *nodes*. Each node is a logical subsystem of devices, where lower-level devices have a dependency on upper-level devices in child-parent relationships. For example, the system node is the highest of all nodes and consists of all the physical devices in the system. The system device is the top of the node and below that are the bus and adapters that have a dependency on the higher-level system device. At the bottom of the hierarchy are devices to which no other devices are connected. These devices have dependencies on all devices above them in the hierarchy.

At startup time, parent-child dependencies are used to configure all devices that make up a node. Configuration occurs from the top node down and any device having a dependency on a higher-level device is not configured until the higher-level device is configured.

Multiple-path I/O (MPIO) is a feature available with AIX 5.2 and later. If a device has an MPIO-capable device driver, it can have more than one parent within this hierarchy, which allows multiple, simultaneous communication paths between the device and a given machine or logical partition within a machine. For information about how this feature works, see “Device Configuration Database.”

Device Classes

Managing devices requires the operating system to comprehend what device connections are allowed. The operating system classifies devices hierarchically into three groups:

- Functional classes
- Functional subclasses
- Device types

Functional classes consist of devices that perform the same function. Printers, for example, comprise a functional class. Functional classes are grouped into subclasses according to certain device similarities. For example, printers have a serial or parallel interface. Serial printers are one subclass and parallel printers are another. Device types are classified according to their model and manufacturer.

Device classes define valid parent-child connections for the operating system. The hierarchy defines the possible subclasses that can be connected for each of the possible child connection locations. For example, the term RS-232 8-port adapter specifies that only devices belonging to the RS-232 subclass can be connected to any of the eight ports of the adapter.

Device classes and their hierarchical dependencies are maintained in an Object Data Manager (ODM) Device Configuration database.

Device Configuration Database

Device information is contained in a predefined database or a customized database that makes up the device configuration database. The predefined database contains configuration data for all possible devices supported by the system. The hierarchical device class information is contained in this database.

The customized database contains configuration data for all currently defined and configured devices in the system. A record is kept of each device currently connected to your system.

The Configuration Manager is a program that automatically configures devices on your system during system startup and run time. The Configuration Manger uses the information from the predefined and customized databases during this process, and updates the customized database afterwards.

Beginning with AIX 5.2, the Multiple-path I/O (MPIO) feature uses a unique device identifier (UDID) to identify each MPIO-capable device, regardless of the path on which it was discovered. The UDID is saved in the device configuration database. When a device is discovered, the UDIDs in the database are checked to determine whether the device is new or whether the discovery is another path to an existing device. When multiple paths to a device are detected, the device driver or the Path Control Manager kernel extension decides which path to use for a particular request. For information on managing MPIO-capable devices, see Managing MPIO-Capable Devices in *AIX 5L Version 5.3 System Management Guide: Operating System and Devices*.

Device States

Devices that are connected to the system can be in one of four states:

Undefined	The device is unknown to the system.
Defined	Specific information about the device is recorded in the customized database, but it is unavailable to the system.
Available	A defined device is coupled to the operating system, or the defined device is configured.
Stopped	The device is unavailable but remains known by its device driver.

If a tty device and a printer alternately use the same tty connector, both a tty device and a printer are defined on the same parent and port in the device configuration database. Only one of these devices can be configured at a time. When the tty connector is configured, the printer specific setup information is retained until it is configured again. The device is not removed; it is in the defined state. Maintaining a device in defined state retains customized information for a device that is not currently in use, either before it is first made available or while it is temporarily removed from the system.

If a device driver exists for a device, the device can be made available through the device driver.

Some devices, in particular TCP/IP pseudo-devices, need the stopped state.

Device Management

You can use the Web-based System Manager Devices application, SMIT, or operating system commands to perform device management tasks such as deleting, adding, or configuring a device.

Device Location Codes

The *location code* is a path from the CPU drawer or system unit through the adapter, signal cables, and the asynchronous distribution box (if there is one) to the device or workstation. This code is another way of identifying physical devices.

The location code consists of up to four fields of information depending on the type of device. These fields represent drawer, slot, connector, and port. Each of these fields consists of two characters.

The location code of a drawer consists of only the drawer field and is simply a two-character code. The location code of an adapter consists of the drawer and slot fields and has the format `AA-BB`, where `AA` corresponds to the drawer location and `BB` indicates the bus and slot that contains the adapter. Other devices have location codes of formats `AA-BB-CC` or `AA-BB-CC-DD`, where `AA-BB` is the location code of the

adapter to which the device is connected, CC corresponds to the connector on the adapter to which the device is connected, and DD corresponds to a port number or SCSI device address.

For information on finding the labels with the location codes on the hardware, see your operator guide.

Adapter Location Codes

The location code for an adapter consists of two pairs of digits with the format AA-BB, where AA identifies the location code of the drawer containing the adapter and BB identifies the I/O bus and slot containing the card.

A value of 00 for the AA field means that the adapter is located in the CPU drawer or system unit, depending on the type of system. Any other value for the AA field indicates that the card is located in an I/O expansion drawer. In this case, the AA value identifies the I/O bus and slot number in the CPU drawer that contains the asynchronous expansion adapter. The first digit identifies the I/O bus with 0 corresponding to the standard I/O bus and 1 corresponding to the optional I/O bus. The second digit identifies the slot number on the indicated I/O bus.

The first digit of the BB field identifies the I/O board containing the adapter card. If the card is in the CPU drawer or system unit, this digit is 0 for the standard I/O bus and 1 for the optional I/O bus. If the card is in an I/O expansion drawer, this digit is 0. The second digit identifies the slot number on the indicated I/O bus (or slot number in the I/O expansion drawer) that contains the card.

A location code of 00-00 is used to identify the standard I/O board.

Examples:

00-05 Identifies an adapter card in slot 5 of the standard I/O board and is located in either the CPU drawer or the system unit, depending on the type of system.

00-12 Identifies an adapter in slot 2 of the optional I/O bus and is located in the CPU drawer.

18-05 Identifies an adapter card located in slot 5 of an I/O expansion drawer. The drawer is connected to the asynchronous expansion adapter located in slot 8 of the optional I/O bus in the CPU drawer.

Printer and Plotter Location Codes

Location codes of 00-00-S1-00 or 00-00-S2-00 indicate the printer, plotter, or tty device is connected to the standard I/O board serial ports s1 or s2. A location code of 00-00-0P-00 indicates the parallel printer is connected to the standard I/O board parallel port.

Any other location code indicates that the printer, plotter, or tty device is connected to an adapter card other than the standard I/O board. For these printers, plotters, and tty devices, the location code format is AA-BB-CC-DD, where AA-BB indicates the location code of the controlling adapter.

AA A value of 00 for the AA field indicates the adapter card is located in the CPU drawer or system unit depending on the type of system. Any other value for the AA field indicates the card is located in an I/O expansion drawer; in which case, the first digit identifies the I/O bus and the second digit identifies the slot number on the bus, in the CPU drawer, that contains the asynchronous expansion adapter to which the I/O expansion drawer is connected.

BB The first digit of the BB field identifies the I/O bus containing the adapter card. If the card is in the CPU drawer or system unit, this digit is 0 for the standard I/O bus and 1 for the optional I/O bus. If the card is in an I/O expansion drawer, this digit is 0. The second digit identifies the slot number on the I/O bus (or slot number in the I/O expansion drawer) that contains the card.

CC The CC field identifies the connector on the adapter card where the asynchronous distribution box is connected. Possible values are 01, 02, 03, and 04.

DD The DD field identifies the port number on the asynchronous distribution box where the printer, plotter, or tty device is attached.

tty Location Codes

Location codes of 00-00-S1-00 or 00-00-S2-00 indicate the tty device is connected to the standard I/O serial ports s1 or s2.

Any other location code indicates the tty device is connected to an adapter card other than the standard I/O board. For these devices, the location code format is AA-BB-CC-DD, where AA-BB indicates the location code of the controlling adapter card.

AA
: A value of 00 for the AA field indicates the adapter card is located in the CPU drawer or system unit depending on the type of system. Any other value for the AA field indicates the card is located in an I/O expansion drawer. In this case, the first digit identifies the I/O bus, and the second digit identifies the slot number on the bus in the CPU drawer that contains the asynchronous expansion adapter where the I/O expansion drawer is connected.

BB
: The first digit of the BB field identifies the I/O bus containing the adapter card. If the card is in the CPU drawer or system unit, this digit will be 0 for the standard I/O bus and 1 for the optional I/O bus. If the card is in an I/O expansion drawer this digit is 0. The second digit identifies the slot number on the I/O bus (or slot number in the I/O expansion drawer) that contains the card.

CC
: The CC field identifies the connector on the adapter card where the asynchronous distribution box is connected. Possible values are 01, 02, 03, and 04.

DD
: The DD field identifies the port number on the asynchronous distribution box where the tty device is attached.

SCSI Device Location Codes

These location codes apply to all SCSI devices including:

- CD-ROMs
- Disks
- Initiator devices
- Read/write optical drives
- Tapes
- Target mode

The location code format is AA-BB-CC-S,L. The AA-BB fields identify the location code of the SCSI adapter controlling the SCSI device.

AA
: A value of 00 for the AA field indicates the controlling adapter card is located in the CPU drawer or system unit, depending on the type of system.

BB
: The BB field identifies the I/O bus and slot containing the card. The first digit identifies the I/O bus. It is 0 for the standard I/O bus and 1 for the optional I/O bus. The second digit is the slot on the indicated I/O bus containing the card. A value of 00 for the BB field indicates the standard SCSI controller.

CC
: The CC field identifies the SCSI bus of the card that the device is attached to. For a card that provides only a single SCSI bus, this field is set to 00. Otherwise, a value of 00 indicates a device attached to the internal SCSI bus of the card, and a value of 01 indicates a device attached to the external SCSI bus of the card.

S,L
: The S,L field identifies the SCSI ID and logical unit number (LUN) of the SCSI device. The S value indicates the SCSI ID and the L value indicates the LUN.

Direct-Bus-Attached Disk Location Codes

For a direct-attached disk device, the location code format is AA-BB. The AA field is a value of 00, that indicates the disk is located in the system unit. The BB field indicates the I/O bus and slot number where the disk is attached. The first digit is always 0, which indicates the disk is attached to the standard I/O bus. The second digit identifies the slot number on the standard I/O bus to which the disk is attached.

Serial-Linked Disk Location Codes

The location code for serial-linked disk drives is of the format AA-BB-CC-DD, where AA-BB indicates the location code of the controlling adapter card.

The individual fields are interpreted as follows:

AA A value of 00 for the AA field indicates the controlling adapter card is located in the CPU drawer or system unit, depending on the type of system.

BB The BB field identifies the I/O bus and slot containing the card. The first digit identifies the I/O bus. It is 0 for the standard I/O bus and 1 for the optional I/O bus. The second digit is the slot on the indicated I/O bus that contains the card.

CC The CC field identifies the connector on the adapter card where the controller drawer is attached. Possible values are 00, 01, 02, and 03.

DD The DD field identifies the logical unit number (LUN) of the disk. This corresponds to the slot in the drawer where the disk resides.

Diskette Drive Location Codes

Diskette drives have location codes of either 00-00-0D-01 or 00-00-0D-02, indicating that they are attached to the standard I/O planar diskette ports 0 or 1.

Dials/LPFKeys Location Codes

For a Dials/LPFKeys device attached to a graphics input adapter, the location code format is AA-BB-CC.

The individual fields are interpreted as follows:

AA A value of 00 for the AA field indicates the controlling adapter card is located in the CPU drawer or system unit, depending on the type of system.

BB The BB field identifies the I/O bus and slot containing the card. The first digit identifies the I/O bus. It is 0 for the standard I/O bus and 1 for the optional I/O bus. The second digit is the slot on the indicated I/O bus that contains the card.

CC The CC field indicates the card connector where the device is attached. The value is either 01 or 02, depending on whether the attached device is port 1 or port 2 on the card.

Note: Serially attached Dials/LPFKeys devices do not indicate location codes. This is because these devices are considered to be attached to a tty device. The tty device is specified by the user during Dials/LPFKeys definition.

Multiprotocol Port Location Codes

The location code for a multiprotocol port is of the format AA-BB-CC-DD where AA-BB indicates the location code of the multiprotocol adapter card.

The individual fields are interpreted as follows:

AA A value of 00 for the AA field indicates the multiprotocol adapter card is located in the CPU drawer or system unit, depending on the type of system.

BB The BB field identifies the I/O bus and slot containing the card. The first digit identifies the I/O bus. It is 0 for the standard I/O bus and 1 for the optional I/O bus. The second digit is the slot on the indicated I/O bus that contains the card.

CC The CC field identifies the connector on the adapter card to which the multiprotocol distribution box is connected. The value is always 01.

DD The DD field identifies the physical port number on the multiprotocol distribution box. Possible values are 00, 01, 02, and 03.

PCI Hot Plug Management

You can insert a new PCI hot plug adapter into an available PCI slot while the operating system is running. This can be another adapter of the same type that is currently installed or of a different type of PCI adapter. New resources are made available to the operating system and applications without having to restart the operating system. Some reasons for adding a hot plug adapter might include:

- Adding additional function or capacity to your existing hardware and firmware.
- Migrating PCI adapters from a system that no longer requires the function provided by those adapters.
- Installing a new system for which adapter cards become available after the initial configuration of optional hardware subsystems, including PCI adapters, and the installation and start of the operating system.

Note: If you add an adapter using a PCI hot plug replace or add operation, or using Dynamic Logical Partitioning, it and its child devices may not be available for specification as a boot device using the **bootlist** command. You may have to restart the machine to make all potential boot devices known to the operating system. An adapter already listed in the boot list, which is then replaced by the exact type of adapter, is still a valid boot device.

You can also remove a defective or failing PCI hot plug adapter or exchange it with another of the same type without shutting down or powering off the system. When you exchange the adapter, the existing device driver supports the adapter because it is of the same type. Device configuration and configuration information about devices below the adapter are retained for the replacement device. Some reasons for replacing an adapter might include:

- Temporarily replacing the card to aid in determining a problem or to isolate a failing FRU.
- Replacing a flawed, failing, or intermittently failing adapter with a functional card.
- Replacing a failing redundant adapter in an HACMP™ or multipath I/Oconfiguration.

When you remove a hot plug adapter, the resources provided by the adapter become unavailable to the operating system and applications. Some reasons for removing an adapter might include:

- Removing existing I/O subsystems.
- Removing an adapter that is no longer required or is failing and a replacement card is not available.
- Migrating an adapter to another system when the function is no longer required on the system from which it is being removed.

Before you can remove or replace a hot plug device, it must be unconfigured. The associated device driver must free any system resources that it has allocated for the device. This includes unpinning and freeing memory, undefining interrupt and EPOW handlers, releasing DMA and timer resources, and any other required steps. The driver must also ensure that interrupts, bus memory, and bus I/O are disabled on the device.

The system administrator must perform the following tasks before and after removing an adapter:

- Terminate and restore applications, daemons, or processes using the device.
- Unmount and remount file systems.
- Remove and recreate device definitions and perform other operations necessary to free up a device in use.
- Put the system into a safe state to be serviced.
- Obtain and install any required device drivers.

The remove and replace operations fail unless the device connected to the identified slot has been unconfigured and is in the defined state. You can do this with the rmdev command. Before placing the adapter in the defined state, close all applications that are using the adapter, otherwise, the command will be unsuccessful.

In some cases, the you can also perform the following tasks:

- Prepare a PCI hot plug adapter to be inserted, removed, or replaced.
- Identify slots or PCI adapters that are involved in the hot plug operation.
- Remove or insert PCI hot plug adapters.

Attention: Although PCI hot plug management provides the capability of adding, removing, and replacing PCI adapters without powering off the system or restarting the operating system, not all devices in hot plug slots can be managed in this fashion. For example, the hard disk that makes up the rootvg volume group or the I/O controller to which it is attached cannot be removed or replaced without powering off the system because it is necessary for running the operating system. If the rootvg volume group is mirrored, you can use the **chpv** command to take the disks offline. If the rootvg volume group resides on one or more disks that are Multi-Path I/O (MPIO) enabled and connected to multiple I/O controllers, one of these I/O controllers can be removed (or replaced) without rebooting the system. In this situation, all paths from the I/O controller to be removed (or replaced) should be unconfigured using the **rmdev -R** command on the adapter. This will unconfigure the paths and the adapter. You can then proceed with Hot Plug Management. Before you attempt to remove or insert PCI hot plug adapters, refer to the *PCI Adapter Placement Reference*, (shipped with system units that support hot plug), to determine whether your adapter can be hot-swapped. Refer to your system unit documentation for instructions for installing or removing adapters.

For instructions on how to manage a PCI hot plug adapter, see the following articles in the *AIX 5L Version 5.3 System Management Guide: Operating System and Devices*:

- Adding a PCI Hot Plug Adapter
- Removing or Replacing a PCI Hot Plug Adapter

Unconfiguring PCI Communications Adapters

This section provides an overview of the process for unconfiguring PCI communications adapters. This includes Ethernet, Token-ring, FDDI, and ATM adapters. For procedural information, see Unconfiguring Communications Adapters in the *AIX 5L Version 5.3 System Management Guide: Operating System and Devices*.

If your application is using TCP/IP protocol, you must remove the TCP/IP interface for the adapter from the network interface list before you can place the adapter in the defined state. Use the netstat command to determine whether your adapter is configured for TCP/IP and to check the active network interfaces on your adapter.

An Ethernet adapter can have two interfaces: Standard Ethernet (en*X*) or IEEE 802.3 (et*X*). *X* is the same number in the **ent***X* adapter name. Only one of these interfaces can be using TCP/IP at a time. For example, Ethernet adapter **ent0** can have en0 and et0 interfaces.

A Token ring adapter can have only one interface: Token-ring (tr*X*). *X* is the same number in the **tok***X* adapter name. For example, Token-ring adapter **tok0** has a tr0 interface.

An ATM adapter can have only one atm interface: ATM (at*X*). *X* is the same number in the **atm***X* adapter name. For example, ATM adapter **atm0** has an at0 interface. However, ATM adapters can have multiple emulated clients running over a single adapter.

The ifconfig command removes an interface from the network. The rmdev command unconfigures the PCI device while retaining its device definition in the Customized Devices Object Class. Once the adapter is in the defined state, you can use the drslot command to remove the adapter.

Example

1. To unconfigure the children of PCI bus `pci1` and all other devices under them while retaining their device definitions in the Customized Devices object class, type:

```
rmdev  -p pci1
```

The system displays a message similar to the following:

```
rmt0 Defined
hdisk1 Defined
scsi1 Defined
ent0 Defined
```

Multiple Path I/O

With Multiple Path I/O (MPIO), a device can be uniquely detected through one or more physical connections, or *paths*. A path-control module (PCM) provides the path management functions.

An MPIO-capable device driver can control more than one type of target device. A PCM can support one or more specific devices. Therefore, one device driver can be interfaced to multiple PCMs that control the I/O across the paths to each of the target devices.

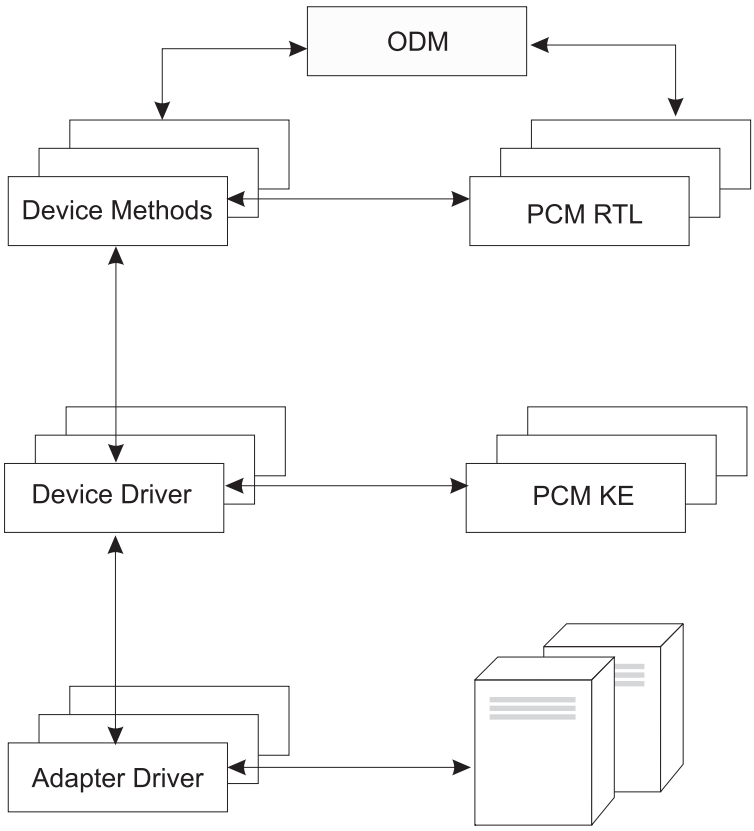

Figure 12. MPIO Component Interaction. This illustration shows the interaction between the different components that make up the MPIO solution. In this figure, the MPIO device driver controls multiple types of target devices, each requiring a different PCM. (KE=Kernel Extension, RTL=Run-time Loadable).

Before a device can take advantage of MPIO, the device's driver, methods, and predefined attributes in the Object Data Manager (ODM) must be modified to support detection, configuration, and management of multiple paths. The parallel SCSI and Fibre Channel disk device drivers and their device methods support MPIO disk devices. Beginning with AIX 5L Version 5.2 with the 5200-04 Recommended Maintenance package, iSCSI disk devices are supported as MPIO devices. Beginning with AIX 5.3, the Fibre Channel tape device driver and its device methods have been modified to support MPIO tape devices. Also, the predefined attributes for some devices in the ODM have been modified for MPIO.

The AIX PCM consists of: the PCM RTL configuration module and the PCM KE kernel extension. The PCM RTL is a run-time loadable module that enables the device methods to detect additional PCM KE device-specific or path ODM attributes that the PCM KE requires. The PCM RTL is loaded by a device method. One or more routines within the PCM RTL are then accessed to perform specific operations that initialize or modify PM KE variables.

The PCM KE supplies path-control management capabilities to any device driver that supports the MPIO interface. The PCM KE depends on device configuration to detect paths and communicate that information to the device driver. Each MPIO-capable device driver adds the paths to a device from its immediate parent or parents. The maintenance and scheduling of I/O across different paths is provided by the PCM KE and is not apparent to the MPIO-capable device driver.

The PCM KE can provide more than one routing algorithm, which can be selected by the user. The PCM KE also helps collect information that can be used to determine and select the best path for any I/O request. The PCM KE can select the best path based on a variety of criteria, including load balancing, connection speed, connection failure, and so on.

The AIX PCM has a health-check capability that can be used to do the following:

- Check the paths and determine which paths are currently usable for sending I/O
- Enable a path that was previously marked failed because of a temporary path fault (for example, when a cable to a device was removed and then reconnected)
- Check currently unused paths that would be used if a failover occurred (for example, when the algorithm attribute value is `failover`, the health check can test the alternate paths)

Not all disk devices and tape devices can be detected and configured using the AIX default PCMs. The AIX default PCMs consist of two path control modules, one to manage disk devices and another to mange tape devices. If your device is not detected, check with the device vendor to determine if a PCM is available for your device.

Configuring an MPIO Device

Configuring an MPIO-capable device uses the same commands as a non-MPIO device. Beginning with AIX 5.2 with 5200-01, the **cfgmgr**, **mkdev**, **chdev**, **rmdev** and **lsdev** commands support managing MPIO device instances and display their attributes. An MPIO device instance also has path instances associated with the device instance. The **mkpath**, **chpath**, **rmpath**, and **lspath** commands manage path instances and display their attributes.

A path instance can be added or removed from a MPIO device without unconfiguring the device. For more information about these commands, see Managing MPIO-Capable Devices in the *AIX 5L Version 5.3 System Management Guide: Operating System and Devices*.

An MPIO device can have additional attributes, but the required attributes that all MPIO devices must support are **reserve_policy** and **algorithm**. The **reserve_policy** attribute determines the type of reserve methodology the device driver will implement when the device is opened, and it can be used to limit device access from other adapters, whether on the same system or another system. The **algorithm** attribute defines the methodology that the PCM uses to manage I/O across the paths configured for a device. For more information about the **reserve_policy** attribute, see “MPIO Device Attributes” on page 135. For more information about the **algorithm** attribute, see “Path Control Module Attributes” on page 136.

Supported Multi-Path Devices

The AIX default PCMs support a set of disk devices and tape devices defined in the **devices.common.IBM.mpio.rte** fileset. Devices not supported by the AIX disk PCMs or tape PCMs require the device vendor to provide attributes predefined in the ODM, a PCM, and any other supporting code necessary to recognize the device as MPIO-capable.

To determine which disk devices are supported by the AIX disk PCM, run the following script :

```
odmget -qDvDr=aixdiskpcmke PdDv | grep uniquetype | while read line
do
      utype=`echo $line | cut -d'"' -f2`
      dvc=`odmget -q"uniquetype=$utype AND attribute=dvc_support" PdAt`
      echo $dvc | grep values | cut -d'"' -f2
done
```

To determine which tape devices are supported by the AIX disk PCM, run the following script :

```
odmget -qDvDr=aixtapepcmke PdDv | grep uniquetype | while read line
do
      utype=`echo $line | cut -d'"' -f2`
      dvc=`odmget -q"uniquetype=$utype AND attribute=dvc_support" PdAt`
      echo $dvc | grep values | cut -d'"' -f2
done
```

The script output displays a list of unique device types supported by the AIX default PCMs. The three device types supported by the AIX Disk PCM are *self-configuring parallel SCSI disk* (**disk/scsi/scsd**) and *MPIO other FC disk*(**disk/fcp/mpioosdisk**), and *MPIO other iSCI* (**disk/iscsi/mpioosdisk**). The device type supported by the AIX tape PCM is *MPIO other FC tape* (**tape/fcp/mpioost**).

MPIO other FC disk and *MPIO other FC tape* devices are a subset of other Fibre Channel disks and other Fibre Channel tapes, respectively. A device is supported as an *MPIO other FC* device only if there are no vendor-provided ODM predefined attributes and the device has been certified to operate with one of the AIX default PCMs. Certification does not guarantee that all device capabilities are supported when the device is configured as an *MPIO other FC* device.

An *MPIO other iSCSI* disk is a subset of other iSCSI disks. A device is supported as an *MPIO other iSCSI* disk only if there is no vendor-provided ODM predefined attributes and the device has been certified to operate with the AIX PCM. Certification does not guarantee that all device capabilities are supported when the device is configured as an *MPIO other iSCSI* disk.

To determine which devices are supported as *MPIO other FC* disk, run the following script:

```
odmget –quniquetype=disk/fcp/mpioosdisk PdAt | grep deflt | cut -d'"' -f2
```

To determine which devices are supported as *MPIO other FC* tape, run the following script:

```
odmget -q "uniquetype=tape/fcp/mpioosdisk AND attribute=mpio_model_map PdAt | grep deflt | cut -d'"' -f2
```

To determine which devices are supported as *MPIO other iSCSI* disks, run the following script:

```
odmget –quniquetype=disk/iscsi/mpioosdisk PdAt | grep deflt | cut -d'"' -f2
```

The script output displays a list of inquiry data that contains the device vendor and model.

To display all MPIO-capable devices that are installed on the system, run the following script:

```
odmget -qattribute=unique_id PdAt | grep uniquetype | cut -d'"' -f2
```

The script output displays a list of unique MPIO-capable device types, supported by the AIX default PCMs and vendor provided PCMs.

MPIO Device Attributes

This section describes attributes that are supported only by multi-path devices. The attributes can be displayed or changed using the Web-based System Manager, SMIT, or commands (in particular, the **lsattr** and the **chdev** commands).

The required device attributes that all MPIO devices must support is **reserve_policy**. Typically, a multi-path device also has the **PR_key** device attribute. A multi-path device can have additional device-specific attributes. Other device-specific attributes are as follows:

reserve_policy
Defines whether a reservation methodology is employed when the device is opened. The values are as follows:

no_reserve
Does not apply a reservation methodology for the device. The device might be accessed by other initiators, and these initiators might be on other host systems.

single_path_reserve
Applies a SCSI2 reserve methodology for the device, which means the device can be accessed only by the initiator that issued the reserve. This policy prevents other initiators in the same host or on other hosts from accessing the device. This policy uses the SCSI2 reserve to lock the device to a single initiator (path), and commands routed through any other path result in a reservation conflict.

Path selection algorithms that alternate commands among multiple paths can result in thrashing when the **single_path_reserve** value is selected. As an example, assume a device-specific PCM has a required attribute that is set to a value that distributes I/O across multiple paths. When **single_path_reserve** is in effect, the disk driver must issue a bus device reset (BDR) and then issue a reserve using a new path for sending the next command to break the previous reservation. Each time a different path is selected, thrashing occurs and performance degrades because of the overhead of sending a BDR and issuing a reserve to the target device. (The AIX PCM does not allow you to select an algorithm that could cause thrashing.)

PR_exclusive
Applies a SCSI3 persistent-reserve, exclusive-host methodology when the device is opened. The **PR_key** attribute value must be unique for each host system. The **PR_key** attribute is used to prevent access to the device by initiators from other host systems.

PR_shared
Applies a SCSI3 persistent-reserve, shared-host methodology when the device is opened. The **PR_key** value must be a unique value for each host system. Initiators from other host systems are required to register before they can access the device.

PR_key
Required only if the device supports any of the persistent reserve policies (**PR_exclusive** or **PR_shared**).

Path Control Module Attributes

In addition to the default AIX default PCMs, a device-specific PCM might be supplied by a device vendor. The set of user changeable attributes is defined by the device vendor. A device-specific PCM can have device and path attributes.

The following are the device attributes for the AIX default PCMs:

algorithm
Determines the methodology by which the I/O is distributed across the paths for a device. The algorithm attribute has the following values:

failover
Sends all I/O down a single path. If the path is determined to be faulty, an alternate path is selected for sending all I/O. This algorithm keeps track of all the enabled paths in an ordered list. If the path being used to send I/O becomes marked failed or disabled, the

next enabled path in the list is selected. The sequence within the list is determined by the path priority “path priority” attribute. This algorithm is available in both AIX disk PCM and AIX tape PCM.

round_robin
Distributes the I/O across all enabled paths. The path priority is determined by the “path priority” path priority attribute value. If a path becomes marked failed or disabled, it is no longer used for sending I/O. The priority of the remaining paths is then recalculated to determine the percentage of I/O that should be sent down each path. If all paths have the same value, then the I/O is then equally distributed across all enabled paths. This algorithm is available only in the AIX disk PCM; AIX tape PCM does not support **round_robin**.

hcheck_mode
Determines which paths should be checked when the health check capability is used. The attribute supports the following modes:

enabled
Sends the **healthcheck** command down paths with a state of enabled.

failed Sends the **healthcheck** command down paths with a state of failed.

nonactive
(Default) Sends the **healthcheck** command down paths that have no active I/O, including paths with a state of failed. If the algorithm selected is failover, then the **healthcheck** command is also sent on each of the paths that have a state of enabled but have no active I/O. If the algorithm selected is round_robin, then the **healthcheck** command is only sent on paths with a state of failed, because the round_robin algorithm keeps all enabled paths active with I/O.

hcheck_interval
Defines how often the health check is performed on the paths for a device. The attribute supports a range from 0 to 3600 seconds. When a value of 0 is selected, health checking is disabled.

dist_tw_width
Defines the duration of a ″time window″. This is the time frame during which the distributed error detection algorithm will cumulate I/Os returning with an error. The **dist_tw_width** attribute unit of measure is milliseconds. Lowering this attributes value decreases the time duration of each sample taken and decreases the algorithms sensitivity to small bursts of I/O errors. Increasing this attribute’s value will increase the algorithms sensitivity to small bursts of errors and the probability of failing a path.

dist_err_percent
Defines the percentage of ″time windows″ having an error allowed on a path before the path is failed due to poor performance. The **dist_err_percent** has a range from 0-100. The distributed error detection algorithm will be disabled when the attribute is set to zero (0). The default setting is zero. The distributed error detection algorithm samples the fabric connecting the device to the adapter for errors. The algorithm calculates a percentage of samples with errors and will fail a path if the calculated value is larger than the **dist_err_percent** attribute value.

The following is the path attribute for the AIX PCM:

path priority
Modifies the behavior of the algorithm methodology on the list of paths.

When the algorithm attribute value is failover, the paths are kept in a list. The sequence in this list determines which path is selected first and, if a path fails, which path is selected next. The sequence is determined by the value of the path priority attribute. A priority of 1 is the highest priority. Multiple paths can have the same priority value, but if all paths have the same value, selection is based on when each path was configured.

When the algorithm attribute value is **round_robin**, the sequence is determined by percent of I/O. The path priority value determines the percentage of the I/O that should be processed down each path. I/O is distributed across the enabled paths. A path is selected until it meets its required percentage. The algorithm then marks that path failed or disabled to keep the distribution of I/O requests based on the path priority value.

Chapter 16. Tape Drives

This chapter covers system management functions related to tape drives. Many of these functions change or get information from the device configuration database, which contains information about the devices on your system. The device configuration database consists of the predefined configuration database, which contains information about all possible types of devices supported on the system, and the customized configuration database, which contains information about the particular devices currently on the system. For the operating system to make use of a tape drive, or any other device, the device must be defined in the customized configuration database and must have a device type defined in the predefined configuration database.

Topics included in this chapter are:

Basic tasks for Tape Drives are listed in Chapter 16, "Tape Drives" in *AIX 5L Version 5.3 System Management Guide: Operating System and Devices*.

Tape Drive Attributes

The following describes tape drive attributes you can adjust to meet the needs of your system. The attributes can be displayed or changed using the Web-based System Manager Devices application, SMIT, or commands (in particular, the **lsattr** and the **chdev** commands).

Each type of tape drive only uses a subset of all the attributes.

General Information about Each Attribute

Block Size

The block size attribute indicates the block size to use when reading or writing the tape. Data is written to tape in blocks of data, with inter-record gaps between blocks. Larger records are useful when writing to unformatted tape, because the number of inter-record gaps is reduced across the tape, allowing more data to be written. A value of **0** indicates variable length blocks. The allowable values and default values vary depending on the tape drive.

Device Buffers

Setting the Device Buffers attribute (using the **chdev** command) to `mode=yes` indicates an application is notified of write completion after the data has been transferred to the data buffer of the tape drive, but not necessarily after the data is actually written to the tape. If you specify the `mode=no`, an application is notified of write completion only after the data is actually written to the tape. Streaming mode cannot be maintained for reading or writing if this attribute is set to the `mode=no` value. The default value is `mode=yes`.

With the `mode=no` value, the tape drive is slower but has more complete data in the event of a power outage or system failure and allows better handling of end-of-media conditions.

Extended File Marks

Setting the Extended File Marks attribute (for **chdev**command, the **extfm** attribute) to the `no` value writes a regular file mark to tape whenever a file mark is written. Setting this attribute to the `yes` value writes an extended file mark. For tape drives, this attribute can be set on. The default value is `no`. For example, extended filemarks on 8 mm tape drives use 2.2 MB of tape and can take up to 8.5 seconds to write. Regular file marks use 184 KB and take approximately 1.5 seconds to write.

To reduce errors when you use an 8 mm tape in append mode, use extended file marks for better positioning after reverse operations at file marks.

Retension

Setting the Retension attribute (for the **chdev** command, the **ret** attribute) to `ret=yes` instructs the tape drive to re-tension a tape automatically whenever a tape is inserted or the drive is reset. *Retensioning* a tape means to wind to the end of the tape and then rewind to the beginning of the tape to even the tension throughout the tape. Retensioning the tape can reduce errors, but this action can take several minutes. If you specify the `ret=no` value, the tape drive does not automatically retension the tape. The default value is `yes`.

Density Setting #1, and Density Setting #2

Density Setting #1 (for the **chdev** command, the **density_set_1** attribute) sets the density value that the tape drive writes when using special files **/dev/rmt***, **/dev/rmt*.1**, **/dev/rmt*.2**, and **/dev/rmt*.3**. Density Setting #2 (for **chdev**, the **density_set_2** attribute) sets the density value that the tape drive writes when using special files **/dev/rmt*.4**, **/dev/rmt*.5**, **/dev/rmt*.6**, and **/dev/rmt*.7**. See “Special Files for Tape Drives” on page 149 for more information.

The density settings are represented as decimal numbers in the range **0** to **255**. A zero (**0**) setting selects the default density for the tape drive, which is usually the high density setting of the drive. Specific permitted values and their meanings vary with different types of tape drives. These attributes do not affect the ability of the tape drive to read tapes written in all densities supported by the tape drive. It is customary to set Density Setting #1 to the highest density possible on the tape drive and Density Setting #2 to the second highest density possible on the tape drive.

Reserve Support

For tape drives that use the Reserve attribute (for the **chdev** command, the **res_support** attribute), specifying the value `res_support=yes` causes the tape drive to be reserved on the SCSI bus while it is open. If more than one SCSI adapter shares the tape device, this ensures access by a single adapter while the device is open. Some SCSI tape drives do not support the reserve or release commands. Some SCSI tape drives have a predefined value for this attribute so that reserve or release commands are always supported.

Variable Length Block Size

The Variable Length Block Size attribute (for the **chdev** command, the **var_block_size** attribute) specifies the block size required by the tape drive when writing variable length records. Some SCSI tape drives require that a nonzero block size be specified in their Mode Select data even when writing variable length records. The **Block Size** attribute is set to **0** to indicate variable length records. See the specific tape drive SCSI specification to determine whether or not this is required.

Data Compression

Setting the Data Compression attribute (for the **chdev** command, the **compress** attribute) to `compress=yes` causes the tape drive to be in compress mode, if the drive is capable of compressing data. If so, then the drive writes data to the tape in compressed format so that more data fits on a single tape. Setting this attribute to `no` forces the tape drive to write in native mode (noncompressed). Read operations are not affected by the setting of this attribute. The default setting is `yes`.

Autoloader

Setting the Autoloader attribute (for the **chdev** command, the **autoload** attribute) to `autoload=yes` causes Autoloader to be active, if the drive is so equipped. If so, and another tape is available in the loader, any read or write operation that advances the tape to the end is automatically continued on the next tape. Tape drive commands that are restricted to a single tape cartridge are unaffected. The default setting is `yes`.

Retry Delay

The Retry Delay attribute sets the number of seconds that the system waits after a command has failed before reissuing the command. The system may reissue a failed command up to four times. This attribute applies only to type OST tape drives. The default setting is `45`.

Read/Write Timeout
The Read/Write Timeout or Maximum Delay for a **READ/WRITE** attribute sets the maximum number of seconds that the system allows for a read or write command to complete. This attribute applies only to type OST tape drives. The default setting is `144`.

Return Error on Tape Change
The Return Error on Tape Change or Reset attribute, when set, causes an error to be returned on open when the tape drive has been reset or the tape has been changed. A previous operation to the tape drive must have taken place that left the tape positioned beyond beginning of tape upon closing. The error returned is a `-1` and `errno` is set to `EIO`. Once presented to the application, the error condition is cleared. Also, reconfiguring the tape drive itself will clear the error condition.

Attributes for 2.0 GB 4 mm Tape Drives (Type 4mm2gb)

Block Size
The default value is 1024.

Device Buffers
The general information for this attribute applies to this tape drive type.

Attributes with Fixed Values
If a tape drive is configured as a 2.0 GB 4 mm tape drive, the attributes for Retension, Reserve Support, Variable Length Block Size, Density Setting #1, and Density Setting #2 have predefined values that cannot be changed. The density settings are predefined because the tape drive always writes in 2.0 GB mode.

Attributes for 4.0 GB 4 mm Tape Drives (Type 4mm4gb)

Block Size
The default value is 1024.

Device Buffers
The general information for this attribute applies to this tape drive type.

Density Setting #1 and Density Setting #2
The user cannot change the density setting of this drive; the device reconfigures itself automatically depending on the Digital Data Storage (DDS) media type installed, as follows:

Media Type	Device Configuration
DDS	Read-only.
DDS |||	Read/write in 2.0 GB mode only.
DDS2	Read in either density; write in 4.0 GB mode only.
non-DDS	Not supported; cartridge will eject.

Data Compression
The general information for this attribute applies to this tape drive type.

Attributes with Fixed Values
If a tape drive is configured as a 4.0 GB 4 mm tape drive, the Retension, Reserve Support, Variable Length Block Size, Density Setting #1, and Density Setting #2 attributes have predefined values that cannot be changed.

Attributes for 2.3 GB 8 mm Tape Drives (Type 8mm)

Block Size
The default value is 1024. A smaller value reduces the amount of data stored on a tape.

Device Buffers
The general information for this attribute applies to this tape drive type.

Extended File Marks
The general information for this attribute applies to this tape drive type.

Attributes with Fixed Values
If a tape drive is configured as a 2.3 GB 8mm tape drive, the Retension, Reserve Support, Variable Length Block Size, Data Compression, Density Setting #1, and Density Setting #2 attributes have predefined values which cannot be changed. The density settings are predefined because the tape drive always writes in 2.3 GB mode.

Attributes for 5.0GB 8mm Tape Drives (Type 8mm5gb)

Block Size
The default value is 1024. If a tape is being written in 2.3 GB mode, a smaller value reduces the amount of data stored on a tape.

Device Buffers
The general information for this attribute applies to this tape drive type.

Extended File Marks
The general information for this attribute applies to this tape drive type.

Density Setting #1 and Density Setting #2
The following settings apply:

Setting	Meaning
140	5 GB mode (compression capable)
21	5 GB mode noncompressed tape
20	2.3 GB mode
0	Default (5.0 GB mode)

The default values are 140 for Density Setting #1, and 20 for Density Setting #2. A value of 21 for Density Setting #1 or #2 permits the user to read or write a noncompressed tape in 5 GB mode.

Data Compression
The general information for this attribute applies to this tape drive type.

Attributes with Fixed Values
If a tape drive is configured as a 5.0 GB 8 mm tape drive, the Retension, Reserve Support, and Variable Length Block Size attributes have predefined values which cannot be changed.

Attributes for 20000 MB 8 mm Tape Drives (Self Configuring)

Block Size
The default value is 1024.

Device Buffers
The general information for this attribute applies to this tape drive type.

Extended File Marks
The general information for this attribute applies to this tape drive type.

Density Setting #1 and Density Setting #2

The drive can read and write data cartridges in 20.0 GB format. During a Read command, the drive automatically determines which format is written on tape. During a Write, the Density Setting determines which data format is written to tape.

The following settings apply:

Setting	Meaning
39	20 GB mode (compression capable)
0	Default (20.0 GB mode)

The default value is 39 for Density Setting #1 and Density Setting #2.

Data Compression

The general information for this attribute applies to this tape drive type.

Attributes with Fixed Values

If a tape drive is configured as a 20.0 GB 8 mm tape drive, the Retension, Reserve Support, and Variable Length Block Size attributes have predefined values which cannot be changed.

Attributes for 35 GB Tape Drives (Type 35gb)

Block Size

The IBM 7205 Model 311 throughput is sensitive to block size. The minimum recommended block size for this drive is 32 K Bytes. Any block size less than 32 K Bytes restricts the data rate (backup/restore time). The following table lists recommended block sizes by command:

Supported Command	Default Block Size (Bytes)	RECOMMENDATION
BACKUP	32 K or 51.2 K (default)	Uses either 32 K or 51.2 K depending on if ″Backup″ is by name or not. No change is required.
TAR	10 K	There is an error in the manual that states a 512 K byte block size. Set the Blocking Parameter to **-N64**.
MKSYSB	See BACKUP	MKSYSB uses the BACKUP Command. No change is required.
DD	n/a	Set the Blocking Parameter to **bs=32K**.
CPIO	n/a	Set the Blocking Parameter to **-C64**.

Note: You should be aware of the capacity and throughput when you select a blocksize. Small blocksizes have a significant impact on performance and a minimal impact on capacity. The capacities of the 2.6 GB format (density) and 6.0 GB format (density) are significantly impacted when you use smaller than the recommended blocksizes. As an example: using a blocksize of 1024 bytes to backup 32 GB of data takes approximately 22 hours. Backing up the same 32 GB of data using a blocksize of 32 K Bytes takes approximately 2 hours.

Device Buffers

The general information for this attribute applies to this tape drive type.

Extended File Marks

The general information for this attribute applies to this tape drive type.

Density Setting #1 and Density Setting #2

The following chart shows the Supported Data Cartridge type and Density Settings (in decimal and hex) for the IBM 7205-311 Tape Drive. When you perform a Restore (Read) Operation, the tape drive automatically sets the density to match the written density. When you perform a Backup Operation (Write), you must set the Density Setting to match the Data Cartridge that you are using.

Supported Data Cartridges	Native Capacity	Compressed Data Capacity	Web-based System Manager or SMIT Density Setting	HEX Density Setting
DLTtape III	2.6 GB	2.6 GB (No Compression)	23	17h
	6.0 GB	6.0 GB (No Compression)	24	18h
	10.0 GB	20.0 GB (Default for drive)	25	19h
DLTtapeIIIxt	15.0 GB	30.6 GB (Default for drive)	25	19h
DLTtapeIV	20.0 GB	40.0 GB	26	1Ah
	35.0 GB	70.0 GB (Default for drive)	27	1Bh

Note: If you request an unsupported Native Capacity for the Data Cartridge, the drive defaults to the highest supported capacity for the Data Cartridge that is loaded into the drive.

Data Compression

The actual compression depends on the type of data being that is being written. (see above table) A Compression Ratio of 2:1 is assumed for this Compressed Data Capacity.

Attributes with Fixed Values

The general information for this attribute applies to this tape drive type.

Attributes for 150 MB 1/4-Inch Tape Drives (Type 150mb)

Block Size

The default block size is 512. The only other valid block size is **0** for variable length blocks.

Device Buffers

The general information for this attribute applies to this tape drive type.

Extended File Marks

Writing to a 1/4-inch tape can only occur at the beginning of tape (BOT) or after blank tape is detected. If data exists on the tape, you cannot overwrite the data except at BOT. If you wish to add data to a tape that has been written and then rewound, you must space forward until the next file mark is detected, which causes the system to return an error. Only then can you start writing again.

Retension

The general information for this attribute applies to this tape drive type.

Density Setting #1 and Density Setting #2

The following settings apply:

Setting	Meaning
16	QIC-150
15	QIC-120
0	Default (QIC-150), or whatever was the last density setting by a using system.

The default values are 16 for Density Setting #1, and 15 for Density Setting #2.

Attributes with Fixed Values
If a tape drive is configured as a 150 MB 1/4-inch tape drive, attributes for Extended File Marks, Reserve Support, Variable Length Block Size, and Data Compression have predefined values which cannot be changed.

Attributes for 525 MB 1/4-Inch Tape Drives (Type 525mb)

Block Size
The default block size is 512. The other valid block sizes are 0 for variable length blocks, and 1024.

Device Buffers
The general information for this attribute applies to this tape drive type.

Extended File Marks
Writing to a 1/4-inch tape can only occur at the beginning of tape (BOT) or after blank tape is detected. If data exists on the tape, you cannot overwrite the data except at BOT. If you want to add data to a tape that has been written and then rewound, you must space forward until the next file mark is detected, which causes the system to return an error. Only then can you start writing again.

Retension
The general information for this attribute applies to this tape drive type.

Density Setting #1 and Density Setting #2
The following settings apply:

Setting	Meaning
17	QIC-525*
16	QIC-150
15	QIC-120
0	Default (QIC-525), or whatever was the last density setting by a using system.

* QIC-525 is the only mode that supports the 1024 block size.

The default values are 17 for Density Setting #1, and 16 for Density Setting #2.

Attributes with Fixed Values
If a tape drive is configured as a 525 MB 1/4-inch tape drive, the Extended File Marks, Reserve Support, Variable Length Block Size, and Data Compression attributes have predefined values which cannot be changed.

Attributes for 1200 MB 1/4-Inch Tape Drives (Type 1200mb-c)

Block Size
The default block size is 512. The other valid block sizes are 0 for variable length blocks, and 1024.

Device Buffers
The general information for this attribute applies to this tape drive type.

Extended File Marks
Writing to a 1/4-inch tape can only occur at the beginning of tape (BOT) or after blank tape is detected. If data exists on the tape, you cannot overwrite the data except at BOT. If you wish to add data to a tape that has been written and then rewound, you must space forward until the next file mark is detected, which causes the system to return an error. Only then can you start writing again.

Retension
The general information for this attribute applies to this tape drive type.

Density Setting #1 and Density Setting #2
The following settings apply:

Setting	Meaning
21	QIC-1000*
17	QIC-525*
16	QIC-150
15	QIC-120
0	Default (QIC-1000), or whatever was the last density setting by a using system.

* QIC-525 and QIC-1000 are the only modes that support the 1024 block size.

The default values are 21 for Density Setting #1, and 17 for Density Setting #2.

Attributes with Fixed Values
If a tape drive is configured as a 1200 MB 1/4-inch tape drive, the Extended File Marks, Reserve Support, Variable Length Block Size, and Data Compression attributes have predefined values which cannot be changed.

Attributes for 12000 MB 4 mm Tape Drives (Self Configuring)

Block Size
The IBM 12000 MB 4 mm Tape Drive's throughput is sensitive to blocksize. The minimum recommended blocksize for this drive is 32 K Bytes. Any block size less than 32 K Bytes restricts the data rate (backup/restore time). The following table lists recommended block sizes by command:

Supported Command	Default Block Size (Bytes)	RECOMMENDATION
BACKUP	32 K or 51.2 K (default)	Will use either 32 K or 51.2 K depending on if ″Backup″ is by name or not. No change is required.
TAR	10 K	There is an error in the manual that states a 512 K byte block size. Set the Blocking Parameter to **-N64**.
MKSYSB	See BACKUP	MKSYSB uses the BACKUP Command. No change is required.
DD	n/a	Set the Blocking Parameter to **bs=32K**.
CPIO	n/a	Set the Blocking Parameter to **-C64**.

> **Note:** You should be aware of the capacity and throughput when you select a blocksize. Small blocksizes have a significant impact on performance and a minimal impact on capacity.

Device Buffers
The general information for this attribute applies to this tape drive type.

Extended File Marks
The general information for this attribute applies to this tape drive type.

Density Setting #1 and Density Setting #2
The following chart shows the Supported Data Cartridge type and Density Settings (in decimal and hex) for the IBM 12000 MB 4 mm Tape Drive. When you perform a Restore (Read) Operation, the tape drive automatically sets the density to match the written density. When you perform a Backup Operation (Write),

you must set the Density Setting to match the Data Cartridge you are using.

Supported Data Cartridges	Native Capacity	Compressed Data Capacity	Web-based System Manager or SMIT Density Setting	HEX Density Setting
DDS III	2.0 GB	4.0 GB	19	13h
DDS2	4.0 GB	8.0 GB	36	24h
DDS3	12.0 GB	24.0 GB	37	25h

Note: If you request an unsupported Native Capacity for the Data Cartridge, the drive defaults to the highest supported capacity for the Data Cartridge that is loaded into the drive.

Data Compression
The actual compression depends on the type of data being that is being written. (see above table) A Compression Ratio of 2:1 is assumed for this Compressed Data Capacity.

Attributes with Fixed Values
The general information for this attribute applies to this tape drive type.

Attributes for 13000 MB 1/4-Inch Tape Drives (Self configuring)

Block Size
The default block size is 512. The other valid block sizes are 0 for variable length blocks, and 1024.

Device Buffers
The general information for this attribute applies to this tape drive type.

Extended File Marks
Writing to a 1/4-inch tape can only occur at the beginning of tape (BOT) or after blank tape is detected. If data exists on the tape, you cannot overwrite the data except at BOT. If you wish to add data to a tape that has been written and then rewound, you must space forward until the next file mark is detected, which causes the system to return an error. Only then can you start writing again.

Retension
The general information for this attribute applies to this tape drive type.

Density Setting #1 and Density Setting #2
The following settings apply:

Setting	Meaning
33	QIC-5010-DC*
34	QIC-2GB*
21	QIC-1000*
17	QIC-525*
16	QIC-150
15	QIC-120
0	Default (QIC-5010-DC)*

* QIC-525, QIC-1000, QIC-5010-DC, and QIC-2GB are the only modes that support the 1024 block size.

The default values are 33 for Density Setting #1, and 34 for Density Setting #2.

Attributes with Fixed Values
If a tape drive is configured as a 13000 MB 1/4-inch tape drive, the Extended File Marks, Reserve Support, and Variable Length Block Size attributes have predefined values which cannot be changed.

Attributes for 1/2-Inch 9-Track Tape Drives (Type 9trk)

Block Size
The default block size is 1024.

Device Buffers
The general information for this attribute applies to this tape drive type.

Density Setting #1 and Density Setting #2
The following settings apply:

Setting	Meaning
3	6250 bits per inch (bpi)
2	1600 bpi
0	Whichever writing density was used previously.

The default values are 3 for Density Setting #1, and 2 for Density Setting #2.

Attributes with Fixed Values
If a tape drive is configured as a 1/2-inch 9-track tape drive, the Extended File Marks, Retension, Reserve Support, Variable Length Block Size, and Data Compression attributes have predefined values which cannot be changed.

Attributes for 3490e 1/2-Inch Cartridge (Type 3490e)

Block Size
The default block size is 1024. This drive features a high data transfer rate, and block size can be critical to efficient operation. Larger block sizes can greatly improve operational speeds, and in general, the largest possible block size should be used.

> **Note:** Increasing the block value can cause incompatibilities with other programs on your system. If this occurs, you receive the following error message while running those programs:
>
> `A system call received a parameter that is not valid.`

Device Buffers
The general information for this attribute applies to this tape drive type.

Compression
The general information for this attribute applies to this tape drive type.

Autoloader
This drive features a tape sequencer, an autoloader that sequentially loads and ejects a series of tape cartridges from the cartridge loader. For this function to operate correctly, the front panel switch should be in the AUTO position and the Autoloader attribute must be set to `yes`.

Attributes for Other SCSI Tapes (Type ost)

Block Size
The system default is 512, but this should be adjusted to the default block size for your tape drive. Typical values are 512 and 1024. 8 mm and 4 mm tape drives usually use 1024 and waste space on the tape if the block size attribute is left at 512. A value of 0 indicates variable block size on some drives.

Device Buffers
The general information for this attribute applies to this tape drive type.

Extended File Marks
The general information for this attribute applies to this tape drive type.

Density Setting #1 and Density Setting #2
The default value is 0 for both of these settings. Other values and their meanings vary for different tape drives.

Reserve Support
The default value is no. This can be set to yes, if the drive supports reserve/release commands. If you are unsure, no is a safer value.

Variable Length Block Size
The default variable length block size value is 0. Nonzero values are used primarily on quarter inch cartridge (QIC) drives. See the SCSI specification for the particular tape drive for advice.

Retry Delay
This attribute applies exclusively to type ost tape drives

Read/Write Timeout
This attribute applies exclusively to type ost tape drives

Attributes with Fixed Values
If a tape drive is configured as an Other SCSI tape drive, the attributes for Extended File Marks, Retension, and Data Compression have predefined values which cannot be changed.

Special Files for Tape Drives

Writing to and reading from files on tapes is done by using **rmt** special files. There are several special files associated with each tape drive known to the operating system. These special files are **/dev/rmt***, **/dev/rmt*.1**, **/dev/rmt*.2**, through **/dev/rmt*.7**. The **rmt*** is the logical name of a tape drive, such as **rmt0**, **rmt1**, and so on.

By selecting one of the special files associated with a tape drive, you make choices about how the I/O operations related to the tape drive will be performed.

Density — You can select whether to write with the tape drive Density Setting #1 or with the tape drive Density Setting #2. The values for these density settings are part of the attributes of the tape drive. Because it is customary to set Density Setting #1 to the highest possible density for the tape drive and Density Setting #2 to the next highest possible density for the tape drive, special files that use Density Setting #1 are sometimes referred to as high density and special files that use Density Setting #2 sometimes are referred to as low density, but this view is not always correct. When reading from a tape, the density setting is ignored.

Rewind-on-Close — You can select whether the tape is rewound when the special file referring to the tape drive is closed. If rewind-on-close is selected, the tape is positioned at the beginning of the tape when the file is closed.

Retension-on-Open — You can select whether the tape is retensioned when the file is opened. Retensioning means winding to the end of the tape and then rewinding to the beginning of the tape to reduce errors. If retension-on-open is selected, the tape is positioned at the beginning of the tape as part of the open process.

The following table shows the names of the rmt special files and their characteristics.

Special File	Rewind on Close	Retension on Open	Density Setting
/dev/rmt*	Yes	No	#1

Special File	Rewind on Close	Retension on Open	Density Setting
/dev/rmt*.1	No	No	#1
/dev/rmt*.2	Yes	Yes	#1
/dev/rmt*.3	No	Yes	#1
/dev/rmt*.4	Yes	No	#2
/dev/rmt*.5	No	No	#2
/dev/rmt*.6	Yes	Yes	#2
/dev/rmt*.7	No	Yes	#2

Suppose you want to write three files on the tape in tape drive rmt2. The first file is to be at the beginning of the tape, the second file after the first file, and the third file after the second file. Further, suppose you want Density Setting #1 for the tape drive. The following list of special files, in the order given, could be used for writing the tape.

1. `/dev/rmt2.3`
2. `/dev/rmt2.1`
3. `/dev/rmt2`

These particular special files are chosen because:

- **/dev/rmt2.3** is chosen as the first file because this file has Retension-on-Open, which ensures that the first file is at the beginning of the tape. Rewind-on-Close is not chosen because the next I/O operation is to begin where this file ends. If the tape is already at the beginning when the first file is opened, using the **/dev/rmt2.1** file as the first file would be faster since time for retensioning the tape is eliminated.
- **/dev/rmt2.1** is chosen for the second file because this file has neither Retension-on-Open nor Rewind-on-Close chosen. There is no reason to go to the beginning of the tape either when the file is opened or when it is closed.
- **/dev/rmt2** is chosen for the third and final file because Retension-on-Open is not wanted since the third file is to follow the second file. Rewind-on-Close is selected because there are no plans to do any more writing after the third file on the tape. The next use of the tape will begin at the beginning of the tape.

Besides controlling tape operations by choosing a particular rmt special file, you can use the **tctl** command to control tape operations.

Appendix A. Comparisons for BSD System Managers

This appendix is for system administrators who are familiar with 4.3 BSD UNIX or System V operating systems. This information explains the differences and the similarities between those systems and AIX.

Topics discussed in this appendix are:

Comparisions Between AIX and BSD for System Managers

The following articles provide information for 4.3 BSD administrators:

Introduction to AIX for BSD System Managers

The following tips can help you get started managing the system:

- Start by logging in as root at the graphics console.
- Perform system management from the system console until you become experienced with the system. It is easier to work from the system console than a remote terminal. Once you are experienced with the system, you can work remotely from an xterm or an ASCII terminal.
- Take advantage of the several AIX facilities for system management tasks. They include:
  – System Management Interface Tool (SMIT). SMIT provides an interface between system managers and configuration and management commands. SMIT can help system managers perform most system administration tasks. For more information, see the Chapter 13, “System Management Interface Tool,” on page 121.
  – The Object Data Manager (ODM). The ODM provides routines that access objects from the ODM databases. The ODM databases contain device configuration information. For more information about how the ODM databases store device information, see Chapter 15, “Devices,” on page 125.
  – The System Resource Controller (SRC). The SRC provides access and control of daemons and other system resources through a single interface. For more information, see the “System Resource Controller Overview” on page 109.

Major Differences between 4.3 BSD and this Operating System

This article summarizes the major differences between this operating system and 4.3 BSD systems. For more detailed discussions of these topics, see the list of articles in “Comparisions Between AIX and BSD for System Managers” on page 151.

Configuration Data Storage

4.3 BSD usually stores configuration data in ASCII files. Related pieces of information are kept on the same line and record processing (sorting and searching) can be done on the ASCII file itself. Records can vary in length and are terminated by a line feed. 4.3 BSD provides tools to convert some potentially large ASCII files to a database (dbm) format. Relevant library functions search the pair of dbm files if they exist, but search the original ASCII file if the dbm files are not found.

Some configuration data for this operating system is stored in ASCII files, but often in a *stanza* format. A stanza is a set of related pieces of information stored in a group of several lines. Each piece of information has a label to make the contents of the file more understandable.

This operating system also supports dbm versions of password and user information. Furthermore, the **/etc/passwd**, **/etc/group**, and **/etc/inittab** files are examples of files for this operating system where the information is stored in traditional form rather than in stanza form.

Other configuration data for this operating system are stored in files maintained by the Object Data Manager (ODM). Web-based System Manager or the System Management Interface Tool (SMIT) can manipulate and display information in ODM files. Alternatively, you can use the ODM commands directly to view these files. To query the ODM files, use the following commands:

- **odmget**

- **odmshow**.

The following ODM commands alter ODM files:
- **odmadd**
- **odmcreate**
- **odmdrop**
- **odmchange**
- **odmdelete**.

> **Attention:** Altering ODM files incorrectly can cause the system to fail, and might prevent you from successfully restarting the system. Only use ODM commands directly on ODM files when task-specific commands, such as those generated by Web-based System Manager or SMIT, are unsuccessful.

Configuration Management

When a system running this operating system starts up, a set of configuration-specific commands are invoked by the Configuration Manager. These configuration-specific commands are called *methods*. Methods identify the devices on the system and update the appropriate ODM files in the **/etc/objrepos** directory.

Device special files in the **/dev** directly are not preinstalled. Some special files, such as those for hard disks, are created automatically during the startup configuration process. Other special files, such as those for ASCII terminals, must be created by the system administrator by using the Web-based System Manager Devices application or the SMIT Devices menu. This information is retained in the ODM for later use by the system.

Disk Management

In this operating system, disk drives are referred to as *physical volumes*. Partitions are referred to as *logical volumes*. As in 4.3 BSD, a single physical volume can have multiple logical volumes. However, unlike 4.3 BSD, a single volume in this operating system can span multiple physical volumes. To do this, you must make several physical volumes into a *volume group* and create logical volumes on the volume group.

Commands in this operating system used for file system and volume management include:
- **crfs**
- **varyonvg**
- **varyoffvg**
- **lsvg**
- **importvg**
- **exportvg**.

The following 4.3 BSD commands are also available:
- **mkfs**
- **fsck**
- **fsdb**
- **mount**
- **umount**.

Differences between these commands for 4.3 BSD and for this operating system of are discussed in “File Systems for BSD 4.3 System Managers” on page 161.

4.3 BSD maintains a list of file systems in the **/etc/fstab** file. This operating system maintains a stanza for each file system in the **/etc/filesystems** file.

New Commands

To handle new configuration and disk management systems, this operating system has about 150 commands that are new to 4.3 BSD administrators. For more information, see “Commands for System Administration for BSD 4.3 System Managers” on page 157.

Startup

This operating system supports automatic identification and configuration of devices. Consequently, the startup process is very different from 4.3 BSD systems. In addition to the kernel, an image of a boot file system and the previous base device configuration information is loaded to a RAM disk. In the first phase of startup, sufficient configuration information is loaded and checked to permit accessing logical volumes. The paging space device is identified to the kernel and the hard disk root file system is checked. At this time, the operating system changes the root file system from the RAM disk to the hard disk and completes the startup procedure, including configuring other devices.

User Authorization

4.3 BSD, and versions of AT&T UNIX operating systems before SVR4, store all user authentication information, including encrypted passwords, in the **/etc/passwd** file. Traditionally, the **/etc/passwd** file could be read by all.

On SVR4 systems, encrypted passwords are removed from the **/etc/passwd** file and stored in the **/etc/shadow** file. Only users with root authority and trusted programs (such as the **/bin/login** program) can read the **/etc/shadow** file.

This operating system stores encrypted passwords in the **/etc/security/passwd** file. Other files in the **/etc/security** directory are the **user** and **limits** files. These three files define the way a user is allowed to access the system (such as using the **rlogin** or **telnet** commands) and the user’s resource limits (such as file size and address space).

Printing

Most 4.3 BSD printing commands are supported with minor differences. One difference is that the **/etc/qconfig** file is the configuration file in this operating system.

The line printing system for this operating system can interoperate with the 4.3 BSD line printing system, both for submitting print jobs to 4.3 BSD systems and for printing jobs submitted from a 4.3 BSD system.

Shells

This operating system supports the Bourne shell, C shell and Korn shell. The full path name for the Bourne shell program is **/bin/bsh**. The **/bin/sh** file is a hard link to the **/bin/ksh** file. This file can be changed by the administrator.

AIX does not support **setuid** or **setgid** for shell scripts in any shell.

Notes:

1. This operating system has no shell scripts that rely on the **/bin/sh**. However, many shell scripts from other systems rely on **/bin/sh** being the Bourne shell.
2. Although the Bourne shell and Korn shell are similar, the Korn shell is not a perfect superset of the Bourne shell.

Accounting for BSD 4.3 System Managers

The accounting files in the **/usr/lib/acct** directory and the system activity reporting tools in the **/usr/lib/sa** directory for this operating system are identical to those available with AT&T System V Release 4 (SVR4) with the addition of 4.3 BSD accounting utilities.

Many of the accounting commands are in the **/usr/lib/acct** directory. To begin system accounting, use the**/usr/lib/acct/startup** command. If accounting is not started, commands such as **lastcomm**(1) cannot return information.

This operating system provides these 4.3 BSD accounting facilities:

last(1)	Indicates last logins of users and terminals
lastcomm (1)	Shows in reverse order the last commands executed
acct(3)	Enables and disables process accounting
ac(8)	Login accounting
accton (8)	Turns system accounting on or off
sa(8)	Generally maintains system accounting files.

This operating system also provides these System V Interface Definition (SVID) Issue II accounting commands and library functions:

acctcms (1)	Produces command usage summaries from accounting records
acctcom (1)	Displays selected process-accounting record summaries
acctcon1 (1)	Converts login/logoff records to session records
acctcon2 (1)	Converts login/logoff records to total accounting records
acctdisk (1)	Generates total accounting records from **diskusg**(1) command output
acctmerg (1)	Merges total accounting files into an intermediary file
accton (1)	Turns on accounting
acctprc1 (1)	Processes accounting information from **acct**(3) command
acctprc2 (1)	Processes output of **acctprc1**(1) command into total accounting records
acctwtmp (1)	Manipulates connect-time accounting records
chargefee(1)	Charges to login name
ckpacct(1)	Checks size of **/usr/adm/pacct** file
diskusg (1)	Generates disk accounting information
dodisk (1)	Performs disk accounting
fwtmp(1)	Converts binary records (**wtmp** file) to formatted ASCII.
	Note: The **wtmp** file is in the **/var/adm** directory
lastlogin(1)	Updates last date on which each person logged in
monacct (1)	Creates monthly summary files
prctmp (1)	Prints session record file produced by **acctcon1**(1) command
prdaily (1)	Formats a report of yesterday's accounting information
prtacct (1)	Formats and prints any total accounting file
runacct (1)	Runs daily accounting
shutacct (1)	Called by system shutdown to stop accounting and log the reason
startup (1)	Called by system initialization to start accounting
turnacct (1)	Turns process accounting on or off
wtmpfix(1)	Corrects time/date stamps in a file using **wtmp** format.

Backup for BSD 4.3 System Managers

The **tar** and **cpio** commands can move data between systems. The **tar** command for this operating system is not fully compatible with the 4.3 BSD **tar** command. The **tar** command for this operating system requires the **-B** flag (blocking input) if it is reading from a pipe. The AT&T **cpio** command is compatible with this version.

This operating system can read and write in **dump** and **restore** command format. For example, the **backup** command for this operating system with the syntax:

```
backup -0uf Device Filesystemname
```

is the same as the 4.3 BSD **dump** command with the syntax:

```
dump 0uf Device Filesystemname
```

Similarly, the **restore** command for this operating sytem with the syntax:

```
restore -mivf Device
```

is the same as the 4.3 BSD **restore** command with the syntax:

```
restore ivf Device
```

This operating system also has the 4.3 BSD **rdump** and **rrestore** commands. The only difference in the two versions is that for this operating system each argument must be preceded by a **-** (dash). For example, the following command:

```
rdump -0 -f orca:/dev/rmt0 /dev/hd2
```

is equivalent to the 4.3 BSD command:

```
rdump 0f orca:/dev/rmt0 /dev/hd2
```

The **backup** command for this operating system with the following syntax:

```
backup -0f /dev/rmt0 /dev/hd2
```

is equivalent to the 4.3 BSD **dump** command with this syntax:

```
dump 0f /dev/rmt0 /dev/hd2
```

Non-IBM SCSI Tape Support

This operating system does not directly support non-IBM SCSI tape drives. However, you can add your own header and interface that use the IBM SCSI driver. For more information, see the information on adding an unsupported device to the system in *AIX 5L Version 5.3 Kernel Extensions and Device Support Programming Concepts* and “Backup Overview” on page 59.

Startup for BSD 4.3 System Managers

On 4.3 BSD systems, the **init** program is the last step in the startup procedure. The main role of the **init** program is to create processes for each available terminal port. The available terminal ports are found by reading the **/etc/ttys** file.

On System V, the **init** program is started at system initialization. The **init** process starts processes according to entries in the **/etc/inittab** file.

This operating system follows the System V initialization procedure. You can edit the **/etc/inittab** file by directly editing the file, using the **telinit** command, or by using the following commands:

chitab(1) Changes records in the **/etc/inittab** file

lsitab(1)	Lists records in the **/etc/inittab** file
mkitab(1)	Makes records in the **/etc/inittab** file
rmitab(1)	Removes records in the **/etc/inittab** file.

Changes made to the **/etc/inittab** file take effect the next time the system is rebooted, or when the **telinit q** command is run.

Commands for System Administration for BSD 4.3 System Managers

This list contains commands that are specifically for administering the environment for this operating system.

bosboot (1)	Initializes a boot device.
bootlist (1)	Alters the list of boot devices (or the ordering of these devices in the list) available to the system.
cfgmgr(1)	Configures devices by running the programs in **/etc/methods** directory.
chcons (1)	Redirects the system console to device or file, effective next startup
chdev(1)	Changes the characteristics of a device
chdisp (1)	Changes the display used by the low-function terminal (LFT) subsystem.
checkcw(1)	Prepares constant-width text for the **troff** command.
checkeq(1)	Checks documents formatted with memorandum macros.
checkmm(1)	Checks documents formatted with memorandum macros.
checknr (1)	Checks **nroff** and **troff** files.
chfont(1)	Changes the default font selected at boot time.
chfs(1)	Changes attributes of a file system.
chgroup (1)	Changes attributes for groups.
chgrpmem (1)	Changes the administrators or members of a group.
chhwkbd (1)	Changes the low function terminal (LFT) keyboard attributes stored in the Object Data Manager (ODM) database.
chitab(1)	Changes records in the **/etc/inittab** file.
chkbd(1)	Changes the default keyboard map used by the low-function terminal (LFT) at system startup.
chkey(1)	Changes your encryption key.
chlang	Sets **LANG** environment variable in the **/etc/environment** file for the next login.
chlicense (1)	There are two types of user licensing, fixed and floating. Fixed licensing is always enabled, and the number of licenses can be changed through the **-u** flag. Floating licensing can be enabled or disabled (on or off) through the **-f** flag
chlv(1)	Changes the characteristics of a logical volume
chnamsv (1)	Changes TCP/IP-based name service configuration on a host
chprtsv (1)	Changes a print service configuration on a client or server machine
chps(1)	Changes attributes of a paging space.
chpv(1)	Changes the characteristics of a physical volume in a volume group.
chque(1)	Changes the queue name.
chquedev (1)	Changes the printer or plotter queue device names.
chssys (1)	Changes a subsystem definition in the subsystem object class.
chtcb(1)	Changes or queries the trusted computing base attribute of a file.
chtz	Changes the system time zone information.
chuser(1)	Changes attributes for the specified user.
chvfs(1)	Changes entries in the **/etc/vfs** file.
chvg(1)	Sets the characteristics of a volume group.
chvirprt (1)	Changes the attribute values of a virtual printer.
crfs (1)	Adds a file system.
crvfs(1)	Creates entries in the **/etc/vfs** file.
exportvg (1)	Exports the definition of a volume group from a set of physical volumes.
extendvg (1)	Adds physical volumes to a volume group.

grpck(1)	Verifies the correctness of a group definition.
importvg(1)	Imports a new volume group definition from a set of physical volumes.
lsallq (1)	Lists the names of all configured queues.
lsallqdev (1)	Lists all configured printer and plotter queue device names within a specified queue.
lsdisp(1)	Lists the displays currently available on the system.
lsfont(1)	Lists the fonts available for use by the display.
lsfs(1)	Displays the characteristics of file systems.
lsgroup(1)	Displays the attributes of groups.
lsitab(1)	Lists the records in the **/etc/inittab** file.
lskbd(1)	Lists the keyboard maps currently available to the low-function terminal (LFT) subsystem.
lslicense (1)	Displays the number of fixed licenses and the status of floating licensing.
lslpp (1)	Lists optional program products.
lsnamsv (1)	Shows name service information stored in the database.
lsprtsv (1)	Shows print service information stored in the database.
lsps	Lists paging space and attributes.
lsque(1)	Displays the queue stanza name.
lsquedev (1)	Displays the device stanza name.
lssrc(1)	Gets the status of a subsystem, a group of subsystems, or a subserver.
lsuser(1)	Displays attributes of user accounts.
lsvfs(1)	Lists entries in the **/etc/vfs** file.
mkcatdefs (1)	Preprocesses a message source file.
runcat(1)	Pipes the output data from the **mkcatdefs** command to the **gencat** command.
mkdev(1)	Adds a device to the system.
mkfont (1)	Adds the font code associated with a display to the system.
mkfontdir (1)	Creates a **fonts.dir** file from a directory of font files.
mkgroup (1)	Creates a new group.
mkitab(1)	Makes records in the **/etc/inittab** file.
mklv(1)	Creates a logical volume.
mklvcopy (1)	Adds copies to a logical volume.
mknamsv (1)	Configures TCP/IP-based name service on a host for a client.
mknotify (1)	Adds a notify method definition to the notify object class.
mkprtsv (1)	Configures TCP/IP-based print service on a host.
mkps(1)	Add an additional paging space to the system.
mkque(1)	Adds a printer queue to the system.
mkquedev (1)	Adds a printer queue device to the system.
mkserver (1)	Adds a subserver definition to the subserver object class.
mkssys(1)	Adds a subsystem definition to the subsystem object class.
mksysb	Backs up mounted file systems in the rootvg volume group for subsequent reinstallation.
mkszfile	Records size of mounted file systems in the rootvg volume group for reinstallation.
mktcpip (1)	Sets the required values for starting TCP/IP on a host.
mkuser (1)	Creates a new user account.
mkuser.sys (1)	Customizes a new user account.
mkvg(1)	Creates a volume group.
mkvirprt (1)	Makes a virtual printer.
odmadd(1)	Adds objects to created object classes.
odmchange (1)	Changes the contents of a selected object in the specified object class.
odmcreate (1)	Produces the **.c** (source) and **.h** (include) files necessary for ODM application development and creates empty object classes.
odmdelete (1)	Deletes selected objects from a specified object class.
odmdrop (1)	Removes an object class.
odmget(1)	Retrieves objects from the specified object classes and places them into an **odmadd** input file.
odmshow (1)	Displays an object class definition on the screen.
pwdck(1)	Verifies the correctness of local authentication information.

redefinevg	Redefines the set of physical volumes of the given volume group in the device configuration database.
reducevg (1)	Removes physical volumes from a volume group. When all physical volumes are removed from the volume group, the volume group is deleted.
reorgvg (1)	Reorganizes the physical partition allocation for a volume group.
restbase (1)	Restores customized information from the boot image.
rmdel(1)	Removes a delta from a Source Code Control System (SCCS) file.
rmdev(1)	Removes a device from the system.
rmf(1)	Removes folders and the messages they contain.
rmfs(1)	Removes a file system.
rmgroup(1)	Removes a group.
rmitab(1)	Removes records in the **/etc/inittab** file.
rmlv(1)	Removes logical volumes from a volume group.
rmlvcopy (1)	Removes copies from a logical volume.
rmm(1)	Removes messages.
rmnamsv (1)	Unconfigures TCP/IP-based name service on a host.
rmnotify (1)	Removes a notify method definition from the notify object class.
rmprtsv (1)	Unconfigures a print service on a client or server machine.
rmps(1)	Removes a paging space from the system.
rmque(1)	Removes a printer queue from the system.
rmquedev (1)	Removes a printer or plotter queue device from the system.
rmserver (1)	Removes a subserver definition from the subserver object class.
rmssys(1)	Removes a subsystem definition from the subsystem object class.
rmuser(1)	Removes a user account.
rmvfs(1)	Removes entries in the **/etc/vfs** file.
rmvirprt (1)	Removes a virtual printer.
savebase (1)	Saves base customized device data in the ODM onto the boot device.
swapoff(1)	Deactives one or more paging space.
swapon(1)	Specifies additional devices for paging and swapping.
syncvg (1)	Synchronizes logical volume copies that are not current.
usrck(1)	Verifies the correctness of a user definition.
varyoffvg(1)	Deactivates a volume group.
varyonvg (1)	Activates a volume group.

Cron for BSD 4.3 System Managers

The **cron** daemon for this operating system is similar to the System V Release 2 **cron** daemon. An entry in the **/etc/inittab** file starts the **cron** daemon.

Devices for BSD 4.3 System Managers

A device on a 4.3 BSD system is accessible to an application only when:

- The device is physically installed and functioning.
- The driver for the device is in the kernel.
- The device special files for the device exist in the **/dev** directory.

A device on this operating system is accessible to an application only when:

- The device is physically installed and functioning.
- The driver for the device is in the kernel or in a loaded kernel extension.
- The device special files for the device exist in the **/dev** directory.
- The object database in the **/etc/objrepos** directory contains entries for the device that match the physical configuration.

The device specific programs called *methods*, found in the **/etc/methods** directory, maintain the object database. The methods are invoked by the Configuration Manager (accessed through the **cfgmgr** command) and other commands.

If a device can no longer be accessed by an application program, it can mean that the hardware is faulty or it can mean that the configuration database in the **/etc/objrepos** directory is damaged.

The **cfgmgr** command processes the configuration database in the **/etc/objrepos** directory and is processed at startup time by the **cfgmgr** command (the Configuration Manager).

The pseudocode below shows the Configuration Manager logic:

```
/* Main */
While there are rules in the Config_Rules database
        {
        Get the next rule and execute it
        Capture stdout from the last execution
        Parse_Output(stdout)
        }
/* Parse Output Routine */
/* stdout will contain a list of devices found */
Parse_OutPut(stdout)
        {
        While there are devices left in the list
                {
                Lookup the device in the database
                if (!defined)
                        Get define method from database and execute
                if (! configured)
                        {
                        Get config method from database and execute
                        Parse_Output(stdout)
                         }
                }
        }
```

File Comparison Table for 4.3 BSD, SVR4, and this Operating System

The following table compares file names and functions between 4.3 BSD, SVR4, and this operating system.

File Comparison Table				
4.3 BSD File	**SVR4 File**	**File for this operating system**	**Database**	**Type (odm/dbm)**
L-Devices	Devices	Devices	no	
L-dialcodes	Dialcodes	Dialcodes	no	
L.cmds	Permissions	Permissions	no	
L.sys	Systems	Systems	no	
USERFILE	Permissions	Permissions	no	
aliases	mail/namefiles	aliases	aliasesDB/DB	dbm
fstab	vfstab	filesystems	no	
ftpusers	ftpusers	ftpusers	no	
gettytab		N/A		
group	group	group	no	
hosts	hosts	hosts	no	

File Comparison Table				
4.3 BSD File	**SVR4 File**	**File for this operating system**	**Database**	**Type (odm/dbm)**
hosts.equiv	hosts.equiv	hosts.equiv	no	
inetd.conf	inetd.conf	inetd.conf	no	
map3270	N/A	map3270	no	
motd	motd	motd	no	
mtab	mnttab	N/A	no	
named.boot	named.boot	named.boot	no	
named.ca		named.ca	no	
named.hosts		named.data (See note)	no	
named.local		named.local	no	
named.pid	named.pid	named.pid	no	
named.rev		named.rev	no	
networks	networks	networks	no	
passwd	passwd	passwd	no	
printcap	qconfig	qconfig		
protocols		protocols	no	
remote	remote	remote	no	
resolv.conf	resolv.conf	resolv.conf	no	
sendmail.cf	sendmail.cf	sendmail.cf	sendmail.cfDB	neither
services		services	no	
shells	shells	N/A		
stab		N/A		
syslog.conf		syslog.conf	no	
syslog.pid		syslog.pid	no	
termcap	terminfo	terminfo		
ttys	ttys	N/A	yes	odm
types		N/A		
utmp	utmp	utmp		
vfont		N/A		
vgrindefs		vgrindefs		
wtmp	wtmp	wtmp		

Note: The file names **named.ca**, **named.hosts**, **named.local**, and **named.rev** are user definable in the **named.boot** file. However, these are the names used for these files in the documentation for this operating system.

File Systems for BSD 4.3 System Managers

This information offers a brief comparison of file systems for this operating system to other system file systems, and an outline of the supported file system types on this operating system.

This operating system uses the **/etc/filesystem** file to list file system device information, and has similar commands for mounting and unmounting file systems.

/etc/filesystems File and /etc/fstab File

4.3 BSD systems store lists of block devices and mount points in the **/etc/fstab** file.

SVR4 systems stores block devices and mount point information in **/etc/vfstab** file.

This operating system stores block device and mount points information in **/etc/filesystems** file. The **crfs**, **chfs**, and **rmfs** commands update the **/etc/filesystems** file.

4.3 BSD system administrators might be interested in the *check* variable in the **/etc/filesystems** file. The *check* variable can be set to the value `True`, `False` or to a number. For example, you can specify `check=2` in the **/etc/filesystems** file. The number specifies the pass of the **fsck** command that will check this file system. The **check** parameter corresponds to the fifth field in an **/etc/fstab** file record.

There is no dump frequency parameter in the **/etc/filesystems** file.

File System Support on this operating system

This operating system supports disk quotas.

This operating system does not allow mounting of diskettes as file systems.

The syntax of the**mount** and **umount** commands for this operating system differs from 4.3 BSD and from SVR4 versions of these commands. The commands to mount and unmount all file systems at once are shown for all three systems in the following table:

mount *and* **unmount** *Commands*

Function	Syntax for this operating system	4.3 BSD Syntax	SVR4 Syntax
mount all file systems	**mount all**	**mount -a**	**mountall**
unmount all file systems	**umount all**	**umount -a**	**umountall**

See the Chapter 5, “File Systems,” on page 35 for more information.

Finding and Examining Files for BSD 4.3 System Managers

This operating system supports the following 4.3 BSD file commands:

- **which**
- **whereis**
- **what**
- **file**.

This operating system does not support the 4.3 BSD **fast find** syntax of the **find** command. At this time, there is no replacement function. The following **ffind** shell script can be used to simulate the functionality:

```
#!/bin/bsh
PATH=/bin
for dir in /bin /etc /lib /usr
do
find $dir -print | egrep $1
done
```

The syntax for the **ffind** script is:

`ffind Filename`

Paging Space for BSD 4.3 System Managers

The following commands assist in managing paging space (also known as swap space):

chps(1)	Changes attributes of a paging space
lsps(1)	List attributes of a paging space
mkps(1)	Add an additional paging space to the system
rmps(1)	Removes a paging space from the system
swapoff(1)	Deactivates one or more paging spaces
swapon(1)	Specifies additional devices for paging and swapping

If a large paging space is required, place one paging logical volume for each hard disk. This allows scheduling of paging across multiple disk drives.

Networking for BSD 4.3 System Managers

This article describes how to use 4.3 BSD ASCII network configurations, additional commands and command options, and name and address resolution for this operating system, as well as the differences between 4.3 BSD network management and network management for this operating system.

How to Change Default Startup to Permit 4.3 BSD ASCII Configuration

You can administer network interfaces for this operating system through the SMIT and ODM files, or through 4.3 BSD ASCII configuration files.

To administer network interfaces through 4.3 BSD ASCII configuration files, uncomment the commands in the **/etc/rc.net** file below the heading:

```
# Part II - Traditional
Configuration
```

Then if you want flat file configuration and SRC support, edit the **/etc/rc.net** file and uncomment the **hostname**, **ifconfig**, and **route** commands with the appropriate parameters.

If you want flat file configuration without SRC support, use the **smit setbootup_option** fast path to change the system to BSD-style **rc** configuration. This option configures the system to use the **/etc/rc.bsdnet** file at startup. You also have to edit the **/etc/rc.bsdnet** file and uncomment the **hostname**, **ifconfig**, and **route** commands with the appropriate parameters.

Additional Options for ifconfig and netstat Commands

The **ifconfig** command for this operaitng system has the following additional options:

mtu The *mtu* variable specifies the maximum transmission unit (MTU) used on the local network (and local subnets) and the MTU used for remote networks. To maximize compatibility with Ethernet and other networks, set both the Token-Ring and Ethernet default *mtu* value to 1500.

allcast
The **allcast** flag sets the Token-Ring broadcast strategy. Setting the **allcast** flag optimizes connectivity through Token-Ring bridges. Clearing the **allcast** flag (by specifying **-allcast**) minimizes excess traffic on the ring.

The **netstat** command for this operating system has the **-v** flag. The **netstat -v** command prints driver statistics such as transmit byte count, transmit error count, receive byte count, and receive error count.

Additional Network Management Commands

The following additional commands are supported on this operating system:

securetcpip The **securetcpip** shell script enables controlled access mode, which provides enhanced network security. It disallows the running of several unsecured TCP/IP programs, such as the **tftp**, **rcp**, **rlogin**, and **rsh** programs. It also restricts the use of the **.netrc** file

gated The **gated** command provides MIB support for SNMP

no The **no** command sets network options that include:

dogticks
Sets timer granularity for **ifwatchdog** routines

subnetsarelocal
Determines if packet address is on the local network

ipsendredirects
Specifies whether the kernel should send redirect signals

ipforwarding
Specifies whether the kernel should forward packets

tcp_ttl Specifies the time-to-live for Transmission Control Protocol (TCP) packets

udp_ttl Specifies the time-to-live for User Datagram Protocol (UDP) packets

maxttl Specifies the time-to-live for Routing Information Protocol (RIP) packets

ipfragttl
Specifies the time-to-live for Internet Protocol (IP) fragments

lowclust
Specifies a low water mark for cluster **mbuf** pool

lowmbuf
Specifies a low water mark for the **mbuf** pool

thewall Specifies the maximum amount of memory that is allocated to the **mbuf** and cluster **mbuf** pool

arpt_killc
Specifies the time in minutes before an inactive complete Address Resolution Protocol (ARP) entry is deleted

iptrace The **iptrace** command provides interface-level packet tracing for Internet protocols.

ipreport The **ipreport** command formats the trace into human-readable form. An example of using this command is the following:

```
iptrace -i en0 /tmp/iptrace.log
# kill iptrace daemon
kill `ps ax | grep iptrace | awk '{ print $1 }'`
ipreport /tmp/iptrace.log | more
```

Name and Address Resolution

The **gethostbyname** and **gethostbyaddr** subroutines in the **libc** library provide support for Domain Name Service, Network Information Services (NIS, formerly called Yellow Pages), and the **/etc/hosts** database. If the **/etc/resolv.conf** file exists, the name server is always checked first. If the name is not resolved and NIS is running, NIS is checked. If NIS is not running, the **/etc/hosts** file is checked.

Differences between this operating system and 4.3 BSD

On this operating system, the network daemons are started from the **/etc/rc.tcpip** file, not the**/etc/rc.local** file. The **/etc/rc.tcpip** shell script is invoked from the **/etc/inittab** file, not the **/etc/rc** file.

If the System Resource Controller (SRC) is running, the TCP/IP daemons run under SRC control. If you do not want the TCP/IP daemons running under SRC control, use the **smit setbootup_option** fast path to change the system to BSD-style **rc** configuration.

These network management functions available on 4.3 BSD are supported by this operating system:

- Kernel-level SYSLOG logging facilities
- Access rights for UNIX domain sockets.

The tn3270 Command

The **tn3270** command is a link to the **telnet** command, but it uses the **/etc/map3270** file and the current **TERM** environment variable value to provide 3270 keyboard mappings. Thus, the **tn3270** command operates exactly like the BSD version.

If you want to change the escape sequences from the defaults used by the **tn3270**, **telnet**, or **tn** commands, set the **TNESC** environment variable before starting these commands.

Online Documentation and man Command for BSD 4.3 System Managers

This operating system supports the **man-k**, **apropos**, and **whatis** commands, but the database used by these commands must first be created with the **catman-w** command.

The **man** command first searches for flat text pages in the **/usr/man/cat?** files. Next, it searches **nroff**-formatted pages in **/usr/man/man?** files. New man pages can be added in flat text or **nroff** form.

Notes®:

1. The **man** command text pages are not provided with the system. The **catman** command creates the database from these text pages. These pages can be either flat text pages stored in the **/usr/man/cat**? files or **nroff** formatted pages stored the in **/usr/man/man**? files.
2. The Text Formatting licensed program must be installed for the **nroff** command to be available for the **man** command to read **nroff**-formatted man pages.

NFS and NIS (formerly Yellow Pages) for BSD 4.3 System Managers

Network File System (NFS) and Network Information Services (NIS) daemons are started from the **/etc/rc.nfs** file. However, before the NFS and NIS daemons can be started, the **portmap** daemon must be started in the **/etc/rc.tcpip** file. By default, the **/etc/rc.nfs** file is not invoked by the **/etc/inittab** file. If you add a line in the **/etc/inittab** file to invoke the **/etc/rc.nfs** script, it should be invoked after the **/etc/rc.tcpip** script.

If NIS is active, include a root entry prior to the **+**:: (plus sign, colon, colon) entry in the **/etc/passwd** file and a system entry prior to the **+**:: entry in the **/etc/group** file. This allows a system administrator to log in as root and make changes if the system is unable to communicate with the NIS server.

NFS can be configured by using Web-based System Manager (type `wsm`, then select `Network`), or the SMIT fast path, **smit nfs**. The Web-based System Manager and SMIT menus refer to NIS (formerly Yellow Pages) as NIS. Many of the NFS and NIS commands are found in the **/etc** and **/usr/etc** directory.

Some NFS environments use an **arch** command to identify machine families and types of machines. For example if you are using the IBM RS/6000, specify the **power** identifier for family (CPU), and the **ibm6000** identifier for type (machine).

Passwords for BSD 4.3 System Managers

The following information details the differences between managing passwords in this operating system and 4.3 BSD systems.

Setting a User Password

When you use the **/bin/passwd** command for this operating system as the root user, you are prompted for the current root user password. An example of using the **/bin/passwd** command follows:

```
# passwd cslater
Changing password for "cslater"
Enter root's Password or
cslater's Old password:
cslater's New password:
Re-enter cslater's
new password:
#
```

The 4.3 BSD version does not prompt for the current root user password. An example of the 4.3 BSD version follows:

```
# passwd cslater
New password:
Retype new password:
#
```

Importing a 4.3 BSD Password File

You can import a 4.3 BSD password file by first copying it to the **/etc/passwd** file and entering:

```
pwdck -y ALL
```

Then the **/etc/security/limits** file must be updated with a null stanza for any new users. The **usrck** command does this, but using the **usrck** command can cause problems unless the **/etc/group** file is imported with the **/etc/passwd** file.

> **Note:** If the **/etc/security/limits** file is modified, the stack must not exceed 65,536 bytes. If it does, running the **usrck** command can cause problems. Change the stack size to 65,536 and run **usrck** command again.

Also run the **grpck** and **usrck** command to verify group and user attributes.

Editing the Password File

In this operating system, the **lsuser**, **mkuser**, **chuser**, and **rmuser** commands are provided for managing passwords. All of these commands can be used by running Web-based System Manager or SMIT. However, all of these commands deal with only one user at a time.

> **Note:** Using an editor to change several user name entries at one time requires editing of several files simultaneously, because passwords are stored in **/etc/security/passwd** file, authorization information is stored in the **/etc/security/user** file, and the remaining user data is stored in the **/etc/passwd** file.

This operating system does not support the **vipw** command but does support the **mkpasswd** command. However, you can still administer passwords on this operating system in a 4.3 BSD manner. Use the following procedure:

1. Put a 4.3 BSD password file in the **/etc/shadow** file.
2. Change the permissions to the file by entering:

   ```
   chmod 000 /etc/shadow
   ```

3. Place the following **vipw** shell script in the **/etc** directory:

```
-----------------------------------------------------
----
#!/bin/bsh
#
# vipw. Uses pwdck for now. May use usrck someday
#
PATH=/bin:/usr/bin:/etc:/usr/ucb # Add to this if your editor is
                                 # some place else
if [ -f /etc/ptmp ] ; then
                 echo "/etc/ptmp exists. Is someone else using vipw?"
        exit 1
fi
if [ ! -f /`which "$EDITOR" | awk '{ print $1 }'` ] ; then
        EDITOR=vi
fi
cp /etc/shadow /etc/ptmp
if (cmp /etc/shadow /etc/ptmp) ; then
        $EDITOR /etc/ptmp
else
        echo cannot copy shadow to ptmp
        exit 1
fi
if (egrep "^root:" /etc/ptmp >/dev/null) ; then
        cp /etc/ptmp /etc/shadow ; cp /etc/ptmp /etc/passwd
        chmod 000 /etc/passwd /etc/shadow
        pwdck -y ALL 2>1 >/dev/null # return code 114 may change
                rc=$?
        if [ $rc -eq 114 ]; then
                chmod 644 /etc/passwd
                        rm -f /etc/passwd.dir /etc/passwd.pag
                mkpasswd /etc/passwd
                        # update /etc/security/limits, or ftp
                        # will fail
                else
                         pwdck -y ALL
                fi
else
        echo bad entry for root in ptmp
fi
rm /etc/ptmp
-------------------------------------------------------------
```

4. If you use the **vipw** shell script or the **mkpasswd** command, be aware that Web-based System Manager, SMIT, and the **mkuser**, **chuser**, and **rmuser** commands, do not use the **mkpasswd** command. You must run:

   ```
   mkpasswd /etc/passwd
   ```

   to update the **/etc/passwd.dir** and **/etc/passwd.pag** files.

   Attention: Initialization of the *IFS* variable and the **trap** statements guard against some of the common methods used to exploit security holes inherent in the **setuid** feature. However, the **vipw** and **passwd** shell scripts are intended for relatively open environments where compatibility is an important consideration. If you want a more secure environment, use only the standard commands fir this operating system.

5. Put the following passwd shell script in the **/usr/ucb** directory:

```
-----------------------------------------------------
#!/bin/ksh
#
# matches changes to /etc/security/passwd file with changes to
#/etc/shadow
#
IFS=" "
PATH=/bin
trap "exit 2" 1 2 3 4 5 6 7 8 10 12 13 14 15 16 17 18 21 22 \
```

```
        23 24 25 27 28 29 30 31 32 33 34 35 36 60 61 62
if [ -n "$1" ]; then
        USERNAME=$1
else
        USERNAME=$LOGNAME
fi
if [ -f /etc/ptmp ]; then
        echo password file busy
        exit 1
fi
        trap "rm /etc/ptmp; exit 3" 1 2 3 4 5 6 7 8 10 12 13 \
                14 15 16 17 18 21 22 23 24 25 27 28 29 30 31 \
                32 33 34 35 36 60 61 62
if (cp /etc/security/passwd /etc/ptmp) ; then
        chmod 000 /etc/ptmp else
        rm -f /etc/ptmp exit 1
fi
if ( /bin/passwd $USERNAME ) ; then
        PW=` awk ' BEGIN { RS = "" }
                $1 == user { print $4 } ' user="$USERNAME:" \
/etc/security/passwd `
else
        rm -f /etc/ptmp
        exit 1
fi
rm -f /etc/ptmp
awk -F: '$1 == user { print $1":"pw":"$3 ":"$4":"$5":"$6":"$7 }
        $1 != user { print $0 }' user="$USERNAME" pw="$PW" \
                /etc/shadow > /etc/ptmp
chmod 000 /etc/ptmp
mv -f /etc/ptmp /etc/shadow
-----------------------------------------------------------
```

6. Change the permissions to the **passwd** script by entering:

   ```
   chmod 4711 /usr/ucb/passwd
   ```

7. Ensure that each user **PATH** environmental variable specifies that the **/usr/ucb** directory be searched before the **/bin** directory.

Performance Measurement and Tuning for BSD 4.3 System Managers

All devices on this operating system have attributes associated with them. To view device attributes, enter:

```
lsattr -E -l devicename
```

Any attributes with the value True can be modified with the command:

```
chdev -l devicename -a attr=value
```

Attention: Changing device parameters incorrectly can damage your system.

By default, the maximum number of processes per user is 40. The default value might be too low for users who have many windows open simultaneously. The following command can be used to change the value systemwide:

```
hdev -l sys0 -a maxuproc=100
```

This example changes the maximum number to 100. The new value is set once the system has restarted.

To view the current setting of this and other system attributes type:

```
lsattr -E -l sys0
```

The **maxmbuf** attribute is not currently supported by the mbuf services.

This operating system supports the **vmstat** and **iostat** commands, but not the **systat** command or load averages.

Printers for BSD 4.3 System Managers

In AIX 5.1 and later, the operating system supports two printing subsystems: 4.3 BSD and System V. The System V style of printing subsystem uses System V Release 4 commands, queues, and files and is administered the same way. The following paragraphs describe what you need to know to manage the 4.3 BSD style of printing subsystem. You control which subsystem is made active through SMIT. Only one subsystem can be active at a time.

Printing is managed by programs and configurations in the **/usr/lpd** directory. The design, configuration, queueing mechanism, and daemon processes of the 4.3 BSD and printer subsystems for this operating system are different. However, they both use the **lpd** protocol for remote printing. Both systems use **/etc/hosts.lpd**, if it exists, or **/etc/host.equiv** otherwise. The printer subsystem for this operating system offers a gateway to 4.3 BSD printer subsystems, so systems using this operating system can submit print jobs to 4.3 BSD systems and accept print jobs submitted by 4.3 BSD systems.

The **/etc/printcap** file of 4.3 BSD does not exist in this operating system. This file is a combination of spooler configuration and printer capability data base. Users need to understand the format and keywords of the **printcap** file to set up a printer correctly.

The **/etc/qconfig** file of this operating system contains only the spooler configuration information. The printer capability is defined in the ODM predefined/customized data base. You can use the **mkvirprt** (make virtual printer) command to define to the system the capabilities of a particular printer.

To make printer lp0 available to print on the remote host viking, put the following in a 4.3 BSD system **/etc/printcap** file:

```
lp0|Print on remote printer attached to
viking:Z
:lp=:rm=viking:rp=lp:st=/usr/spool/lp0d
```

To do the same on this operating system, put the following in the **/etc/qconfig** file:

```
lp0:
        device = dlp0
        host = viking
        rq = lp
dlp0:
        backend = /usr/lib/lpd/rembak
```

For more information about the printer subsystem, see the Printer Overview for System Management.

This operating system supports the following printer commands and library functions:

cancel(1)	Cancels requests to a line printer
chquedev(1)	Changes the printer or plotter queue device names
chvirprt(1)	Changes the attribute values of a virtual printer
disable(1)	Disables a printer queue
enable(1)	Enables a printer queue
hplj(1)	Postprocesses **troff** output for HP LaserJetII with the K cartridge
ibm3812(1)	Postprocesses **troff** output for IBM 3812 Mod 2 Pageprinter
ibm3816(1)	Postprocesses **troff** output for IBM 3816 Pageprinter
ibm5587G(1)	Postprocesses **troff** output for IBM 5587G with 32x32/24x24 cartridge
lp(1)	Sends requests to a line printer
lpr(1)	Enqueues print jobs
lprm(1)	Removes jobs from the line printer spooling queue
lpstat(1)	Displays line printer status information

lptest(1)	Generates the line printer ripple pattern
lsallqdev (1)	Lists all configured printer queue device names within a queue
lsvirprt(1)	Displays the attribute values of a virtual printer
mkque(1)	Adds a printer queue to the system
mkquedev(1)	Adds a printer queue device to the system
mkvirprt(1)	Makes a virtual printer
pac(1)	Prepares printer/plotter accounting records
piobe(1)	Print Job Manager for the printer backend
pioburst(1)	Generates burst pages (header and trailer pages) for printer output
piocmdout(3)	Subroutine that outputs an attribute string for a printer formatter
piodigest(1)	Digests attribute values for a virtual printer definition and stores
pioexit(3)	Subroutine that exits from a printer formatter
pioformat(1)	Drives a printer formatter
piofquote(1)	Converts certain control characters destined for PostScript printers
piogetstr(3)	Subroutine that retrieves an attribute string for a printer formatter
piogetvals(3)	Subroutine that initializes Printer Attribute database variables for printer formatter
piomsgout(3)	Subroutine that sends a message from a printer formatter
pioout(1)	Printer backend's device driver interface program
piopredef(1)	Creates a predefined printer data stream definition
proff(1)	Formats text for printers with personal printer data streams
qadm(1)	Performs system administration for the printer spooling system
qconfig(4)	Configures a printer queueing system.
qstatus(1)	Provides printer status for the print spooling system
restore(3)	Restores the printer to its default state
rmque(1)	Removes a printer queue from the system
rmquedev(1)	Removes a printer or plotter queue device from the system
rmvirprt(1)	Removes a virtual printer
splp(1)	Changes or displays printer driver settings
xpr(1)	Formats a window dump file for output to a printer.

Terminals for BSD 4.3 System Managers

Traditionally, 4.3 BSD system managers enable or disable terminal ports by modifying the **/etc/ttys** file and sending a **HUP** signal to the **init** program.

This operating system stores terminal port information in the ODM and starts terminals when the **init** program reads the **/etc/inittab** file. In this operating system, use the Web-based System Manager Devices application or SMIT to configure terminal ports.

There is no fixed mapping between the port and the device special file name in the **/dev** directory. Consequently, it is confusing to system managers who are new to this operating system which port is to be configured. When using SMIT, the first planar serial port (physically labeled s1) is referred to as location `00-00-S1`, adapter `sa0`, and port `s1` in the SMIT menus. The second planar serial port (physically labeled s2) is referred to as location `00-00-S2`, adapter `sa1`, and port `s2`.

Use the **penable** and **pdisable** commands to enable and disable a port.

termcap and terminfo

Like System V, this operating system uses **terminfo** entries in **/usr/lib/terminfo/?/*** files. Users with 4.3 BSD Systems might find the following commands helpful:

captoinfo (1)
Converts a **termcap** file to a **terminfo** file

tic (1) Translates the **terminfo** files from source to compiled format.

This operating system includes source for many **terminfo** entries. Some of these might need to be compiled with the **tic** command. The **termcap** file is provided in **/lib/libtermcap/termcap.src** file.

UUCP for BSD 4.3 System Managers

This operating system provides System V Basic Networking Utilities (BNU) which are often referred to as the HDB UUCP.

Dialers(4)	Lists modems used for BNU remote communications links
Maxuuxqts(4)	Limits the number of instances of the BNU **uuxqt** daemons that can run
Permissions(4)	Specifies BNU command permissions for remote systems
Poll(4)	Specifies when the BNU program should poll remote systems
Systems(4)	Lists remote computers with which the local system can communicate
rmail(1)	Handles remote mail received through BNU
uucheck(1)	Checks for files and directories required by BNU
uuclean(1)	Removes files from the BNU spool directory
uucleanup(1)	Deletes selected files from the BNU spooling directory
uucpadm(1)	Enters basic BNU configuration information
uudemon.admin(1)	Provides periodic information on the status of BNU file transfers
uudemon.cleanu(1)	Cleans up BNU spooling directories and log files
uudemon.hour(1)	Initiates file transport calls to remote systems using the BNU program
uudemon.poll(1)	Polls the systems listed in the BNU Poll file
uulog(1)	Provides information about BNU file-transfer activities on a system
uupoll(1)	Forces a poll of a remote BNU system
uuq(1)	Displays the BNU job queue and deletes specified jobs from the queue
uusnap(1)	Displays the status of BNU contacts with remote systems
uustat(1)	Reports the status of and provides limited control over BNU operations.

This operating system also provides the 4.3 BSD **uuencode** and **uudecode** commands. The HDB **uugetty** command is not supported.

For more information, see the lists of BNU files, file formats, and directories in *AIX 5L Version 5.3 System User's Guide: Communications and Networks*.

Index

Special characters

A

B

C

K

L

M

N

P

Q

R

S

T

U

V

W

Y

Z

Vos remarques sur ce document / Technical publication remark form

Titre / Title : Bull AIX 5L System Management Concepts Operating System and Devices

Nº Reférence / Reference Nº : 86 A2 48EM 02

Daté / Dated : October 2005

ERREURS DETECTEES / ERRORS IN PUBLICATION

AMELIORATIONS SUGGEREES / SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Vos remarques et suggestions seront examinées attentivement.
Si vous désirez une réponse écrite, veuillez indiquer ci-après votre adresse postale complète.

Your comments will be promptly investigated by qualified technical personnel and action will be taken as required.
If you require a written reply, please furnish your complete mailing address below.

NOM / NAME : _____ Date : _____

SOCIETE / COMPANY : _____

ADRESSE / ADDRESS : _____

_____

Remettez cet imprimé à un responsable BULL ou envoyez-le directement à :

Please give this technical publication remark form to your BULL representative or mail to:

BULL CEDOC
357 AVENUE PATTON
B.P.20845
49008 ANGERS CEDEX 01
FRANCE

Technical Publications Ordering Form

Bon de Commande de Documents Techniques

To order additional publications, please fill up a copy of this form and send it via mail to:
Pour commander des documents techniques, remplissez une copie de ce formulaire et envoyez-la à :

BULL CEDOC
ATTN / Mr. L. CHERUBIN
357 AVENUE PATTON
B.P.20845
49008 ANGERS CEDEX 01
FRANCE

Phone / Téléphone : +33 (0) 2 41 73 63 96
FAX / Télécopie +33 (0) 2 41 73 60 19
E–Mail / Courrier Electronique : srv.Cedoc@franp.bull.fr

Or visit our web sites at: / Ou visitez nos sites web à:
http://www.logistics.bull.net/cedoc
`http://www-frec.bull.com` `http://www.bull.com`

CEDOC Reference # Nº Référence CEDOC	**Qty** Qté	**CEDOC Reference #** Nº Référence CEDOC	**Qty** Qté	**CEDOC Reference #** Nº Référence CEDOC	**Qty** Qté
_ _ _ _ _ _ _ _ _ [_ _]		_ _ _ _ _ _ _ _ _ [_ _]		_ _ _ _ _ _ _ _ _ [_ _]	
_ _ _ _ _ _ _ _ _ [_ _]		_ _ _ _ _ _ _ _ _ [_ _]		_ _ _ _ _ _ _ _ _ [_ _]	
_ _ _ _ _ _ _ _ _ [_ _]		_ _ _ _ _ _ _ _ _ [_ _]		_ _ _ _ _ _ _ _ _ [_ _]	
_ _ _ _ _ _ _ _ _ [_ _]		_ _ _ _ _ _ _ _ _ [_ _]		_ _ _ _ _ _ _ _ _ [_ _]	
_ _ _ _ _ _ _ _ _ [_ _]		_ _ _ _ _ _ _ _ _ [_ _]		_ _ _ _ _ _ _ _ _ [_ _]	
_ _ _ _ _ _ _ _ _ [_ _]		_ _ _ _ _ _ _ _ _ [_ _]		_ _ _ _ _ _ _ _ _ [_ _]	
_ _ _ _ _ _ _ _ _ [_ _]		_ _ _ _ _ _ _ _ _ [_ _]		_ _ _ _ _ _ _ _ _ [_ _]	
[_ _] : **no revision number means latest revision** / pas de numéro de révision signifie révision la plus récente					

NOM / NAME : ______________________________ Date : __________

SOCIETE / COMPANY : ______________________________

ADRESSE / ADDRESS : ______________________________

____

PHONE / TELEPHONE : ______________ FAX : ______________

E–MAIL : ______________________________

For Bull Subsidiaries / Pour les Filiales Bull :
Identification: ______________________________

For Bull Affiliated Customers / Pour les Clients Affiliés Bull :
Customer Code / Code Client : ______________________________

For Bull Internal Customers / Pour les Clients Internes Bull :
Budgetary Section / Section Budgétaire : ______________________________

For Others / Pour les Autres :
Please ask your Bull representative. / Merci de demander à votre contact Bull.

BULL CEDOC
357 AVENUE PATTON
B.P.20845
49008 ANGERS CEDEX 01
FRANCE

ORDER REFERENCE
86 A2 48EM 02