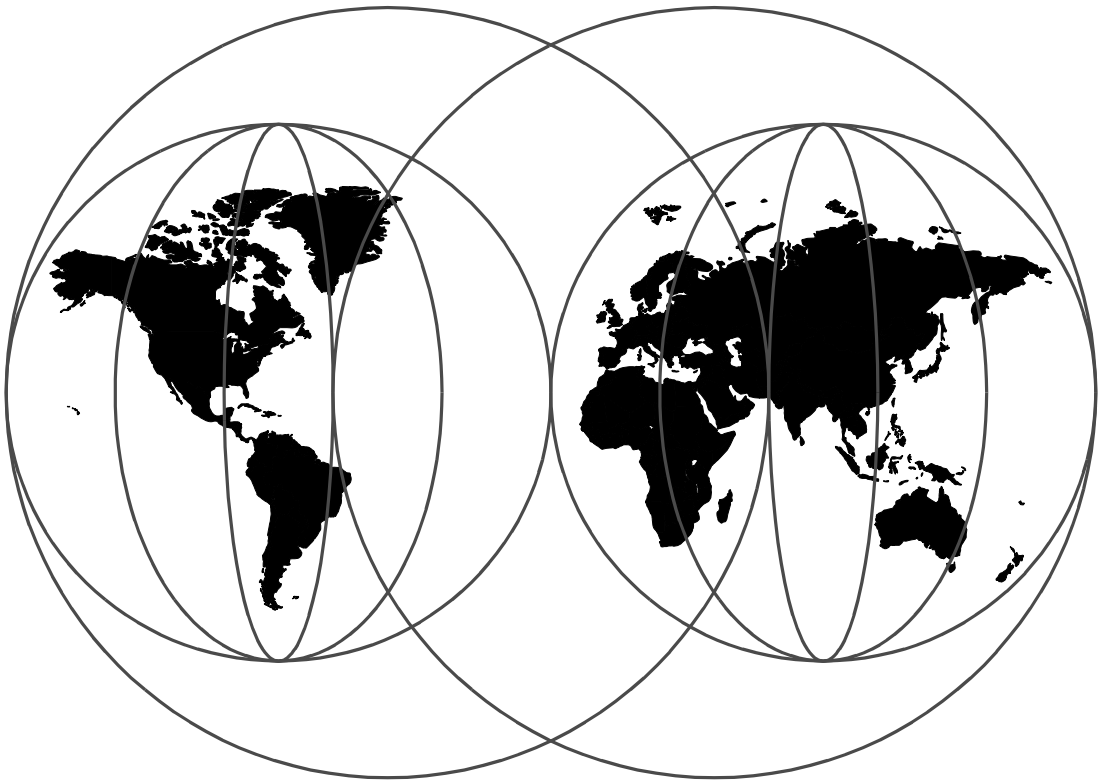
AIX Version 4.3 Differences Guide

Richard Cutler, Zhu Li, Armin Roell



International Technical Support Organization

http://www.redbooks.ibm.com

SG24-2014-01

International Technical Support Organization

SG24-2014-01

AIX Version 4.3 Differences Guide

December 1998

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix A, “Special Notices” on page 305.

Second Edition (December 1998)

This edition applies to AIX Version 4 Release 3, program number 5765-C34 and subsequent releases.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. JN9B Building 045 Internal Zip 2834
11400 Burnet Road
Austin, Texas 78758-3493

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Figures

Tables

Preface

This redbook focuses on the latest enhancements introduced in AIX Version 4.3.2. It is intended to help system administrators, developers, and users understand these enhancements in order to evaluate potential benefits in their own environments.

AIX Version 4.3 includes many new features, including 64-bit application support, IP Version 6, X11 Release 6, Lightweight Directory Access Protocol (LDAP), and improved scaling over a wider range of platforms. The availability of two new Web-based utilities, Web-Based Systems Manager and a Web-based Documentation Search Service, signal AIX's move toward a standard, unified interface for system tools. There are many other enhancements available with AIX Version 4.3, and you can explore them all in this redbook.

This publication is an update to the previously published AIX Version 4.3 Differences Guide, First Edition, which focused on the enhancements introduced in AIX Version 4.3.0. Certain sections of the First Edition have been removed, or edited as required, to reflect the fact that the online documentation provided with AIX Version 4.3 now adequately covers many of the original topics.

How this Redbook is Organized

Throughout this publication, each major section heading indicates which level of AIX 4.3 introduced the enhancement by including the maintenance level in parentheses. For example, the following section heading:

Multiple Concurrent Reads (4.3.1)

indicates that the feature was introduced in AIX Version 4.3.1. If no maintenance level is given, then the feature was included in the initial AIX Version 4.3.0.

The Team That Wrote This Redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Austin Center.

Richard Cutler is an AIX Technical Specialist at the RS/6000 Technical Center in the UK. He has worked with the RS/6000 platform since its introduction and is currently responsible for assisting software vendors and business partners when they migrate their products to AIX.

Zhu Li is an RS/6000 and AIX Technical Specialist at the Technical Support Center in IBM China. She is working on AIX-related problem solving for customers, business partners, and IBM Internal.

Armin Olaf Roell joined IBM Germany in 1995 and works as an RS/6000 system engineer responsible for presales technical support. At present, he is a member of the AIX Technology Focus Group and specializes in general AIX Base Operating System-related matters.

The project that produced this publication was managed by:

Scott Vetter IBM Austin

The authors of the First Edition are:

Colin Fearnley	IBM Johannesburg, South Africa
Andreas Gruber	IBM Munich, Germany
John Hance	IBM Australia
Kevin Murrell	IBM Austin, USA
John Newman	IBM Basingstoke, UK

Thanks to the following people for their invaluable contributions to this project. Without their help, this publication would have been impossible:

Greg Althaus	IBM Austin
Ron Arroyo	IBM Austin
David Babbitt	IBM Austin
Bill Baker	IBM Austin
Greg Birgen	IBM Austin
Larry Brenner	IBM Austin
Luke Browning	IBM Austin
Bill Bulko	IBM Austin
Daisy Chang	IBM Austin
Lliese Chelstowski	IBM Austin
Julie Craft	IBM Austin
John Emmons	IBM Austin
Kevin Fought	IBM Austin
Stan Gowen	IBM Austin
Emilia Hezari	IBM Austin
Elizabeth Higgins	IBM Austin
Dan Hinderliter	IBM Austin
Tom Homer	IBM Austin
Gary Hook	IBM Austin
William J. Hymas	IBM Austin
Yohji Kumazawa	IBM Japan
Joy Latten	IBM Austin
John Maddalozzo	IBM Austin
James Manon	IBM Austin
Brandon Mayfield	IBM Austin
Gerald McBrearty	IBM Austin
Mark McConaughy	IBM Austin
Dwayne McConnell	IBM Austin
Casey McCreary	IBM Austin
Hye-Young McCreary	IBM Austin

Bruce Mealey	IBM Austin
Steve Nasypany	IBM Austin
Chris Nelson	IBM Austin
Grover Neuman	IBM Austin
Ram Pandiri	IBM Austin
Priya Paul	IBM Austin
Stephen Peckham	IBM Austin
George Penokie	IBM Austin
Deanna Quigg	IBM Austin
Mark D. Rogers	IBM Austin
Ken Rozendal	IBM Austin
Ron Saint Pierre	IBM Austin
Yim SeongSoo	IBM Korea
Jim Shaffer	IBM Austin
Jeff A. Smith	IBM Austin
Jeanne Sparlin	IBM Austin
Marc Stephenson	IBM Austin
Randy Swanberg	IBM Austin
Andrew Taylor	IBM Austin
Ashu Tiwary	IBM Austin
Marvin Toungate	IBM Austin
Art Tysor	IBM Austin
Basu Vaidyanathan	IBM Austin
Wayne Wheeler	IBM Austin
Dalal Younis	IBM Austin

Comments Welcome

Your comments are important to us!

We want our redbooks to be as helpful as possible. Please send us your comments about this or other redbooks in one of the following ways:

- Fax the evaluation form found in “ITSO Redbook Evaluation” on page 331 to the fax number shown on the form.
- Use the electronic evaluation form found on the Redbooks Web sites:

  For Internet users `http://www.redbooks.ibm.com`
  For IBM Intranet users `http://w3.itso.ibm.com`

- Send us a note at the following address:

  `redbook@us.ibm.com`

Chapter 1. Hardware Announcements

The following RS/6000 severs, workstations, and adapters were companion announcements with AIX Version 4.3.2. AIX is exhaustively tested on every new hardware enhancement.

1.1 RS/6000 7017 Enterprise Server Model S70 Advanced

The RS/6000 Enterprise Server Model S70 Advanced (Figure 1) is a member of a new generation of 64-bit, 4-way, 8-way, or 12-way symmetric multiprocessing (SMP) enterprise servers. The Model S7A provides the power, capacity, reliability, and expandability to support your next generation mission-critical commercial computing.

With the Model S7A, you can manage the evolution of your business into 64-bit computing while still supporting your existing 32-bit applications. The I/O subsystem supports 32-bit and 64-bit standard PCI adapters.

1.1.1 System Highlights

The Model S7A is packaged as a central electronics complex (CEC) and an I/O rack. The Model S7A Central Electronics Complex entry configuration starts with a 4-way scalable SMP system that uses the 64-bit 262 MHz RS64 II processor with 8 MB of Level 2 (L2) cache per processor. The 4-way SMP can be expanded to a 12-way SMP, and the system memory can be expanded to 32 GB.

The I/O rack contains the first drawer containing the following:

- Service processor
- High-performance disk drive
- 32X maximum speed CD-ROM
- 1.44 MB, 3.5-inch diskette drive
- Two PCI disk adapters

Up to three additional Model S7A I/O drawers can be added, with the restriction of a maximum of two I/O drawers per rack. Additional I/O racks can also be ordered with the S7A. Existing RS/6000 7015 Model R00 racks, or the 7014 Model S00 rack, can also be used for additional storage and communication drawers. This helps to protect your existing investment in SSA or SCSI DASD.

A fully configured system would be as follows:

- 12-way processor
- 32 GB of system memory
- 55 available PCI adapter slots
- 48 hot-pluggable disk bays
- 7 available media bays

The remaining space in the I/O racks can be used for additional storage and communication subsystems.

The RS/6000 Enterprise Server Model S70 Advanced is shipped and delivered with all the internal adapters and devices already installed and configured. AIX Version 4.3.2 software is included with every S7A and can be preinstalled if desired.

Figure 1. RS/6000 Enterprise Server Model S7A

1.1.2 I/O Drawer Specification

Other than the CPU speed increase and maximum supported memory configuration, the main difference between the new Model S7A and the Model S70 is the new I/O drawer.

The new I/O drawer differs from the previous S70 drawer in the following details:

- Ten EIA units in size instead of seven EIA units.
- Dual redundant hot pluggable power supplies. Although the previous model drawer has two power units, they power different drawer components. This means the failure of a single power supply results in the whole drawer being unusable.
- N+1 hot pluggable cooling fans.
- Support for internal SSA disk drives (not in primary drawer). It is only possible to have internal SCSI disks in the previous drawer.
- Ultra SCSI internal disks connected to an Ultra SCSI adapter card. The previous drawer used Ultra SCSI disks connected to a SCSI-2 F/W adapter.
- The option of implementing an internal SCSI disk system as two 6-packs, each driven by a separate SCSI adapter. This is useful when mirroring disks for availability, as it means the disk adapter is no longer a single point of failure. The previous drawer implemented the internal SCSI disk as one 12-pack. All disks in the drawer were attached to one SCSI controller.

- Two media bays per I/O drawer. The previous model I/O drawer has three media bays. As with the previous model drawer, the first drawer in a system comes with a 32X CD-ROM drive in one of the bays.
- -48v DC power option is no longer available.

1.2 RS/6000 43P 7043 Model 150

The IBM RS/6000 43P 7043 Model 150 is an entry-level desktop RS/6000 workstation, or workgroup server, offered at an affordable price. The 150 provides a continuation of the successful 43P line of entry workstations.

The Model 150 is a uni-processor system that provides an enhanced performance over its predecessor, the Model 140, by using a 375 MHz PowerPC 604e processor and an enhanced memory controller. With this memory controller, the Model 150 uses SDRAM memory and an 83 MHz memory bus speed. The system memory can also be expanded up to 1 GB.

With Ethernet and Ultra SCSI controllers integrated on the planar, the Model 150 also contains five PCI slots and five bays for expansion and growth capability. This makes the Model 150 an attractive investment as either a technical workstation or an entry workgroup server.

The Model 150 supports a variety of 2D and 3D graphics adapters (including a new advanced 3D graphics adapter), offering excellent graphics price and performance. In addition, a robust set of disk drive and communications features are available.

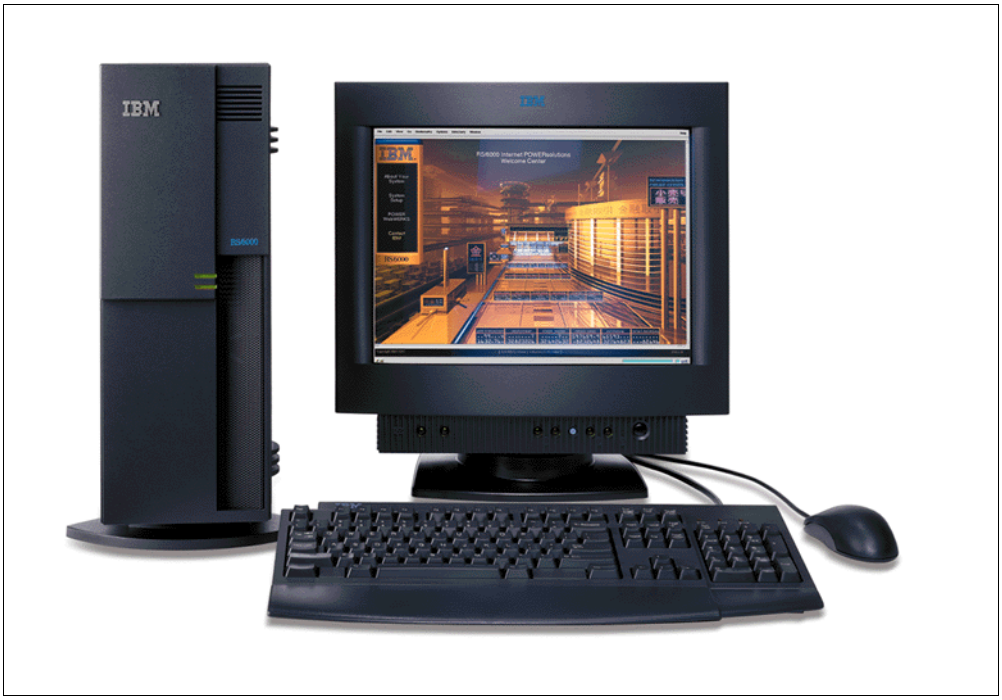


Figure 2. RS/6000 43P Model 150

The Model 150 incorporates the following features:

- 375 MHz PowerPC 604e processor.

- 1 MB of Level 2 (L2) cache.
- Four memory slots, which allow system memory from 128 MB to 1 GB of ECC SDRAM.
- Integrated 10/100 Mbps Ethernet with RJ45 and AUI connectors.
- Integrated Ultra SCSI controller.
- Five PCI slots in total. Two are short slots, and three are full length. The number of available slots depends on the type of graphics adapter selected, if any.
- Five bays in total. There are two disk and three media bays. The standard 4.5 GB SCSI disk occupies one disk bay; the other disk bay is available. A 1.44 MB 3.5-inch diskette drive and a 32X max. speed CD-ROM occupy two of the media bays. One media bay is available.
- Two serial ports, one parallel port, keyboard, mouse, and graphics tablet ports.

A variety of 2D and 3D graphics adapters, including a new advanced 3D graphics adapter GXT3000, are available for the Model 150. The available adapters are shown in Table 1.

Table 1. Graphics Adapters Available for the Model 150

Adapter	Feature Code	PCI Slot	Type
GXT120P	2838	1-5 (4 max)	short
GXT250P	2851	1-5 (4 max)	short
GXT255P	2852	1-5 (4 max)	short
GXT550P	2855	2, 3 (1 max)	long
GXT3000P	2825	3 (1 max)	long

1.3 RS/6000 43P 7043 Model 260

The Model 260 is designed for use as a mid- to high-range graphics workstation and as an entry server. It is a new addition to the RS/6000 workstation and workgroup server family.

The Model 260 is an affordable 64-bit symmetric multiprocessing (SMP) system with true multithreaded application support and extended floating-point capabilities. It provides significant performance enhancements over the Model 150 and the Model 240.

The graphics capability, along with the excellent price and performance, position the Model 260 as an MCAD, CAE, Geophysical, and entry technical server design and analysis solution.

It can be used as an entry SMP/64-bit server for ISVs and customers who want a cost-effective development platform for developing and testing applications that will run on larger RS/6000 systems.

The dual processing power of the Model 260 and its small package make it an excellent solution for Internet service providers and customers that need, or want, a stand-alone Internet server. The Model 260 is well suited for network computing

and interoperability. It can be used as either a stand-alone multi-user, departmental, transaction, or database server.

Figure 3 shows the Model 260 with additional equipment.

Figure 3. RS/6000 43P Model 260

The following is a summary of the Model 260:

- 1- or 2-way SMP system using the 64-bit 200 MHz POWER3 processor.
- 4 MB of Level 2 (L2) cache per processor.
- The system memory can be expanded up to 4 GB.
- A 10/100 Mbps Ethernet, Ultra SCSI, and a service processor are integrated within the Model 260.
- Contains five PCI slots. Two of the slots are 64-bit at 50 MHz, and three are 32-bit at 33 MHz.
- Contains five bays. There are two disk and three media bays. One disk bay is available. A 1.44 MB 3.5-inch diskette drive and a 32X CD-ROM occupy two of the media bays.
- A variety of 2D and 3D graphics adapters, including a new advanced 3D graphics adapter (GXT3000), are available for the Model 260.
- An auto-restart (reboot) option, when enabled, automatically reboots the system following an unrecoverable software error, hardware failure, or environmental-induced (power) failure.

The AIX Version 4.2.1 or 4.3.2 operating system is included with each Model 260 and can be preinstalled, if desired.

The Model 260 is designed for customer setup of the machine and for subsequent addition of most features (adapters/devices).

Table 2 shows a summary of all supported graphics adapters for the Model 260.

Table 2. Graphics Adapters Available for the Model 260

Feature Code	Adapter	Max. Quantity	Slot Priority	Size
2838	GXT120P	4	1,2,3,4,5	short
2851	GXT250P	4	1,2,3,4,5	short
2852	GXT255P	4	1,2,3,4,5	short
2825[1]	GXT3000P	1	2 only	long
[1] Note: GXT3000P requires AIX 4.3.2. Support for AIX 4.2.1 is planned at a later date.				

Note

Any GXT550P shipped before October, 1998 is not compatible. The GXT800P must be installed in slot 1. Slot 2 and 3 will not be available.

1.4 GXT3000P PCI Graphics Accelerator

AIX Version 4.3.2 introduces support for the new high performance RS/6000 POWER GXT3000P PCI Graphics Accelerator that attaches to the RS/6000 43P 7043 Models 150 and 260.

The GXT3000P provides hardware acceleration for the new OpenGL 1.2 version of the API. This adapter accelerates advanced 3D graphics functions, such as Gouraud shading, antialiasing, depth buffering, fog, atmospheric effects, and blending. This enables your 3D applications to run more quickly and with good interactive performance. Additional benefits when using the POWER GXT3000P graphics accelerator are:

- Increased subpixel addressing over previously supported adapters
- Face culling support
- OpenGL polygon offset support
- OpenGL polygon mode support
- OpenGL specular lighting support in hardware
- Trilinear texture mapping

The POWER GXT3000P offers a highly flexible frame buffer that can be dynamically configured to provide a broad set of color and feature options. When using OpenGL and graPHIGS APIs, the GXT3000P offers support for 8-bit and 24-bit double-buffered color and includes 8-bits of double-buffered alpha buffers for more realistic transparency control. In addition, this accelerator provides 8-bit overlay buffers that enhance the speed of a graphical user interface (GUI), 8-bit stencil buffers, 8-bit window ID buffer, and a 24-bit Z-buffer for hidden surface removal operations. The POWER GXT3000P allows for the display of up to 16.7 million simultaneous colors.

The GXT3000P graphics accelerator is a single 64-bit PCI card that attaches to one 32-bit or 64-bit PCI slot in the RS/6000 Models 150 or 260 but covers the slot adjacent to it (uses two slots of space).

The GXT3000P supports display resolutions of up to 1280x1024 at an 85 Hz refresh rate, including monitors that comply with the ISO 9241 Part 3 ergonomic standard.

1.5 Gigabit Ethernet-SX PCI Adapter

The 1 Gb network adapter is the next generation high speed LAN connection initially targeting backbones and consolidation of 10/100 Mb Ethernet segments.

The device driver of the PCI Gigabit Ethernet adapter is supported in AIX Versions 4.2.1 and 4.3.2. Earlier versions of AIX will not be supported. The machine models that support this adapter are: RS/6000 Models 260, F50, H50, S70, and S7A.

The Gigabit Ethernet-SX PCI adapter is designed for customers that require high performance LAN networks. It supports:

- 64-bit wide PCI slots at 33 or 66 MHz
- Multiple IP subnets without traversing routers
- Up to 64 IP address assignments on one network connection
- Hardware for TCP, UDP, and IP checksums (requires AIX 4.3.2 or later)
- Remote network boot on both Open Firmware and legacy PCI boxes
- User configurable options for multiprocessor systems

When the user-configurable option for MP systems with high bandwidth adapters is set, this option enables TCP/IP input processing to be off-loaded to kernel threads to improve overall throughput (one processor runs the device driver interrupt, while others perform the TCP/IP processing).

This driver conforms to the following specifications:

- PCL Local Bus Revision 2.0 and 2.1
- IEEE 802.1Q frame tagging
- IEEE 802.3z switches and duplex repeaters
- IEEE 802.3q VKAN tagging
- IEEE 802.3x flow control standards

The adapter and device driver support fiber-optic cabling 1000 Base SX and full-duplex mode. The adapter uses short wave 850 nm multimode 62.5 or 50 micron fiber cables.

The adapter and device driver support jumbo frame 9014 byte Ethernet packets. AIX Version 4.2.1 only supports standard size Ethernet packets. Currently, jumbo packets only work when Alteon switches are used.

Installation of AIX using this adapter is supported. The code to support this is a part of the system firmware. The AIX product device driver will run on the adapter after control has been passed to AIX.

Table 3 is a comparison of various Ethernet adapters.

Table 3. Ethernet Adapters Comparison

Features	Ethernet 10 Base T	Fast Ethernet 100 Base T	Gigabit Ethernet 1000 Base X
Data Rate	10 Mbps	100 Mbps	1 Gbps
Cat 5 UTP	100 m	100 m	100 m (1000 Base-T)
STP/Coax	500 m	100 m	25 m (1000 Base-CX)
Multimode Fiber	2 km	412 m (hd)	500 m (1000 Base-SX)
Singlemode Fiber	25 km	2 km (fd)	3 km (1000 Base-LX)

More information about the Gigabit Ethernet standard can be found at:

- `http://www.gigabit-ethernet.org`
- `http://grouper.ieee.org/groups/802/`

1.5.1 Adapter Parameters

You can alter the following adapter parameters:

- Transmit Queue Size (tx_que_size)

  This indicates the number of transmit requests that can be queued for transmission by the device driver. Valid values range from 512 through 2048. The default value is 512.

  Note: the fixed hardware transmit queue size of 512 is included in this number.

- Transmit and Receive Jumbo Frames (jumbo_frames)

  Setting this attribute to yes indicates that frames up to 9014 bytes in length may be transmitted or received on this adapter. If you specify the no value, the maximum size of frames transmitted or received is 1514 bytes.

  Note: specifying yes will use additional system memory for buffers.

- Enable Alternate Ethernet Address (use_alt_addr)

  Setting this attribute to yes indicates that the address of the adapter, as it appears on the LAN network, is the one specified by the ALTERNATE ETHERNET address attribute. If you specify the no value, the unique adapter address written in a ROM on the adapter card is used. The default value is no.

- Alternate Ethernet Address (alt_addr)

  This allows the adapter unique address, as it appears on the LAN network, to be changed. The value entered must be an Ethernet address of 12 hexadecimal digits and must not be the same as the address of any other Ethernet adapter. There is no default value. This field has no effect unless the Enable ALTERNATE ETHERNET address attribute is set to the yes value, in which case, this field must be filled in. A typical Ethernet address is: 0x02608C000001. All 12 hexadecimal digits, including leading zeros, must be entered.

  Note: the universal/local bit does not have to be set to local.

- Use Production Microcode (use_pcode)

  Setting this attribute to yes indicates that the production microcode should be downloaded to the adapter. If you specify the no value, the debug version of the microcode that has traces enabled is used. The default value is yes.

1.5.2 Error Logging

The following error log entries are currently logged by the device driver:

- ERRID_GXENT_ADAP_ERR

  Indicates that the adapter self-check failed during initialization, and that the error is irrecoverable. User intervention is necessary to fix the problem.

- ERRID_GXENT_DOWNLOAD_ERR

  Indicates that an error occurred while downloading firmware to the adapter, and that the error is irrecoverable. User intervention is necessary to fix the problem.

- ERRID_GXENT_EEPROM_ERR

  Indicates that an error occurred while reading the adapter EEPROM, and that the error is irrecoverable. This error will only happen at configuration time. User intervention is necessary to fix the problem.

- ERRID_GXENT_TX_ERR

  Indicates that the device driver has detected a transmission error (transmit time out). The device driver will enter network recovery mode where it will reset the adapter, re-download the firmware, and so on. If the recovery is successful, the driver will log ERRID_GXENT_ERR_RCVRY_EXIT; otherwise, ERRID_GXENT_ADAP_ERR or ERRID_GXENT_DOWNLOAD_ERR will be logged. User intervention is not required (unless ERRID_GXENT_ADAP_ERR or ERRID_GXENT_DOWNLOAD_ERR is also logged).

- ERRID_GXENT_LINK_DOWN

  Indicates that the adapter has detected that the link between the adapter and the switch is down. The device driver will not enter network recovery mode. Transmission requests on the hardware and software transmit queues will not be deleted. The device driver will reject transmission requests until the link is established. Control operations will still be allowed during this time. When the link is established, the device driver will log ERRID_GXENT_ERR_RCVRY_EXIT. User intervention is necessary to fix the problem.

- ERRID_GXENT_ERR_RCVRY_EXIT

  Indicates that the adapter encountered a temporary error (TX_ERR or LINK_DOWN) that halted network activity, and the problem has been resolved.

- ERRID_GXENT_UNSUPPORTED_ERR

  Indicates that the adapter is not an IBM adapter. This error will only happen at configuration time and is irrecoverable. User intervention is necessary to fix the problem.

1.6 Gigabit Fibre Channel Adapter

AIX 4.3.2 supports the new Gigabit PCI adapter for Fibre Channel Arbitrated Loop (FC-AL) connectivity on RS 6000 S70/S7A machines. The adapter has an expected availability date of March 26, 1999. It uses a shortwave over optical fibre cable for a distance of up to 500 meters. With the use of optical fiber cabling, this adapter allows users to establish a high speed network for local and remote storage.

Fibre Channel Arbitrated Loop (FC-AL) began as the Fiber Channel Standard (FCS) originally developed by IBM as a communications technology for computer systems. FC-AL provides a high-speed interconnect running at 100 MB/s for a single loop, 200 MB/s for dual loop-theoretical.

FC-AL is an arbitrated loop technology that allows only one-device access to the bus for I/O operations.

This adapter is supported on AIX 4.3 with APAR IX81852.

This adapter has the following characteristics:

- One initiator capability over an optical fiber link, one Gigabit Fibre Channel Adapter per loop
- Transmitting distance at 500 meters when using 50/125 micron cable and 175 meters when using 62.5/125 micron cable
- Data rate throughput at 100 MB/s burst, 75 MB/s sustained
- 25 MHz to 33 MHz clock speeds
- 132 MB/s peak data transfer rate
- 32-bit PCI Bus Master Mode DMA, 32 bit PCI Bus Slave Mode interface
- Slave Mode supports medium device select timing
- 32-bit PCI adapter with 64-bit addressing capability
- Transmitting rate of 1062.5 MB/s
- Short PCI form factor (6.875 x 4.2 inches)
- Bus master, scatter/gather architecture
- Selectable big/little endian
- GLM medium adapter allows future use of copper or longwave optics
- Supports 2048 exchanges
- 6 KBx36 frame buffer (4 KB receive, 2 KB transmit)
- 128 KBx32 SSRAM for processor memory
- 512 KB Flash EPROM that contains microcode
- 233 MHz ARM processor as the adapter controller
- Support for 5.0 Volt or 3.3 Volt electrical signaling
- Estimated performance is 5000 to 9000 SCSI 4 KB OP/s
- PCI Local Bus Revision 2.1 compliant
- No IPL/Boot support

The Gigabit Fibre Channel adapter provides an attachment of one or two external RAID subsystems to one or two hosts. The two host configuration supports HACMP. Single host configuration with two adapters support fail-over from one controller to the other. A follow-on product will provide 4-way attachment.

The new adapter supports connectivity to the IBM 2102-F10 Fibre Channel RAID Disk Array subsystem with IBM 2102-D00 Storage Drawers through the optional IBM 2103-H07 Fibre Channel Hubs.

The supported configurations include:

- Point-to-Point

  In this configuration, an adapter is connected directly to one RAID subsystem using the shortwave cable. Even though it is physically a point-to-point configuration, it logically appears to be a private (closed) arbitrated loop. Maximum distance between the adapter and subsystem is 500 meters.

- Private Arbitrated Loop

  In this configuration, one hub is attached to the adapter through the shortwave cable. The hub is a non-intelligent device. It is considered to be part of the cabling and is not recognized by the adapter as a device. From the hub, up to two RAID subsystems can be attached.

Although Gigabit Fibre Channel and Gigabit Ethernet could use similar physical media, such as fiber optic cable, Gigabit Fibre Channel and Gigabit Ethernet operate at different speeds and protocols. For example, Fibre Channel operates at 1.0625 Gbps, while Gigabit Ethernet operates at 1.25 Gbps. Fibre Channel protocol is optimized for the low-latency and high reliability required by storage, while Gigabit Ethernet protocol is optimized for LAN.

Table 4 compares the features of FC-AL and SSA technology.

Table 4. Features of FC-AL and SSA

Features	**FC-AL**	**SSA**
Bandwidth	100MB/s	80MB/s (20MB per link)
Connectivity (nodes per loop)	126 nodes	128 nodes
Distance between nodes	30m (copper) 10km (optical)	25m (copper) 2.4km (optical)
Arbitration	Yes	No
Connector	9-pin DIN (copper)	9-pin DIN

Chapter 2. AIX Kernel Enhancements

This chapter examines the changes in the AIX base kernel that are new with AIX Version 4.3.

2.1 Binary Compatibility

The AIX architecture and development teams place a very high priority on ensuring that binary compatibility exists for customers who want to migrate their applications to later versions of AIX. The following sections explain the extent of this compatibility and the few areas where problems may arise.

2.1.1 Compatibility between AIX Version 4 Releases

Applications written using earlier releases of AIX Version 4 (Release 1 or Release 2) for RS/6000 POWER, POWER2, POWER3, and PowerPC-based models, can be executed on AIX Version 4 Release 3 without recompilation for the same and newer models in that processor family (POWER, POWER2, POWER3, or PowerPC). The exceptions to this statement are applications using:

- Non-shared compiles of AIX shared libraries
- Features explicitly described as non-portable by IBM in the AIX Version 4 reference manuals
- Undocumented AIX internal features
- X11R5 server extensions (AIX Version 4.3 Only)
- Applications compiled using POWER2-, POWER3-, or PowerPC-specific compiler options but executed on models other than POWER2, POWER3, or PowerPC.

Note: Applications compiled on a given release level of AIX Version 4 may not operate properly on systems running an earlier release of AIX Version 4.

Any program intended to run in all environments, POWER, POWER2, POWER3, and PowerPC (601 and newer PowerPC processors), *must* be compiled using the common mode option of the compiler. Programs compiled to exploit POWER2 technology must be run on POWER2-based processors. Programs compiled to exploit POWER3 technology must be run on POWER3-based processors. Programs compiled to exploit PowerPC-based technology must be run on PowerPC-based processors. Existing binaries do not need to be recompiled to operate on the target processors.

64-bit applications produced using AIX Version 4 Release 3 on any of the 32-bit processor models, or the 64-bit processor models, will execute without recompilation on the 64-bit processor models. 32-bit applications produced using AIX Version 4 Release 3 on either 32- or 64-bit processor models will execute without recompilation on both models.

2.1.2 X11 Compatibility

The AIX 4.3 X-server has been upgraded to the X Consortium Release 6 version of X (commonly known as X11R6). The libraries shipped by IBM with X11R6 are backward-compatible, and the client applications that access these libraries will

work as on previous releases of AIX. As on earlier releases of AIX, IBM will also ship X11R3, X11R4, and X11R5 compatibility installation options for maximum customer flexibility. In this way, client applications will experience no problems with compatibility.

The majority of applications using X fall into this category and will not have any difficulties. However, a small number of X applications use the loadable extension facility provided by the X server.

The X-server allows for the addition of new function through its extension mechanism. For each extension, part of the extension is loaded into the X-server before it can be executed. X11R6 has modified this mechanism, and it is this part of the extension that must be made compatible with X11R6 to execute properly. All extensions supplied by IBM have been made compatible and will execute properly. In some circumstances, a customer may have an extension that will not work with X11R6, such as:

- Customers who have sample extensions downloaded from the X Consortium FTP site.
- Customers who have developed their own extensions.
- Customers using third-party extensions.

In these cases, the extensions will need to be made compatible with X11R6 before they will execute properly. Customer-developed extensions and sample X consortium extensions will need to be recompiled with the X11R6 environment. For third-party extensions, the customer should contact the vendor for a X11R6-compatible update.

Customers using non-IBM display adapters may also be using vendor-supplied software specific to those devices that use X-server capabilities. If so, this software must be made compatible with X11R6 to operate properly. The customer should contact the vendor of the display adapter for this software.

IBM provides a porting guide with AIX Version 4.3 that also appears on *The Developers Connection* CD to assist customers and vendors developing adapters or extensions for AIX. *The Developers Connection* can be found at the following URL:

`http://www.developer.ibm.com/devcon/`

2.1.3 AIX Version 3 Application Compatibility

All AIX applications correctly written using AIX Version 3 Release 2 or greater for POWER, POWER2, and PowerPC-based models will run on AIX Version 4 without recompilation for the same models. Exceptions are applications that use:

- Their own loadable kernel extensions
- Certain High Function Terminal (HFT) control interfaces
- X11R3 input device interfaces
- The CIO LAN device driver interface
- SCSI device configuration methods (IHVs)
- The nlist() interface
- DCE threads

Other exceptions include applications compiled using POWER2 or PowerPC-specific compiler options that run on models other than POWER2 or PowerPC.

Any program designed to run in all environments, that is, POWER, POWER2 and PowerPC (601 and above), *must* be compiled using the common mode option of the compiler. Programs compiled to exploit POWER2 technology must be run on POWER2-based processors. Programs compiled to exploit PowerPC-based technology must be run on PowerPC-based processors. Existing code does not need to be recompiled to run.

Note: Applications created on a system using AIX Version 4 may not function properly on a system using AIX Version 3.

For these statements to apply, applications must have been created using the AIX shared libraries.

2.1.4 Client/Server Application Compatibility

An RS/6000 system using AIX Version 3 Release 2 or greater can operate as a server system for client machines using AIX Version 4 with the following exceptions:

- Service of Version 4 diskless/dataless machines
- Network install of Version 4 clients
- Service SNA or X.25 to Version 4 clients
- Service HCON to Version 4 clients
- Service CGE extensions of PEX and PEX-PHIGS
- Use of AIX Version 4 client install formats

An AIX system using AIX Version 4 may operate as a server system for client machines using AIX Version 3 Release 2 or greater as long as the proper compatibility options are installed. All statements about binary compatibility apply in this case. Version 4 applications may not execute properly on Version 3 systems using remote network mounts of file systems.

In both cases, minor administrative changes must be made to the AIX Version 3 systems to support the new AIX Version 4 LFT terminal type.

2.1.5 IBM Licensed Program Products Compatibility

There are hundreds of Licensed Program Products (LPP) available for AIX Version 4. IBM LPPs currently sold for AIX Version 4 Release 1 or Release 2 will operate without change on AIX Version 4 Release 3 with certain exceptions, such as newer versions or releases of a product being available. For information about a specific LPP, the *AIX Version 4 IBM Licensed Program Products RoadMap* can be found at the following URL.

`http://www.rs6000.ibm.com/software/Apps/LPPmap.html`

This document contains information about latest LPP version levels, support, and AIX release compatibility.

For AIX systems using AIX Version 3 Release 2 or greater needing to migrate to AIX Version 4, the publication *A Holistic Approach to AIX V4 Migration, Planning Guide*, SG24-4651, contains information about LPPs and AIX release compatibility.

2.2 AIX 4.3 for 12-Way SMP Performance (4.3.1)

AIX Version 4.3 is tuned for 12 CPU SMP performance efficiency. This is a continuation of the scalability evolution. Certain kernel areas identified by the AIX performance team were investigated, tuned, and redesigned where necessary to eliminate situations that could have impeded 12-way SMP performance.

2.3 32 GB Real Memory Support (4.3.2)

AIX now supports up to 32 GB of real memory and has been enabled to support larger memory sizes as hardware grows in capacity.

CHRP (Common Hardware Reference Platform) is the system architecture base for systems with large physical memory or any memory above 32-bit real addresses. AIX will only support real memory addressing greater than 32-bits on CHRP systems. Current AIX versions and their supported maximum memories are provided in Table 5.

Table 5. Maximum Supported Memory Sizes

AIX Version	Maximum memory supported
AIX 4.1.5	2 GB
AIX 4.2.1	4 GB
AIX 4.3.0	16 GB
AIX 4.3.2	32 GB

2.4 Lock-Based Dumping

In AIX Version 4.1 and 4.2, the AIX dump routines always dumped the same data areas. This generic policy meant that certain key data areas were kept out of system dumps because their inclusion would greatly increase the size of the dump. For AIX Version 4.3, dump routines have been added that gather additional information for inclusion in a dump (based on the status of certain locks or flags in the kernel) when the system dump is initiated.

If a lock protecting a structure is held at the time of the dump, then almost certainly, that structure must have been in the process of being updated and should be included in the dump. The primary area where this information is of use is in the Virtual Memory Manager (VMM).

With these additional routines, the need to inconvenience customers with debug kernels or reproduce the problem on test systems with the kernel debugger is greatly reduced.

2.4.1 Dump Support

You can now use the `dump` interface, through a dump table, to dump specified memory using a real address without requiring the real address to have virtual address mapping.

To support dumping real memory using a real address, a new data structure, and a new magic number, `DMP_MAGIC_REAL`, has been defined. The following are modified to check for the magic number and handle the new table format:

- savecore.c
- copydump.c

savecore() and copycore() check for both DMP_MAGIC and DMP_MAGIC_REAL and are able to process either of the dump table formats.

When displaying the data for a dump table, `crash` and `dmpfmt` use an address format that distinguishes real addresses from virtual addresses. A real address can be entered on a `crash` subcommand as *r:address*. For example:

```
> od r:10012
```

When `dump` is initialized, it allocates one page from the pinned heap. It also gets the real address for this page. When dumping memory referenced by a virtual address, dump will do the following for each page (or page segment) to be copied:

1. Turn data translation off
2. Copy the data to the buffer at the real address
3. Turn data translation back on
4. Dump the data

This retrieves the data in real mode while calling the device driver code in virtual mode.

Note: Only one page will be dumped at a time.

2.4.2 Programming Interface

The external dump interfaces are found in /usr/include/sys/dump.h, and new structures have been defined in this file. In the dumpinfo structure, dm_hostip becomes __ulong32_t. The structures dump_read and dumpio_stat are defined for the kernel and extensions only.

2.5 Bad Block Relocation during System Dump (4.3.1)

In previous versions of AIX, if the LVM detected a bad block and received an I/O error while processing a system dump, the dump was aborted if no secondary dump device was available. In AIX Version 4.3.1, the LVM will now try to relocate the bad block so that processing of the system dump can continue and information is not lost.

2.6 Kernel Protection

Memory overlays are extremely destructive and, in certain cases, can destroy the kernel's ability to respond to interrupts, making it impossible to obtain a dump. A

destructive overlay can be caught when it occurs if the kernel code is protected from overwrites. The AIX kernel has therefore been enhanced to provide some protection against these types of errors. The first page of memory is now protected from writes by setting page protection bits in virtual page tables. A similar scheme has been implemented for other pages in the kernel that contain nothing but code (since code should never be altered). Any attempt to overlay protected pages now results in dumps that point directly to the program that tried to do the overwriting. This cuts out the most expensive and time-consuming part of memory overlay debugging for a large number of overlay cases.

For kernel text, enough symbol information has been added to the kernel space so that the kernel text is protected during system initialization. Note that pages containing a mixture of data and text, or data only, cannot be protected, so some kernel text remains writable.

Kernel extension text areas are optionally protected. A run-time check enables the system loader to protect kernel extension text areas. If xmdbg is set by the bosdebug or bosboot commands, text pages are protected at kernel extension load time. Pages that share text and data are not protected.

Note: This change has impacted kernel and kernel extension code that attempts to modify text. Self-modifying kernel extensions will cause the system to crash unless those extensions also modify the protection of the text pages.

This design protects as many pages in the kernel space as is practical without disturbing delicate assembler routines or increasing the working set needed to run the kernel.

2.6.1 Storage Protection Macro

The STORE_PROTECT macro has been added to store-protect whole pages that reside between two symbols (x and y). This macro is defined as follows:

```
#define STORE_PROTECT(x,y) if (STARTOFPAGE(y) > NEXTPAGE(x))  \
        vm_protect(NEXTPAGE(x),STARTOFPAGE(y)-NEXTPAGE(x),RDONLY)
```

The STORE_PROTECT macro has the effect of protecting all pages starting with the next page boundary beyond x until the last page boundary before y. This macro is used during system initialization for the various regions in the kernel and conditionally by the loader during kernel extension load time.

During system initialization, k_protect() is called to protect the regions marked by the bind steps. k_protect() is called from main() in the following sequence:

```
debugger init();    /* start the kernel debugger */
kmem init();        /* initialize kernel memory heaps */
k_protect();        /* store protect kernel text areas */
strtdisp();         /* start up the dispatcher */
```

When called, k_protect() does the following:

- Store protects low.o areas, at least the first three pages
- Store protects pinned text sub binds
- Store protects paged text sub binds

2.6.2 Debug Modifications

Since the debugger cannot store to some areas in virtual mode due to kernel protection, the debugger has been altered so that all stores to virtual memory addresses are first transplanted and then performed in real mode. This operation is transparent to the debug user. It requires a modification to the get_put_data_aligned() routine so that virtual operations are translated and performed in real mode. I/O space has not been affected.

2.6.3 Stack Overflow Protection

A stack overflow detection mechanism has been implemented. i_poll() and i_poll_soft() check the MST save-area located lower in memory to see if the csa_prev values, that would be used if the interrupt is interrupted, are valid. If this location contains incorrect data, it is repaired if possible, and the code logs an error.

2.7 SMP TTY Handling

Currently on J30 and J40 SMP systems equipped with 128-port adapters, when the system is under load, CPU 0 spends a great deal of time off-level polling the various 128-port adapters for incoming events.

To alleviate this problem, instead of CPU 0 being used as a timer handler, the load has been passed to other CPUs that are available, thereby improving the overall SMP performance.

2.8 Faster Per-Thread Data

In previous versions of AIX, all threads shared an identical address space. When per-thread data needed to be accessed, a fairly expensive lookup had to be performed by the get_thread_specific() routine.

In a non-threaded version of the OpenGL API (which is very call intensive), tests show that you can expect to spend roughly 150 cycles per call (on average) in a routine. Using the existing get-thread-specific() routine would add approximately 70 cycles (or 50 percent overhead) to enable a multi-threaded OpenGL API. A much faster mechanism to access per-thread data for 32-bit systems is therefore required. For 32-bit systems, a separate segment 0 for each processor is now provided. This segment contains a page of thread-specific data that is modified as each thread is swapped in. Faster access to private memory should also provide benefits to the thread libraries.

2.9 Expanded Limits on Open Files (4.3.1)

In previous versions of AIX, a single process was limited to a maximum of 2000 open files at any one time. There was also a total system-wide limit of 200,000 open files. This number was entirely arbitrary, and although it was perfectly adequate for most processes, it was not enough for some. AIX Version 4.3.1 increases these limits to the following.

- Maximum of 1,048,576 open files system wide.
- Maximum of 32,767 open files per process.

The maximum number of file descriptors per process is defined by the constant OPEN_MAX. In AIX 4.3.1 it is 32767.

However, this change can create certain compatibility problems with programs that were compiled with the old OPEN_MAX value of 2000. So there must be a way to enforce the old OPEN_MAX value for existing programs, yet allow new programs to exploit the new capability. This has been done with the existing resource limit functions. There was already a limit for number of available file descriptors, but it has always been set to RLIM_INFINITY. In AIX 4.3.1, the setrlimit() and getrlimit() system calls can be used to maintain specific values for RLIMIT_NOFILE. By default, the soft limit will be the old value of OPEN_MAX, 2000. The default and maximum hard limit will be the new OPEN_MAX value, 32767. With these limits, everything should continue to work as before with no user intervention. If a user increases the soft limit, then programs written to exploit the increased table size can be used.

In addition to the system calls for managing limits, the user can change their limit for number of file descriptors with the `ulimit -n` command. Because the hard limit is set to OPEN_MAX, any user can increase the limit, privileged access is not required.

2.10 Multiple Concurrent JFS Reads (4.3.1)

AIX uses a simple lock type to serialize access to the in-core inode of a file or directory on a JFS file system. This is to ensure data integrity, particularly on MP systems, where multiple threads can be accessing an inode simultaneously. When reading a file, the lock is used to serialize updates to the last access time stamp in the inode. This lock has been identified as a potential performance bottleneck in the situation where multiple threads are attempting to read the same file, particularly when migrating from UP to MP systems.

This type of problem affects customers who use databases on JFS file systems and do not have a choice because their database application does not support raw partitions. Examples include Progress, and Universe, to name two. There are also some large customers who use JFS for their databases. The problem stems from the length of time the lock is held. A thread would obtain the lock and then initiate the I/O to read the required data before updating the access time field in the inode and releasing the lock. During this time, other threads would be blocked from accessing the file.

To alleviate this problem, AIX 4.3.1 has changed the mechanism for reads from JFS file systems to minimize the length of time the inode lock is held by a thread.

When only one thread is reading the file, no change has been made. The reading thread obtains the inode lock and sets a flag in the inode to indicate that a read operation is in place. When the I/O for the read is complete, the thread updates the access time field in the inode and releases the lock.

When a thread attempts to get the inode lock and determines that a read is in progress, instead of blocking on the inode lock, it calls a kernel service to pre-read the data it requires. Once the pre-read has completed, the thread will attempt to obtain the inode lock to update the access time field. This solution reduces the amount of time a thread is required to hold the inode lock when

performing a read operation, therefore allowing greater throughput on multiple concurrent reads.

If a thread attempting to read from a file cannot get the inode lock and there is a write operation in progress, then the thread blocks on the lock waiting for the write operation to complete.

2.11 Increase in the Upper Limit of Trace Buffer (4.3.1)

The current upper limit for a trace buffer produced by the trace command with -T option is around 55 MB on SMP systems. This only allows a few seconds of a performance benchmark execution to be recorded. As a consequence, only a fraction of the data needed can be collected, since the benchmarks can take up to several minutes.

For AIX Version 4.3.1, the upper limit is increased to the size of a segment and two segments when double buffering is used or as close to that as possible, allowing the amount of information collected to be more complete and useful.

2.12 Kernel Scaling Enhancements (4.3.2)

With increasing demands being placed on machines acting as busy network servers, it is possible that in certain situations some kernel resources may become exhausted. As machines supporting larger amounts of physical memory, adapters, and devices are introduced, it makes sense for the kernel to be able to use larger resource pools when required. Therefore, the `crash` utility, used for examining system images and the running kernel, is enhanced to understand the new increased system resources.

The following sections describe the major enhancements.

2.12.1 Network Memory Buffer Pool

The kernel allocates memory from the network memory buffer pool, commonly called the *mbuf* pool, to be used as buffers by the networking subsystem. The size of the mbuf pool is a tunable parameter and is changed using the `thewall` option of the `no` command.

2.12.1.1 Network Memory Buffer Pool Size Increase

The maximum size of the mbuf pool is now hardware dependent. Previous versions of AIX allocated the mbuf pool from the kernel heap. AIX 4.3.2 now uses a dedicated memory segment for the mbuf pool on most machines, and four contiguous memory segments on CHRP machines. This allows a maximum mbuf pool of 256 MB on most machines and 1 GB on CHRP hardware. The kernel will allocate an amount of virtual memory equal to one half the amount of physical memory, or the maximum value allowed for the hardware type, whichever is smaller. For example, on a machine with 128 MB of memory, the default value of `thewall` will be 64 MB. On a CHRP machine with 16 GB of memory, the default value will be 1 GB.

The larger mbuf pool will allow greater network throughput on large SMP systems.

2.12.1.2 Network Memory Buffer Pool Allocation Algorithm

The algorithm used by the net_malloc kernel service for allocating buffers of various sizes has been changed. The high-water marks for various buffer sizes have been increased, particularly the 2 KB size used by the Ethernet and token ring device drivers.

Requests for memory buffers between 16 KB and 128 KB are now rounded to the nearest power of 2. Allocations in this range were rounded to the nearest page size on prior versions of AIX. This change reduces the number of different sizes of buffers available, which in turn reduces the number of free lists the algorithm is required to manage.

The method used by net_malloc to maintain free list information has been changed. On prior versions of AIX, each CPU maintained a free list for each different size of buffer. There is now one central free list for each buffer size between 16 KB and 128 KB in size. Each CPU still maintains a free list for each of the smaller buffer sizes. This change minimizes the amount of memory used by unallocated large buffers, while at the same time still allowing each CPU to maintain a list of smaller buffers for faster allocation.

2.12.1.3 Network Memory Buffer Pool Statistics

The kernel maintains usage statistics for the buffers allocated from the network memory buffer pool. The information contains details of the number of buffers of each size, and for each size, information on the number of buffers in use and the number of failed requests. This information can be displayed using the `netstat -m` command. In addition to maintaining information indexed by buffer size, the kernel also maintains information indexed by the purpose the buffer is being used for. The type indexed information can be expensive to maintain, so AIX Version 4.3.2 has introduced a method to disable it.

The new extendednetstats network variable, which is altered using the `no` command, determines whether the by-type statistical information should be collected by the kernel. The variable has a default value of 1, which means that the kernel will collect the information. In order to improve networking performance, the default AIX installation disables collection of this information by changing the value of extendednetstats to 0. This is performed in the file /etc/rc.net that is run during system start up. If you want to view the by-type statistics, you should comment out the section of /etc/rc.net that changes the value of extendednetstats.

The following section is near the end of the file and appears as:

```
######################################################################
# This disables extended netstat statistics for performance
# reasons.  To have extended netstat statistics enabled on
# future reboots, comment out the following three lines.
######################################################################
if [ -f /usr/sbin/no ] ; then
        /usr/sbin/no -o extendednetstats=0 >>/dev/null 2>&1
fi
```

The information collected when extendednetstats is set to a value of 1 is displayed near the end of the output produced by the `netstat -m` command. An example of the output is shown below.

```
# netstat -m
```

```
61 mbufs in use:
32 mbuf cluster pages in use
143 Kbytes allocated to mbufs
0 requests for mbufs denied
0 calls to protocol drain routines
0 sockets not created because sockthresh was reached

Kernel malloc statistics:

******* CPU 0 *******
By size       inuse     calls failed    free   hiwat   freed
32              201       749      0      55     640       0
64               98       325      0      30     320       0
128              61       257      0      35     160       0
256             135      7324      0      25     384       0
512             110       937      0       2      40       0
1024             35       276      0       5     100       0
2048              0       482      0       6     100       0
4096             34        68      0       2     120       0
16384             1         1      0      18      24       7
32768             1         1      0       0     511       0

By type       inuse     calls failed  memuse  memmax  mapb
mbuf             61      6952      0   15616   21760     0
mcluster         32      1123      0  131072  141312     0
socket          212      1034      0   76480   77536     0
pcb              77       457      0   14144   14400     0
routetbl         19        21      0    3008    3648     0
fragtbl           0       132      0       0      32     0
ifaddr            6         6      0    1472    1472     0
mblk             23       196      0    4992    5504     0
mblkdata          2       125      0     512    2816     0
strhead          11        19      0    3232    3680     0
strqueue         18        38      0    9216   10752     0
strmodsw         22        22      0    1408    1408     0
strosr            0        17      0       0     256     0
strsyncq         26        88      0    2752    3200     0
streams         138       153      0   15520   16096     0
file              1         1      0     128     128     0
kernel table     14        14      0   50016   50016     0
locking           3         3      0     384     384     0
temp              9        15      0    5568   10112     0
mcast opts        0         2      0       0     128     0
mcast addrs       2         2      0     128     128     0

Streams mblk statistic failures:
0 high priority mblk failures
0 medium priority mblk failures
0 low priority mblk failures
#
```

When extendednetstats is set to a value of 0, the by-type information is not displayed.

2.12.2 Expanded Kernel Heap

AIX version 4.3.2 has added a dedicated memory segment for the kernel heap. The kernel heap is a general memory allocation area and is used to store the dynamic data structures created and used by the kernel, kernel extensions, and device drivers.

This enhancement has increased the maximum size of the heap by an additional 256 MB. The kernel heap was previously located in the upper part of segment 0 that was not used for kernel text pages. The maximum size of the heap depended on the size of the kernel image that was running. The original location is now used as an overflow buffer and is only used if the dedicated 256 MB kernel heap segment becomes exhausted.

2.12.3 Larger Pipe Buffer Pool

The increased size of the kernel heap means that more space can be used for pipe buffers by the kernel, therefore increasing the number of simultaneously open pipes. As with the mbuf pool, the amount of kernel virtual memory reserved for the pipe buffer pool depends on the total amount of physical memory. The system will allocate an amount of virtual memory equivalent to one eighth of the physical memory, or 64 MB, whichever is smaller, with a minimum allocation of 16 MB. Of the memory reserved for use as pipe buffers, 1 MB is pinned in physical memory for faster initial buffer allocation. The size of each individual pipe buffer remains the same as on previous versions of AIX, at 4 KB.

2.12.4 Inter-Process Communication Identifier Enhancement

The limits of the maximum number of IPC identifiers have been increased, as provided in Table 6.

Table 6. IPC Identifier Limits

Value	Previous Limit	New Limit
Message queue IDs	4096	131072
Semaphore set IDs	4096	131072
Shared memory IDs	4096	131072

In addition to increasing the number of identifiers available, AIX Version 4.3.2 has also implemented a new algorithm to handle the ipcget() routine.

The previous implementation used a sequential search algorithm for traversing the list of IPC identifiers. For a table size of N, the algorithm resulted in N operations for a search miss and N/2 operations for a search hit. Although this is very simple, it does not scale very well. The algorithm has been replaced with a hash table implementation that is better matched to the larger number of IPC identifiers now available.

The IPC support commands, such as `ipcrm` and `ipcs`, have also been changed to take account of the increased limits.

2.12.5 Boot Logical Volume Scaling

In AIX 4.3.2, the boot logical volume is expanded to enable system configurations with up to 1,000 devices.

As more and more devices are added to a system, the ODM object classes containing device configuration data will grow larger and larger. It is possible that the RAM file system used in the initial stages of booting will not be large enough for the larger ODM files. The existing boot process accounts for this when booting from disk by dynamically expanding the RAM file system based on the amount of system memory. The increased savebase area will not fit on the boot logical volume when adding large amount of devices.

2.13 Scheduler Enhancements (4.3.2)

The scheduler on AIX Version 4.3.2 has been enhanced to increase the impact of using the `nice` command to alter the priority of a thread. The following sections

explain how the changes have been implemented and show sample results by comparing the new scheduler with the previous version.

2.13.1 Thread Priority Calculation Changes

All threads on the system have a priority value between 0 and 127, with 60 being the default initial value. As a thread runs and consumes CPU time, the priority value changes numerically as a function of CPU time recently used. A numerically higher number represents a less favored priority. A thread that started with the default priority of 60 may have an instantaneous priority in the range 60 to 126. The value of 127 is reserved for the wait process. The scheduler runs all threads at priority N that are marked as runable before it runs any threads at priority N+1, thus favoring threads using less CPU time.

The `nice` and `renice` commands, and the setpriority system call, can be used to change the initial priority of a thread by a given delta. The delta can be in the range -20 to 19. Thus a thread can have an initial absolute priority in the range 40 to 79. The absolute initial priority, or *nice* value, is included in the calculation of a threads priority. This introduces the idea of relative priority between threads.

In addition to the nice value, the `schedtune` command can be used to fine tune the method used to calculate the new priority. The calculation also has parameters that scale the relative importance of recent CPU utilization (the -r option to schedtune, shown as sched_R) and historical CPU utilization (the -d option to schedtune, shown as sched_D). Both the sched_R and sched_D parameters have a default value of 16.

On versions of AIX prior to 4.3.2, thread priority is calculated using the following algorithm:

- Once per clock tick: cpu = cpu + 1 for the currently running thread, limited to a maximum of 120
- Priority calculation: (cpu * sched_R) / (2 * 16) + nice, limited to a maximum of 126
- Once per second ageing of all threads: cpu = cpu * sched_D / 32

With the default values in place, this equates to:

- Priority calculation: cpu / 2 + 60
- Once per second ageing of all threads: cpu = cpu / 2

The scheduler on AIX 4.3.2 now uses the following algorithm to calculate thread priorities.

- Once per clock tick: cpu = cpu + 1 for the currently running thread, limited to a maximum of 120
- Priority calculation part 1:

  ```
  xnice = (nice > DEFAULT_NICE) ? (2*nice) - 60 : nice
  ```

- Priority calculation part 2 (limited to a maximum of 126):

  ```
  p = (cpu * sched_R * (xnice + 4))/(32*(DEFAULT_NICE + 4)) + xnice
  ```

- Once per second ageing of all threads: cpu = cpu * sched_D / 32

The `nice` value has a much greater impact on the priority of a thread. It is now included in the calculation as a multiplier of the recent CPU usage in addition to the use as a constant factor.

With the default values of 16 for sched_R and sched_D, and 60 for `nice` and DEFAULT_NICE, the priority calculation equates to:

- Priority calculation: cpu / 2 + 60
- Once per second ageing of all threads: cpu = cpu / 2

Although the algorithm has changed, the default values provide an identical function.

2.13.2 Sample Results of Altering Nice Value

The following tables list the results of changing the nice value of a thread on two identical machines; one running the old algorithm, and the other running the new algorithm. In each case, the tables list the percentage of CPU time delivered to one thread that has been *niced* by the indicated delta, which is in competition with varying numbers of default priority threads. All threads were running the same CPU bound application.

From comparison of the values in Table 7 and Table 8, it can be seen that the effect of a positive nice delta on a thread has been enhanced. Take, for example, a thread running with a nice delta of 19 in competition with one default thread. Previously, the niced thread would receive 41 percent of the CPU, with the default thread receiving the remaining 59 percent. With the new algorithm, the niced thread has been reduced to 15 percent, with the default thread increasing to 85 percent.

In addition, the effect of a negative nice delta has been increased. A thread running with a nice delta of -20 competing against 31 default threads now receives 32 percent of the CPU, compared with 23 percent under the previous

algorithm. Correspondingly, the CPU delivered to each of the remaining 31 default threads has decreased from 2.5 percent to 2.2 percent.

Table 7. Old Priority Algorithm, sched_R and sched_D Defaulted to 16

nice delta	**Number of threads running (1 niced, others default)**							
	1	2	3	4	5	8	16	32
-20	100	60	47	40	36	30	25	23
-15	100	58	43	36	32	26	21	17
-10	100	55	40	32	28	21	15	13
-5	100	53	37	29	24	17	11	8
0	100	50	33	25	20	13	6	3
5	100	48	30	21	16	7	2	0
10	100	45	27	17	12	4	0	0
15	100	43	23	14	8	0	0	0
19	100	41	21	11	4	0	0	0

Table 8. New Priority Algorithm, sched_R and sched_D Defaulted to 16

nice delta	**Number of threads running (1 niced, others default)**							
	1	2	3	4	5	8	16	32
-20	100	78	60	52	48	42	36	32
-15	100	69	51	43	39	33	26	24
-10	100	60	45	37	32	25	19	16
-5	100	55	39	31	25	19	12	9
0	100	50	33	25	20	13	6	3
5	100	42	24	16	11	4	0	0
10	100	32	17	8	4	0	0	0
15	100	22	11	3	0	0	0	0
19	100	15	6	0	0	0	0	0

It can be seen from the values in Table 9 and Table 10 that decreasing the value of the sched_R parameter makes the effect of the nice delta even greater. By reducing the value of sched_R, the priority algorithm places less emphasis on recently used CPU time. Consequently, the thread with a negative nice delta receives even more CPU time.

Table 9. Old Priority Algorithm, sched_R=8

nice delta	Number of threads running (1 niced, others default)							
	1	2	3	4	5	8	16	32
-20	100	78	60	55	52	47	43	41
-15	100	68	54	48	44	39	34	32
-10	100	60	47	40	36	30	25	23
-5	100	55	40	32	28	21	15	13
0	100	50	33	25	20	13	6	3
5	100	45	27	18	12	3	0	0
10	100	40	20	10	4	0	0	0
15	32	14	2	0	0	0	0	0
19	100	24	8	0	0	0	0	0

Table 10. New Priority Algorithm, sched_R=8

nice delta	Number of threads running (1 niced, others default)							
	1	2	3	4	5	8	16	32
-20	100	98	96	94	92	86	70	59
-15	100	84	68	58	53	50	43	41
-10	100	68	52	46	41	36	28	27
-5	100	57	42	35	31	23	18	13
0	100	50	33	25	20	13	6	3
5	100	37	18	9	4	0	0	0
10	100	17	6	0	0	0	0	0
15	100	2	0	0	0	0	0	0
19	100	0	0	0	0	0	0	0

Take Note

Changing system scheduling parameters using the `schedtune` command can cause unexpected results, particularly if more than one system parameter is changed. See *RS/6000 Performance Tools In Focus*, SG24-4989 for more information on the `schedtune` command, its parameters, and other performance monitoring and tuning tools.

Chapter 3. 64-Bit Enablement

This chapter covers the introduction of 64-bit systems and the support provided in AIX Version 4.3 for these systems. In the first section, an introduction to 64-bit architectures and its benefits is provided, including the hardware and software aspects of 64-bit implementations. The design chosen for RS/6000 systems and the AIX operating system is also explained.

The second section describes the changes made to the core operating system that are necessary to run AIX on 64-bit hardware and enable 64-bit applications. These changes include modifications to the basic application development tools like compiler, linker, and debugger, and other tools that operate on object files.

3.1 Introduction to 64-Bit Computing

The following sections describe some of the features of the new 64-bit environment.

3.1.1 64-Bit Architecture and Benefits

From an operational point of view, an architecture is said to be 64-bit when:

- It can handle 64-bit-long data; in other words, a contiguous block of 64 bits (8 bytes) in memory is defined as one of the elementary units that the CPU can handle. This means that the instruction set includes instructions for moving 64-bit-long data and instructions for performing arithmetic operations on 64-bit-long integers.
- It generates 64-bit-long addresses, both as effective addresses (the addresses generated and used by machine instructions) and as physical addresses (those that address the memory cards plugged into the machine memory slots). Individual processor implementations may generate shorter physical addresses, but the architecture must support 64-bit addresses.

The benefits of 64-bit architectures can be summarized as follows:

- Extended-precision arithmetic. The ability to use very long integers in computations is a feature that can be very useful in specialized applications.
- Access to large data sets. The ability to create and maintain very large file systems is increasingly important for many users. In particular, data warehousing applications, scientific, and multimedia applications frequently require these features.
- Large address spaces. A 64-bit architecture has the capability of addressing huge address spaces. You should realize that the step to 64-bits is much more than just a doubling of 32-bits. In terms of addressability, it represents a four billion-fold increase. With clever exploitation, these large address spaces can result in spectacular performance improvements or gains in productivity through simplified programming of very large technical problems.

The ability to handle large address spaces is considered the greatest potential benefit for users since, in the future, complex applications such as large databases, large numeric applications, and multimedia environments will need to manage and operate on larger data sets. Since internal memory is much faster than most storage devices, the ability to fetch and keep more data in memory,

where it can be directly manipulated, should provide dramatic performance improvements. Table 11 shows the size of the address spaces that can be managed as a function of the length of the address that the CPU generates.

Table 11. Size of Address Space as a Function of Address Length

Address Length	Flat Address Space
8-bit	256 Bytes
16-bit	64 Kilobytes
32-bit	4 Gigabytes
52-bit	4000 Terabytes
64-bit	16,384,00 Terabytes

3.1.2 64-Bit Challenges

As previously mentioned, 64-bit architectures can prove advantageous in many areas of application. It should be noted, however, that these advantages can come at a cost. Extra addressability must be accompanied by very large amounts of system memory to work effectively. Applications compiled in 64-bit mode also consume more disk space than their 32-bit equivalents, adding to the cost of system storage. It is also important to remember that 32-bit applications that cannot or do not take advantage of the features previously mentioned should remain as 32-bit binaries. If compiled in 64-bit mode without change, they will probably not see any performance improvement. It is possible that the application will run slightly slower due to cache effects and longer code-path length.

Although the vast majority of current applications will not fully utilize the functions and capabilities of a 64-bit architecture, the applications of the near future will increasingly view 32-bit technology as a limiting factor.

3.1.3 64-Bit PowerPC Design

The PowerPC architecture is, by its nature, an open, extendable design. There is nothing in the chip architecture itself that would affect binary compatibility as you migrate across different PowerPC implementations. The PowerPC processor architecture was defined from the start as a 64-bit architecture that is a superset of the 32-bit architecture implemented in the 601, 603, and 604 processors.

An important aspect of the 64-bit version of PowerPC is its binary compatibility with the previous PowerPC processors. From the standpoint of the 32-bit and 64-bit specifications, there are a few differences, as shown in Figure 4. The number of CPU registers (the basic storage cell where the CPU stores the data on which it performs its computations) remains the same, but these registers are now 64 bits long instead of 32 bits. A few other control registers also move from 32 to 64 bits in length. Note that the floating point registers do not change in size, since they already conform to industry standards for floating-point that require 32- or 64-bit-long data.

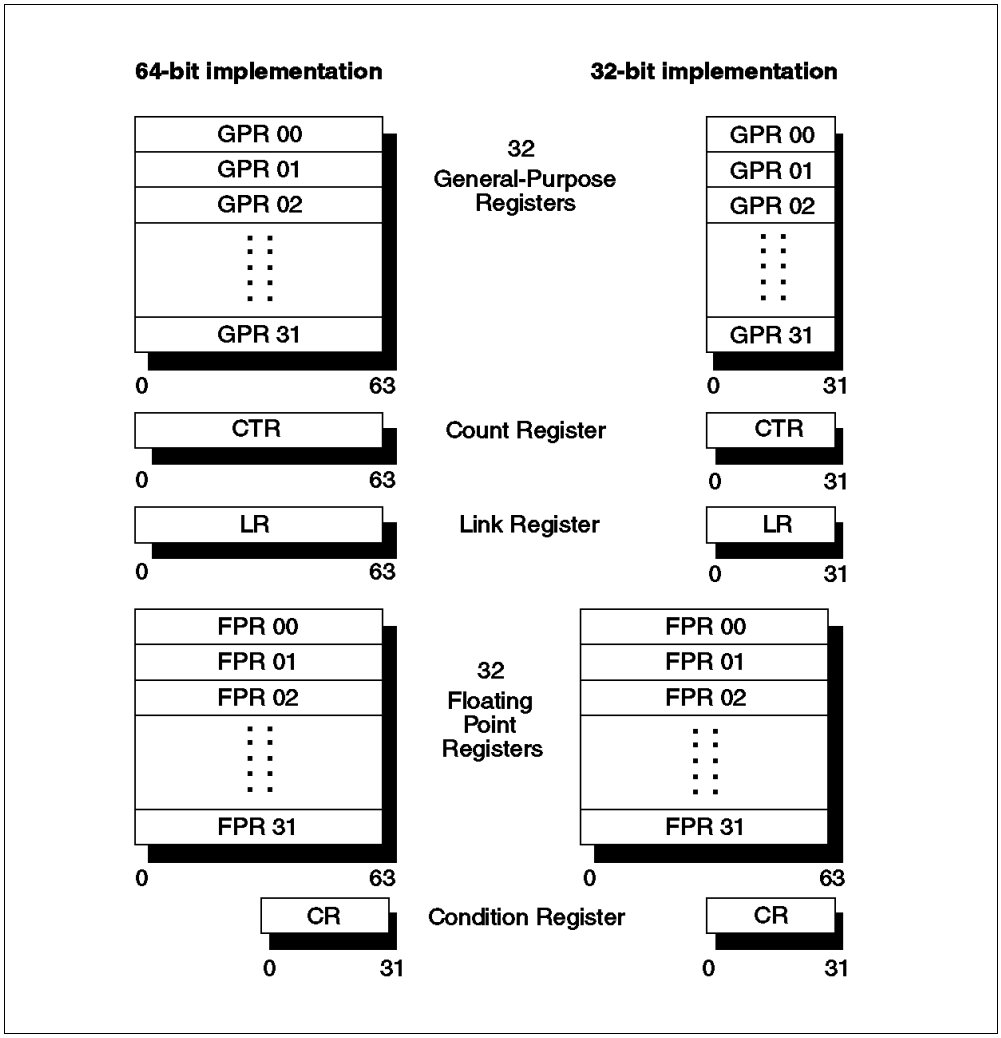


Figure 4. Register Implementation of 32-Bit and 64-Bit PowerPC Processors

In the 64-bit implementation of PowerPC, existing machine instructions do not change drastically. Many instructions simply work the same in 64-bit mode. That is, they can manage 64-bit long data and use/generate 64-bit-long addresses. New instructions, that were not implemented in the previous PowerPC chips, are included for the handling of 64-bit data.

A 64-bit PowerPC can also work in 32-bit mode. In this way, any application that currently runs on the 32-bit PowerPCs can run unchanged. For example, arithmetic instructions running in 32-bit mode operate only on the lower-half of the CPU register involved and consider only that half of the register in the result. 32-bit addresses are handled in the same way.

The virtual address space is the amount of virtual memory that an application can address independent of the size of the physical memory actually installed in the machine on which it is running. Figure 5 shows a simplified representation of the virtual address space that the PowerPC architecture can manage in 32-bit and in 64-bit mode. As shown, the 32-bit implementation is already capable of addressing a very large (2^{52} bytes, refer also to Table 11) address space. The 64-bit implementation goes up to 2^{80} bytes (a huge number that signifies nearly

one trillion terabytes. Other 64-bit architectures currently on the market mainly address a 2^{64} bytes-wide virtual address space.

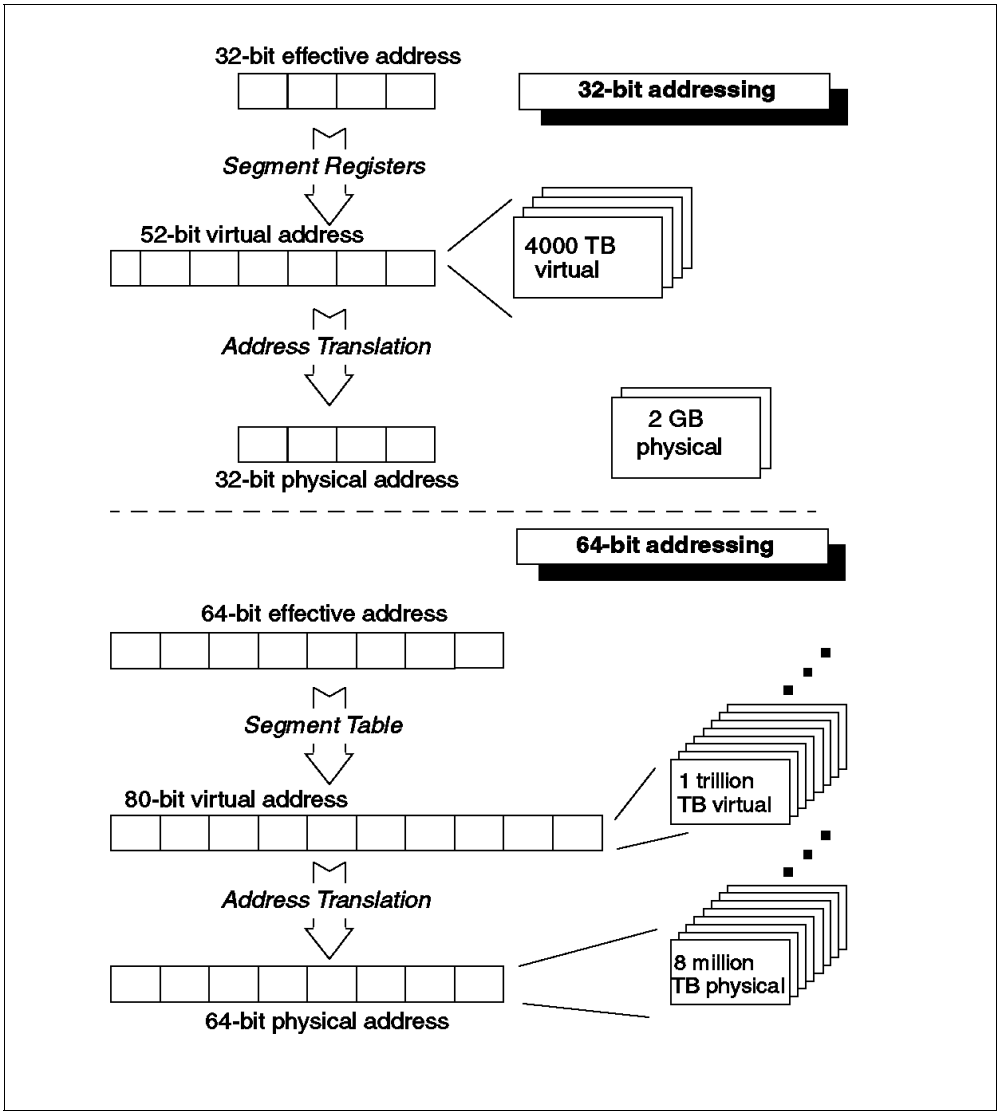


Figure 5. Comparison of Address Translation in 32-Bit and 64-Bit Mode

3.1.4 AIX 64-Bit Design Criteria

It is important to note that the 64-bit execution environment for application processes is an upward-compatible addition to AIX capability, not a replacement for the existing 32-bit function. The design IBM chose for 64-bit AIX allows existing 32-bit applications to run with 64-bit applications with no changes, thus protecting the investment users have made in their current applications. Users can take advantage of the features of 64-bit AIX when business needs dictate.

AIX 64-bit support is intended to be run on 64-bit hardware systems, not simply any system containing a 64-bit PowerPC processor. A 64-bit hardware system is one that is based on the CHRP architecture and identifies 64-bit properties for the processor(s) and processor host bridges (PHBs) in configurations with memory addressing greater than 32 bits.

For AIX, and the applications that run on AIX, the 64-bit PowerPCs have two important attributes. They are very fast when running as 32-bit processors, and they offer the opportunity of running a 64-bit environment. AIX 4.3 exploits these attributes separately. There are two different execution environments in AIX 4.3, the 32-bit execution environment and the 64-bit execution environment. These two environments are only available for 64-bit hardware. There is no 64-bit execution environment on 32-bit hardware.

Generally, the AIX 4.3 kernel remains 32-bit, and only selected parts, such as the Virtual Memory Manager, are upgraded to be aware of the 64-bit address space. This means that the number of AIX kernels remains two (uniprocessor and multiprocessor).

Although there were a number of choices regarding the basic data types of 64-bit processes, the choice that was made by the Aspen working group, formed by XOPEN, and a consortium of hardware vendors is called LP64, short for Long-Pointer 64. This is commonly also called the 4/8/8 model, which stands for the Integer/Long/Pointer type sizes. The benefit of this configuration is that it provides 64-bit addressing with 8-byte pointers, large object support (8-byte longs), and backward compatibility (4-byte integers). Other alternatives included an 8/8/8 solution, called ILP64, and the LLP64 that were not adapted. Obviously, the choice for AIX was LP64.

3.2 64-Bit Core Design

As stated, the kernel of the AIX operating systems remains 32-bit. To allow 64-bit applications to run and use memory in the 64-bit address space, extensions to the kernel were introduced. This is discussed in Section 3.2.2, “System Calls” on page 37. A fundamental change was also introduced in the object module format that basically enables executables to overcome the size limit of 2^{32} bytes. The subsection on XCOFF addresses these changes.

The implications for device drivers of 64-bit and 32-bit devices in a 64-bit execution environment are explained in Section 3.2.4, “Device Drivers” on page 49.

The loading of 64-bit modules is shared between kernel-space code and user-space code. The changes to the loader are discussed in 3.2.5, “Loader” on page 51.

The Virtual Memory Manager is one of the key parts of the operating system when it comes to large address space and mapping of virtual to real memory. The consequences are summarized in the last part of this section.

3.2.1 Segment Register Mapping

Since the 64-bit hardware uses 64-bit effective addresses while operating in 64-bit execution mode, keeping sixteen segment registers would have required increasing the size of segments proportionally to allow for the increased address space. This is less useful to applications than increasing the number of segments in a process' address space. Therefore, segmentation on the 64-bit hardware remains in units of 256 megabytes. In 64-bit execution mode, the sixteen segment registers used for 32-bit addressing are replaced with a segment table (analogous to the hardware page table) that contains up to 256 of the most

recently used mappings of segment numbers to segment ID for the process' user address space.

The architecture also allows a Segment Lookaside Buffer (SLB) to hold a smaller number of recently used segment number to segment ID mappings for the process' user address space. For 32-bit mode, part of the SLB can be used by the hardware to represent the sixteen segment registers.

64-bit processes are limited to 60-bit effective addresses. This is a convenient number for the VMM since 2^{60} represents effective segment IDs up to 32-bits, which fits into one register in the kernel. The 60-bit effective address space will be sparsely instantiated up to the rlimit values for the process or up to some limitation imposed by overall system resources. The choice of address space layout was made to reduce the number of constraints on the size of each area and to allow for future expansion and new uses for address spaces. At some future date, the address space may be expanded to more than 60 bits.

The first sixteen segment numbers are freed as much as possible from default system use and left for application use. This has many advantages. It makes finding incorrect use of 32-bit addresses in 64-bit processes easier; it allows 32-bit and 64-bit applications to share memory at the same addresses in each process, and it allows 64-bit applications to be specially designed to work with interfaces that do not understand 64-bit addresses. For example ioctl().

Segment number 0 is used for the first kernel segment in user mode. It is read-protected in user mode, except for the system call (SVC) tables, svc_instruction code, and system configuration structure.

Segment number 1 is used for the second kernel segment in user mode. It contains the 64-bit system call (SVC) tables.

Segment number 2 is still used for the process private segment. No program-allocatable data is contained in this segment. The address of the user block is at the same location in 32-bit and 64-bit processes.

The segment numbers from 3 to 12 and segment number 14 are reserved for application use by the shmat() and mmap() interfaces.

Segment numbers 13 and 15 are reserved for the new user space loader (see 3.2.5, “Loader” on page 51) used at exec() and load() time. These segments are global, read-only, and are always mapped in.

The address space from segment numbers 0xA0000000 to 0xEFFFFFFF are reserved for future system use. These segments numbers are not given out in response to user requests.

Table 12. Address Space Layout in User Mode

Segment Number	Use in 64-Bit Mode	Use in 32-Bit Mode
0	Kernel	Kernel
1	Kernel	User
2	Process Private	Process Private
3	shmat/mmap	Available for User

Segment Number	Use in 64-Bit Mode	Use in 32-Bit Mode
4-0xC	shmat/mmap	shmat/mmap
0xD	Loader use	Shared libraries
0xE	shmat/mmap	shmat/mmap
0xF	Loader use	Shared lib data
0x10-0x6FFFFFFF	Application text, data, Bss, heap	N/A
0x70000000-0x7FFFFFFF	Default shmat/mmap	N/A
0x80000000-0x8FFFFFFF	Private load	N/A
0x90000000-0x9FFFFFFF	Shared library text and data	N/A
0xA0000000-0xEFFFFFFF	Reserved for system use	N/A
0xF0000000-0xFFFFFFFF	Application stack	N/A

The following items give more detail on the use of the various segments in 32-bit and 64-bit mode:

- Process private data

  Segment number 2 continues to contain the process private segment. This segment is substantially different for 32-bit and 64-bit processes. The process private segment continues to contain errno, errnop, environ, the top_of_stack structure, and the exec() arguments. It also contains the user structure, primary user thread structure, and the primary kernel thread stack.

  It does not contain the user thread stack or any user data. The user thread structures (other than the primary) is moved to the kernel thread stack segments for both 32-bit and 64-bit processes. The errno, errnop, and environ locations are different in 32-bit and 64-bit mode. The top_of_stack structure is reformatted for the 64-bit values for 64-bit processes.

  The errno, errnop, environ, the top_of_stack structure, and the exec arguments (all the user accessible data) are kept in the lowest one megabyte of this segment and are user-modifiable. All data above this in the segment is read-protected from user access. The segment table, adspace, and segstate structures for the process are allocated from a region above the first megabyte in the segment. The segment table is pinned. The region above these adspace structures and below the primary kernel thread stack is used for the overflow heap for the per-process loader heap.

- Executable text area

  The text area starts in segment number 16. All segments from the start of the executable, through, and including, the loader section of the executable, are mapped in. No segments beyond the segment containing the loader section of the executable are mapped in the address space. The text segments are mapped read-only if the underlying file system supports mapping (JFS, NFS, CD-ROM file system, AFS, and DFS support mapping). Otherwise, the text section and the loader section are copied to working storage segments.

- Executable data and BSS

  Following the text segments are working segments containing the executable's initialized data and BSS (uninitialized data) areas. The data area is vm_map()ed from the executable and relocated.

- Heap

  The break value for the process is initialized just above the BSS area. As the heap is grown (with brk() and sbrk() calls), new segments are allocated above the current segment containing the current break value up to the segment containing the new break value. The heap is not allowed to grow into segment number 0x70000000 or beyond any shmat or mmap segment.

- shmat and mmap segments

  Starting at segment number 0x70000000, shmat and mmap segments are allocated if no address is specified. Segments are allocated in increasing order in the first available segment at, or after, segment number 0x70000000. The shmat and mmap segments are placed where requested if the segment number is available and less than segment number 0x80000000.

- Explicitly-loaded modules

  Explicitly-loaded modules (using the load() system call) were previously loaded into the heap of the process. This creates complexity in dealing with heap expansion/contraction and explicit loads.

  Explicitly-loaded objects are now loaded into separate segments starting at segment number 0x80000000. Segment numbers are allocated in increasing order in the first available segment number at, or after, segment number 0x80000000.

  Explicitly-loaded objects are limited to segment numbers 0x80000000 to 0x8FFFFFFF. To reduce segment table faults, multiple loaded modules are vm_map()ed into these working storage segments. The data for the loaded modules is loaded into these segments (and relocated) also.

- Shared library text and data segments

  Starting at segment number 0x90000000, shared library text and data segments are allocated. These segment numbers are allocated globally (at the same address in all 64-bit address spaces) to allow sharing of the segments (in the case of text segments) and vm_map()ing of the segments (in the case of data segments). Global shared library text and data segments are maintained by the loader using the current method. Shared library text and data segments are limited to segment numbers 0x90000000 to 0x9FFFFFFF.

- User stack

  The initial user stack is allocated starting at the top of segment number 0xFFFFFFFF and will consume additional segment numbers below this as the stack grows. New segment numbers will be allocated at stack growth time. Only the segment number below the segment number containing the current top of stack is allocated. References to more than one segment number away from the current top of the stack are treated as wild references.

  Note: This restricts local variable allocation to less than 256 megabytes of the total allocation in each function. The stack growth is limited to segment numbers ranging from 0xF0000000 to 0xFFFFFFFF as segment numbers in the range 0xA0000000 to 0xEFFFFFFF are reserved for future system use.

- Big data programs

  The maxdata field is used for 32-bit programs to indicate that the heap should be placed in a different place where a larger heap can be accommodated. In 64-bit processes, the address space will always be able to accommodate a large heap, so no indication is necessary. The maxdata and maxstack fields, if set, are used to set the soft rlimit value for 32-bit and 64-bit applications. If the limit specified is greater than the hard rlimit values, the exec() will fail.

3.2.2 System Calls

Because the AIX kernel remains 32-bit, the interfaces to the various system calls must be through the types and structures of 32-bit mode C. 64-bit mode applications, however, are compiled with 64-bit mode types and structures. This section explains how the different types are communicated to the kernel.

On a 64-bit PowerPC, AIX will run both 32-bit-mode processes and 64-bit-mode processes. The problem is that 64-bit applications compiled with the same AIX header files that 32-bit processes use (but compiled for 64-bit execution mode) build data structures, arrays, and scalars using the rules of 64-bit C, though the kernel expects data structures, arrays, and scalars that match the ones built by 32-bit applications using 32-bit C. If a 64-bit application builds, for example, an ioveC structure containing an address and a length, this structure cannot be passed directly to the kernel's kreadv() system call routine because that routine cannot interpret the 64-bit address and 64-bit length.

Clearly, some code must be placed between the 64-bit application's system calls and the 32-bit kernel (see Figure 6). AIX takes advantage of the following two features in implementing this interface code:

- The 32-bit and 64-bit name spaces are completely separate, and calls from 64-bit applications are never resolved to 32-bit entry points. Specifically, the calls from 64-bit applications to traditional system call entry points should not be resolved to the 32-bit entry points of the same names exported by the kernel.
- Since UNIX no longer makes a distinction between system calls and subroutines, it is no longer necessary to strictly follow the old UNIX semantics. If a caller passes a bad address to any system-supplied subroutine (whether system call or not), it is permissible to end the calling process, as will happen if a library routine dereferences an invalid pointer. This means that routines, traditionally considered to be system calls, can reside in subroutine libraries, and almost all of the system call interface code needed for 64-bit processes can be placed in a user-mode library such as libc.a.

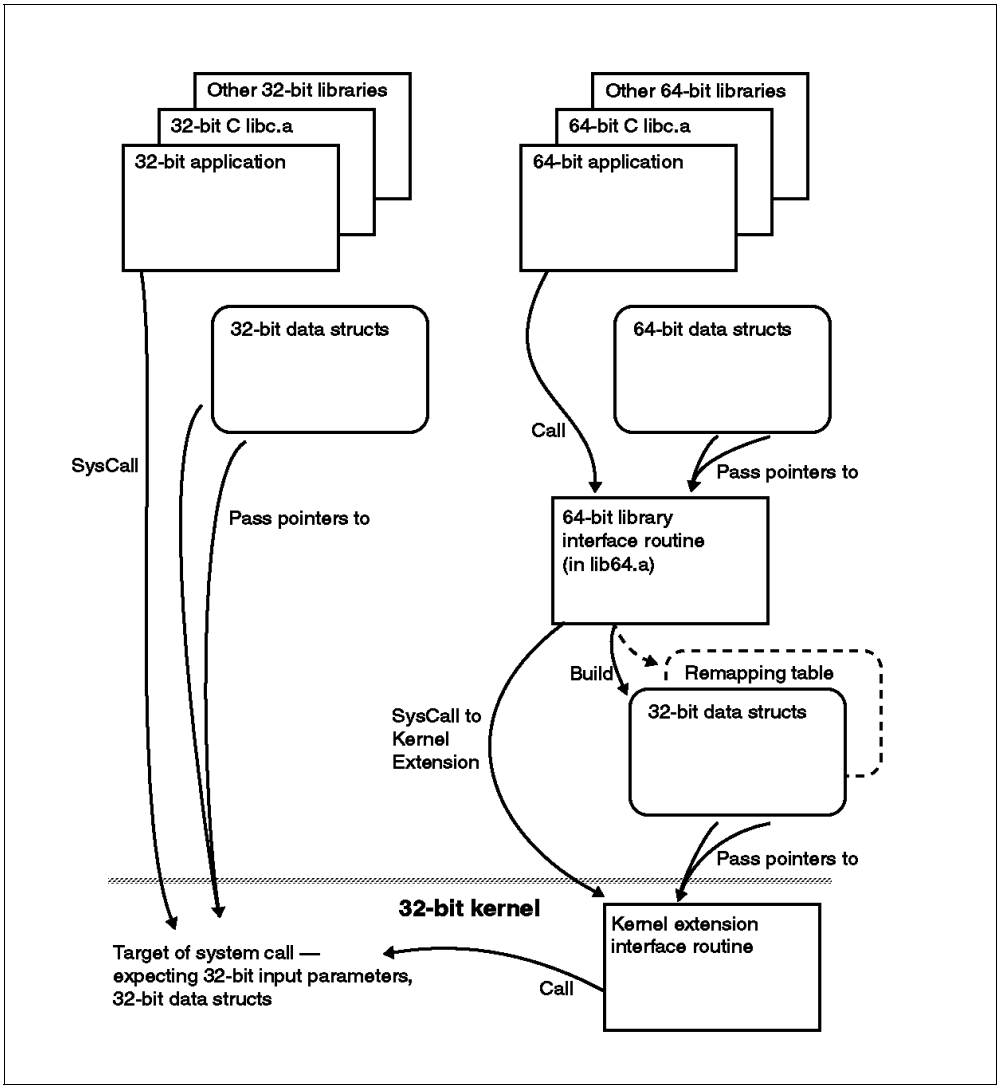


Figure 6. Interfacing 64-Bit Processes to a 32-Bit Kernel

Each system call that is exported by the 32-bit kernel is represented in 64-bit mode by a 64-bit-mode library routine in libc64.a that is the call target of what would have been (in 32-bit mode) a system call. These library routines handle any necessary reformatting of data to communicate with the 32-bit system call routines in the kernel. In many cases, they build a remapping table that tells how the required portions of the 64-bit address space should be reflected in a 32-bit map for the kernel.

On the kernel side, in 32-bit mode, a kernel extension routine is added for each of the system calls supported in 64-bit mode. These routines are invoked from the 64-bit library routines through AIX's syscall interface. They accept the reformatted data from the library routines and perform any necessary remapping of addresses using data supplied by the library routines. The kernel can then properly *see* data structures, buffers, and so on in the user spaces that are referred to in the call.

A typical system call involves several pieces of code located in various places in the system. The names of these various pieces are all derived from the original

name of the 32-bit system call. If a particular 32-bit system call is named bletch(), then:

- The name of the C language routine in libc64.a that intercepts 64-bit calls to bletch() is precisely that: bletch. Note that the kernel does not export a symbol named bletch to 64-bit-mode processes, so all 64-bit calls to that label reach the library routine.
- The C language routine library, at the point where it needs to invoke the kernel extension routine for this system call, calls __bletch() (two leading underscores). This is an assembly-language *glue* routine reached through the branch-and-link instruction. The name of its actual entry point will be .__bletch (leading period).
- The glue routine, still in the 64-bit library, loads an entry from the TOC and issues a system call instruction. The TOC entry is assembled as a pointer to bletch64 (no prefix, suffix 64).
- The kernel extension routine is named bletch64(), and this is the name that the kernel extension exports as a syscall. The kernel extension routine will itself call bletch() the existing 32-bit kernel service.

The reason for the two underscores is for conformance with ANSI C, which reserves names not starting with two underscores (or one underscore and a capital letter) for the user, except for existing non-conforming UNIX names that are *grandfathered*. All external symbols in libc64.a must begin with two underscores.

3.2.2.1 64-Bit to 32-Bit Data Reformatting

The goal of the 64-bit interface is to make it appear to the kernel that the system service request came from a 32-bit program. To this end, the width of any data passed across the interface in either direction must be adjusted to match the expected size.

Data reformatting is done by the 64-bit library routine that is the call target of 64-bit system calls. It receives 64-bit data from the caller, does any necessary reformatting to 32-bit data, and calls its corresponding kernel extension routine that passes the (now 32-bit) data on to the kernel's system call service routine. On completion of the system call, a return code is (generally) passed back. In addition, the kernel may have passed data back in user space. When the library routine regains control, it expands the return code from 32 bits to 64 and expands any returned data in user space.

For scalar parameters, the library does the following before calling the kernel extension routine:

- char, unsigned char - Passed without change.
- short, unsigned short - Passed without change.
- int, unsigned int, long, unsigned long - Tested to make sure that the value being passed will fit in the 32-bit version of int or unsigned int. A value that is too large generally results in setting errno to EINVAL and returning a return code of -1. Values that are not too large are passed as 32-bit integers.

  Note: Integers that are larger than 2^{32} are valid in some cases. For example, the lseek() routine takes an off_t (that is a long) as the seek position, a value

that can be larger than 2^{32} for large files. In this case, the system call is directed to the 32-bit llseek() interface that is prepared to handle long integers.

- float, double - Passed without change.
- pointer - Converted from 64-bit effective address to 32-bit effective address as described in Section 3.2.2.2, “64-Bit to 32-Bit Address Remapping” on page 40.

Many system calls involve passing the address of one or more structures in storage. If the structures involve any data types whose sizes differ between 32-bit and 64-bit mode (int, long, pointer), the library routine must pass a (32-bit) pointer to a local 32-bit copy of the data constructed on its own stack.

Some system calls are 64-bit-enabled, meaning they understand 64-bit pointers or longs. In such cases, the library code typically passes these by value in adjacent registers. The kernel code understands to parse the input as such. The shmat() call is an example of a service that understands a 64-bit address in adjacent registers.

If the kernel is going to look at the data being passed, the library routine allocates 32-bit versions of the same structure(s) and copies the data, field by field, through assignment statements. This results in automatic truncation of the int and long fields, some of which may need testing for magnitude before the conversion is done. Pointers are converted as described in Section 3.2.2.2, “64-Bit to 32-Bit Address Remapping” on page 40.

If the kernel fills in data as the result of a system call, the data must be widened by the library routine on return from the kernel extension. Space for the 32-bit version of the structures to be filled-in must be allocated by the library routine before calling the kernel extension. Assignment, field by field, will do the proper widening (zero extension or sign extension, as appropriate).

Returned pointers, such as from sbrk() or shmat(), require special handling between the library routine and kernel extension routine to ensure that the proper 64-bit values are returned to the library routine.

Note that some system calls involve data passing in both directions, into and out of the kernel, and thus require action on the part of the library routine before, and after, the system call.

3.2.2.2 64-Bit to 32-Bit Address Remapping

As shown in Figure 7, the PowerPC architecture divides the Effective Address (EA) into three fields:

- The Effective Segment ID (ESID)
- A 16-bit page number within the segment
- A 12-bit byte offset within the page

The width of the ESID varies with execution mode:

- In 32-bit mode, the ESID is 4 bits and is often referred to as the Segment Register number.
- In 64-bit mode, the ESID is 36 bits.

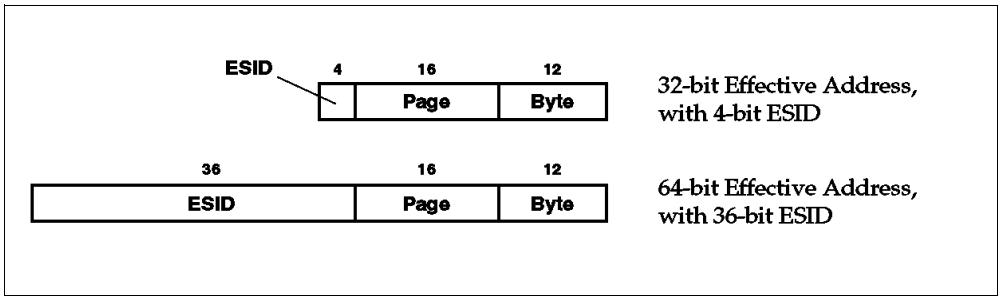


Figure 7. Effective Segment IDs in 32-Bit and 64-Bit Mode

For each process, the kernel maintains an array of sixteen entries, each holding a segment register value that defines the mapping of user effective addresses to system virtual addresses at the point of a system call. When a 32-bit application issues a system call and passes pointers to data in user space, the entries in this array are loaded into segment registers by the copyin() and copyout() routines to access user-space data. The kernel never accesses user-space data directly, but only through copyin() and copyout().

The 32-bit AIX kernel does not generally understand 64-bit address spaces. To perform kernel services for 64-bit address spaces, the 16-element mapping array previously mentioned must be set up so that the relevant user-space data for a system call can all be accessed through 32-bit addresses.

The VMM provides services to remap 64-bit pointers passed through system calls into a 32-bit address space. The 64-bit library code fills out a data structure describing the 64-bit pointers to be remapped using one of the __remap*() services. This structure is passed to one of the remap*_64() kernel services by the 64-bit kernel extension to remap the specified segments in the user address space into a 32-bit address space for use by the normal kernel system call routines.

The general form of the remapping structure is an array of effective segment identifiers and an integer indicating the number of such ESIDs to be remapped. A pointer to this remapping structure is passed to the remap*_64() routine. An ESID is 36 bits long and is represented as two integers; the high-order 32-bits of the ESID in the first word, and the remaining four bits of the ESID in the high-order four bits of the second word. To optimize the remapping of a single ESID, the two words describing the ESID are passed in registers rather than passing a pointer to a remapping structure. Additional bits are defined in the unused bits of the second word to indicate which form is being used – pointer or in-line registers.

Remapped addresses are only valid in the kernel for the duration of the system call. Subsequent system calls may not rely on 32-bit remapped addresses from previous system calls. Extensions with such requirements should save the segment register value and use the long-form virtual address instead or call as_unremap64() to obtain the 64-bit unremapped address to store.

3.2.2.3 Remap Library Data Structures

All the remap data structures are defined in the header file <sys/remap.h>, except the kernel MST-extension remap region that was discussed earlier. This header file is shipped with AIX because all of these structures may be required by the user.

The addr struct is a library-side structure that holds the information relevant to the remapping of the address of a single 64-bit area. It also holds the size of the structure pointed to. Segment crossings are allowed for items in the addr struct, with the limitation that a single item to remap may not be contained in more than fifteen segments. This limits the maximum size of an object to be remapped to 3.75 GB, assuming that it is aligned to a segment boundary.

The uremap struct is a library-side structure that consists of an array of addr structs and an integer giving the size of the array. In the general case, the number of entries can be quite large, such as the number of addresses of environment variables passed on execve(). The size is set at 17 in the structure definition to cover cases that need to pass this many addresses (writev() is one such case). Cases with more than 17 addresses require individually-constructed structs similar to uremap or just the C-allowed use of an index greater than an array-size, with appropriate storage malloc()'d by the caller. The naddr field of the uremap struct can accommodate large numbers of addr structs input. The point is that __remap() is not limited to 17 addresses to remap on one call. It can accept a large quantity. As of this writing, the upper bound has not been set. The number 17 was specifically arrived at from the iovec struct. 17 is iovcnt+1 to allow for the vectors plus the address of the structure itself.

The remap struct is the basic data structure used to communicate remapping information from the library side to the kernel extension. A single remap struct contains an unremapped 64-bit address (actually ESID), and additionally, the low bits of a remap struct (address) also have meaning. This will be discussed later.

The kremap struct is an array of remap structs plus an integer giving the number of array elements actually used. In the general case, this structure is passed by the library side to the kernel extension, which accesses it through copyin64(). The elements of the r[.] array represent up to fifteen unremapped 64-bit addresses.

The remapping code gets the right answer implicitly by making sure that the library remap code and the kernel remap code allocate srnos in the same order while processing the kremap struct. It is particularly important, for segment crossings, that adjacent segments become allocated in the kernel as needed. The r_xingcnt field in the remap struct is used to indicate the number of segments that a particular mapping crosses, so the segments may be created adjacently.

It is important not to confuse the uremap and kremap structures. The uremap structures are where the full 64-bit to 32-bit address translations are kept for each 64-bit address remapped. The kremap structs are passed to the kernel and indicate on a segment-basis which 64-bit ESIDs are remapped to which 32-bit sreg numbers. The mapping of entries in a uremap struct to entries in a kremap struct is typically many-to-one. For example, two pointers in the same ESID will create two uremap entries and one kremap entry.

3.2.2.4 Remap Library Programming Interfaces

The remap library services set up the remap data structures to be passed to the kernel remap services. There are three library remapping interfaces:

```
union __remap_handle __remap(struct uremap *up, struct kremap *kp)
struct remap __remap1(struct addr *ua)
void __remap2(struct addr *ua, struct kremap *kp)
```

Three interfaces exist, rather than just one, to improve performance on remapping. For the case of syscalls that require only one or two parameters to be remapped, the parameters may be passed in registers. This avoids the copyin() to the kernel by remap_64() of a kremap struct and is a significant performance saving. So, __remap1() or __remap2() should be used when only one or two parameters require remapping, respectively. The __remap() call should be used for syscalls requiring three or more remappings.

Like other things in 64-bit libc.a, __remap() is only available to applications running in 64-bit mode and linked with the correct library. The input to __remap() are the 64-bit addresses to remap in the addr structs (in uremap). The __remap() code fills in the addr structs with remapped 32-bit addresses. It also fills in the kremap struct with the 64-bit ESIDs in the slot corresponding to the 32-bit sreg number that __remap() picked for the remapping. Slot 0 of the kremap struct corresponds to 32-bit sreg 1. This is because sreg 0 is never given out for reasons previously stated having to do with NULL pointers. As such, only fifteen different segments can be remapped with one kremap struct.

There is no formal provision at the moment for remapping a given user-mode address to a specific remapped address specified by the library. It is expected that applications can avoid dependencies such as this. In the event that this is not possible, two alternatives exist; add a new force-remap service, or create a kremap structure with the specified remapping as though it had already been processed by __remap().

3.2.2.5 Remap Optimization for Multiple Addresses

If more than one item must be remapped, there is a folding optimization done by __remap(). For the second, and subsequent addresses in the uremap struct, __remap() determines whether the ESID of the address being operated on has already been remapped. If so, there is no need to make another kremap entry for it.

The preceding works fine for circumstances where segment crossings are not involved. It does, however, become complicated when you put segment-crossings into the equation. Assume that __remap() is to process the uremap struct in array-order. The addresses will not be sorted. Also assume that the first parameter, which does not cross a segment boundary and resides in 64-bit ESID 3, becomes remapped to 32-bit srno 1. If the second parameter does cross a segment boundary and starts at 64-bit ESID 2, in the absence of any special optimizations, this second parameter would have to be remapped to 32-bit srnos 2 and 3 (0 is unavailable for use). Note that the second parameter must occupy adjacent 32-bit segments.

There are several ways around this problem. One way is to have __remap() presort the input addr structs by ascending address, but this could get expensive, since it could have a virtually unbounded number of input address structures to process. The actual __remap() implementation is a two-pass method. The first pass (for a majority of the cases) takes a simple approach. In the preceding example, it would simply associate the 32-bit srnos 1, 2, and 3 as explained. As long as the rest of the remappings fit in the maximum fifteen slots that are available, this is not a problem, and there is no expensive sorting or post-processing required. The simple first pass will do folding only as far as checking to see if the 64-bit ESID has been remapped.

It is also possible to have a situation similar to the preceding example, except that the second remapping could touch fifteen segments, starting with ESID 2. This case would fail the first pass, and a second pass becomes necessary to sort them into ascending order. Although this second pass, when required, does add some overhead, it guarantees that the ranges to be mapped will fit into fifteen segments.

3.2.2.6 Remap Kernel Programming Interfaces

There are remapping programming interfaces that are exported to kernel extensions from the kernel. These are primarily for use by the 64-bit kernel extension; however, applications that add their own system calls will need these as well. For this reason, these services take up regular namespace and are not prefixed with an underscore. See the man pages for formal documentation of these routines as_unremap64() is intended for kernel extensions that really need to have the unremapped address.

The library-side maps 64-bit addresses to 32-bit in that it selects the sreg numbers to correspond to the ESIDs of the 64-bit addresses. However, the kernel-extension still has to update the address space map for the 64-bit thread to reflect these values, so that when the kernel uses these 32-bit addresses to access memory, the proper SID will be inferred. These services update the MST-extension remap struct described previously:

```
int      remap_64(struct remap r)
int      remap1_64(struct remap r)
int      remap2_64(struct remap r1, struct remap r2)
unsigned long long as_unremap64(uint addr32)
```

The reason for three different remap routines is to optimize for cases where only one or two ESIDs are used. This is a very common case. The remap_64() call handles the case of greater than two remap structs input as well as all other cases. Parameters to remap1_64() and remap2_64() are passed entirely in registers, so these routines do not have to copyin() a remap struct from user-mode.

There is also an internal kernel routine, remap(), that is used by copyin64() and others. The purpose of remap() is to support 64-bit-enabled code in the kernel that handles non-remapped addresses. Code calling remap() passes the address of a remap-like struct on the stack (typically), and this is used as a save area for the regular MST remap fields that are overwritten by this remap() call. At the end of the copyin64(), or any other call, restoremap() is called with the address of the stack-resident remap-like struct. The original contents of the MST remap fields that were temporarily overwritten are written back to the MST. An additional internal routine, as_remapaddr(), is used to return the original 64-bit unremapped address (modulo SEGSIZE) for a given 32-bit remapped address.

```
static void remap(ptr64 addr, uint nbytes, struct remaps * ptr)
static void restoremap(struct remaps * ptr)
ptr64       as_remapaddr(uint addr32)
```

These services take only one address and length to remap, and there is no place, currently, where calls to remap() are nested without restoremap(). Only one data structure at a time can be remapped in this fashion. The on-stack remaps structure is able to store the entire mstext->remaps array when there are fifteen segments total in the range to be remapped through savemap().

3.2.2.7 Optimizations for One or Two Parameters

The library-side and the kernel-side remap routines have optimizations built into them to pass one or two parameters by value if possible. This provides a significant improvement over requiring a full copyin64() of the kremap structure every time. There are three cases that the remap() code has to handle:

- The output library remapping resulted in only one remapping or remap struct (R_IN_LINE - see the following).
- The output remapping resulted in two remappings, and __remap2() was called (with two addr structs only).
- The output remapping resulted in more than one remapping if __remap() was called.

The low order bit of the cookie passed to the kernel remap services, all the way from the library, identifies what the cookie actually is. For example, although remap_64() is documented to have a struct remap() passed to it, if the R_IN_LINE bit is not set in this, then this struct remap is a pointer to a struct kremap, on which a copyin64() must be performed. If, however, the R_IN_LINE bit is set on input, then that indicates that all of the parameters collapsed into a single remapping in the library (case 1), and remap_64() can call remap1_64() to do the work. This avoids the copyin() when many parameters reside in the same segment (ESID). This should be a common case.

The R_IN_LINE flag is interpreted differently by the remap2_64() kernel service. The __remap2() subroutine sets the R_IN_LINE flag in the first remap struct if the output remappings collapsed into one. This indicates that the second parameter to remap2_64() should be ignored. See the next section for coding and calling conventions for __remap2().

3.2.2.8 Using the Remapping Services

There are very rigid rules on how to invoke the remapping services and how to handshake with the kernel-side remapping wrappers. Typically, whatever library remapping service was invoked for a particular wrapper, it should have the corresponding kernel remapping service called as well. For example, if __remap2() is called in the library wrapper, the kernel-extension should call remap2_64().

The following is very important:

- For a given system call, it is only permissible to make a single call to the remapping kernel services. In other words, it is not legal to call remap_64() twice on the same system call. Similarly, it is not legal to call remap1_64() and then remap2_64() on the same system call.

To detect potential misuse of the remapping services, the remap data structures are coded with a unique code for whichever library remap service created them. If a kernel service other than the correct one for a given library service is called, an error will result. For example, callers of __remap1() must call remap1_64().

Typically, a library wrapper will call a remap service and check if the return code is -1. If so, it will fail. After that, the returned value remap structure from the particular __remap* service is passed as the first parameter. If this is a __remap2() call, the first parameter is kremap.r[0], and the second parameter is kremap.r[1]. Of course, each of these parameters is split into two 32-bit registers.

A single remap struct fits into a single 64-bit register in user-mode but requires two registers each to pass to the kernel.

3.2.3 64-Bit XCOFF Format

The extended common object file format (XCOFF) combines the standard common object file format (COFF) with the TOC module format concept. This allows dynamic linking and replacement of units within an object file.

Until AIX 4.2, the XCOFF format assumed that addresses were 32 bits, memory could be no larger than 2^{32} bytes, and therefore, no object (csect, text section, and so on) could be larger than 2^{32} bytes. The XCOFF header files contained 32-bit fields for *value* in the symbol table, 32-bit length-fields for sections, and so on.

For PowerPC's 64-bit execution mode, these sizes are too restrictive. The reason for moving an application from 32-bit mode to 64-bit mode is to use addresses larger than 32 bits. In general, this means that all fields in XCOFF structures that can hold an address or a size should be increased to 64 bits. At the very least, it should be possible to describe a *bss* (common, or uninitialized data) object with a size greater than 2^{32} bytes.

3.2.3.1 XCOFF Design

There are two XCOFF formats; one for 32-bit applications, and one for 64-bit applications. The 64-bit XCOFF differs from the 32-bit XCOFF format in several ways, as listed in the following:

- The file size can be up to 2^{63} bytes (rather than 2^{31} bytes).
- Each XCOFF section can be up to 2^{63} bytes (rather than 2^{31} bytes).
- The virtual addresses in the process can be up to 2^{64} bytes (rather than 2^{32}).
- The offsets of objects within the XCOFF file can be up to 2^{63} bytes (rather than 2^{31}).
- Line numbers can be up to 2^{31} (rather than 2^{15}).

The 64-bit XCOFF does not differ from the 32-bit XCOFF format in the following fields because they are considered to be big enough:

- Symbol table indexes will still be limited to 2^{31}, allowing about half this many symbols in an executable (each symbol uses an average of about two symbol table entries).
- The string table will be limited to 2^{31} bytes in size, limiting the sum of the length of all symbol names to less than this value.

The following design issues have been implemented:

- All header file declarations for both 32-bit XCOFF and 64-bit XCOFF are contained in the same files used for 32-bit XCOFF declarations.
- Field names within structures are the same for structures that have different versions for 32-bit XCOFF and 64-bit XCOFF.
- Where possible, the structure sizes and layouts from the 32-bit XCOFF definition were not changed. Some fields have been rearranged to avoid alignment padding and to increase the number of fields that are at the same offset in both XCOFF versions.

- Fixed width types are used in all header files. The following are the *fixed width* types: char, short, int, and long long. (Of course, these types are only fixed with respect to AIX.)

  Note: Pointers exist in some of the existing header files. Since pointers are not fixed-width types, source code using these pointers will not compile in 64-bit mode.

- Source code compatibility is maintained for 32-bit programs written to process 32-bit XCOFF files.
- Minimal changes are required to port a 32-bit program that manipulates 32-bit XCOFF files to a 32-bit program that manipulates 64-bit XCOFF files.
- Minimal changes are required to port a 32-bit program that manipulates 32-bit XCOFF files to a 64-bit program that manipulates 32-bit XCOFF files.

3.2.3.2 Using the XCOFF Formats

There are different options for an application to use the XCOFF formats. The following strategies are possible:

- Using 32-bit XCOFF declarations.

  To only use the 32-bit XCOFF definitions, an application must include the appropriate header files. This will define only the structures for 32-bit XCOFF files. The 64-bit XCOFF structures and field names will not be defined. Structure names and field names will match those in previous versions of AIX, providing source compatibility.

  Note: Existing uses of shorthand type notation (for example, uint, ulong) have been removed.

- Using 64-bit XCOFF declarations.

  To only use the 64-bit XCOFF definitions, an application must define the preprocessor macro __XCOFF64__ . This will define only the structures for 64-bit XCOFF files. The 32-bit XCOFF structures and field names will not be defined. Structure names and field names will match the 32-bit XCOFF versions.

- Using both XCOFF declarations.

  To use separate 32-bit XCOFF and 64-bit XCOFF definitions, an application must define both the preprocessor macros __XCOFF32__ and __XCOFF64__. This will define structures for both kinds of XCOFF files. Structure and typedef names for 64-bit XCOFF will have the suffix _64 added to them, even if a common structure could be used.

- Using a hybrid of both XCOFF declarations.

  To use a hybrid of both the 32-bit XCOFF and 64-bit XCOFF definitions, an application must define the preprocessor macro __XCOFF_HYBRID__. This will define single structures that can be used with both 32-bit XCOFF and 64-bit XCOFF, where possible. Where fields in structures are a different size or at a different offset, suffixes 32 and 64 are used to differentiate between the fields. For example, the symbol table definition (in /usr/include/syms.h) will have the names n_offset32 and n_offset64, which should be used for 32-bit XCOFF and 64-bit XCOFF files respectively.

Depending on the execution environment of the executable and the targeted XCOFF format, the following combinations exist:

- 32-bit program manipulating 32-bit XCOFF files.

  A 32-bit program that manipulates 32-bit XCOFF files will require no change to continue to do so with the new header files.

  Note: Since the types of some fields are being changed from long to int, code that takes the address of such a field will result in a compiler warning when compiled in ANSI mode.

- 32-bit program manipulating 64-bit XCOFF files.

  An existing 32-bit program that manipulates 32-bit XCOFF files can be recompiled to manipulate 64-bit XCOFF files by defining the symbol __XCOFF64__. Some code changes will be necessary, but the changes with respect to the file format will be limited to cases where 32-bit XCOFF and 64-bit XCOFF use different constructs. In particular, n_name will not be defined in struct syment, and use of struct auxent will require changes since auxiliary symbols are redefined.

- 64-bit program manipulating 32-bit XCOFF files.

  An existing 32-bit program that processes 32-bit XCOFF files can be recompiled to a 64-bit program without change (with respect to the XCOFF definition) with two exceptions:

  - Pointers in the existing XCOFF files will be defined as ints in 64-bit mode.
  - Existing header files use preprocessor macro definitions in some cases. These same macros may no longer exist when compiling in 64-bit mode, so incidental use of the macros may require a code change.

3.2.3.3 Incomplete aouthdr Structure

Non-executable XCOFF files do not require a full-size auxiliary header. Current practice defines a short 32-bit auxiliary header that is generated by the compiler or the linker when the output file is not an executable. A short 64-bit auxiliary header will not be required by this definition. Applications examining non-executables must examine f_opthdr in the XCOFF header to determine how much of the auxiliary header is in the file.

There will be no auxiliary header used for non-executable 64-bit XCOFF files. Applications needing the fields from the auxiliary header for non-executable 64-bit XCOFF files should use the information in the section headers to generate these values. The fields where this may be necessary are text, data, and BSS sizes.

3.2.3.4 XCOFF Magic Number

The calling conventions for 32-bit mode and 64-bit mode are different in detail because one saves 32-bit General Purpose Registers (GPRs) onto the stack frame, and the other saves 64-bit GPRs. Calling from a program of one mode to a subroutine of the other mode is not supported. The linkage editor ld refuses a request to link programs of differing execution mode.

Because of this, a new magic number has been introduced for 64-bit execution mode. The primary purpose of the XCOFF magic number is to identify the associated Application Binary Interface (ABI), which implies a hardware system

and an execution environment. The 64-bit XCOFF magic number implies a 64-bit PowerPC processor and 64-bit execution mode on that processor.

The magic number keeps the linkage editor from binding 64-bit programs with 32-bit programs and keeps the loader from trying to execute 64-bit programs on 32-bit hardware.

The magic number is defined in the header file /usr/include/filehdr.h and has the name U803XTOCMAGIC with the value 0757.

3.2.4 Device Drivers

AIX 4.3 supports 64-bit applications on 64-bit PowerPC hosts in addition to maintaining support for 32-bit applications on all other supported hosts. Thus, on 64-bit hosts, both 32-bit and 64-bit applications can run simultaneously. To minimize the impact of adding 64-bit support, the kernel continues to run in 32-bit mode but provides interfaces to 64-bit applications by remapping the 64-bit application space address into a 32-bit address for the kernel. Thus, the following is true for device drivers in general and I/O drivers specifically:

- 32-bit versions of device drivers will operate correctly without change on AIX Version 4.3 in support of 32-bit applications.
- 64-bit applications require modification of only the entry points (such as ioctl()s) for proper operation.

The 4/8/8 model requires two primary changes for an I/O device driver:

- Providing ioctl support for 64-bit applications.
- Ensuring that structures describing fixed sized entities are size-invariant between both 32-bit and 64-bit applications.

3.2.4.1 Changes to ioctl()

The third argument of an ioctl call is referred to as the *arg* parameter. For some ioctls, the arg parameter can be a pointer. For AIX 4.3, the kernel guarantees that the arg parameter received by a device driver is always a 32-bit value. For 64-bit applications, the kernel will remap the address to a 32-bit address. Often, the arg parameter is a pointer to a data structure that may contain additional pointers. The kernel has no knowledge of this and, as a result, it is the device driver's responsibility to interpret these correctly. Device drivers that support 64-bit embedded pointers need to notify the kernel of this by setting the DEV_64BIT define for the d_opts flag passed to the devswadd() call from the config entry point of the device driver. For example a 64-bit-enabled SMP driver would use the following code segment:

```
devsw_struct.d_opts = DEV_MPSAFE | DEV_64BIT;
devswadd(devno,&devsw_struct);
```

For device drivers that do not set the DEV_64BIT flag, all ioctls from 64-bit applications will fail with an errno of EINVAL.

Since data structures with embedded pointers cannot remain size-invariant between 32-bit and 64-bit applications, a 64-bit-enabled device driver will need to maintain an internal-use-only 64-bit equivalent (recall the device driver will be compiled for 32-bit mode) of all such structures that can be passed as arg parameters. This can be accomplished by cloning the structure definition and replacing all pointers with type ptr64 (defined in types.h as unsigned long long).

3.2.4.2 Parameter Passing Examples

For example, assume a device driver's shipped header file has struct A, and a device driver supports an ioctl call whose arg parameter is the address of struct A:

```
struct A {
     int        aaa;
     char       *bbb;
     char       c;
     int        *ddd;
}
```

For a 32-bit application using struct A as the arg parameter of an ioctl, the device driver can recast the arg parameter as struct A. However, if the device driver determines the caller is a 64-bit application (through a call to the IS64U kernel macro), then the device driver will have to recast struct A to a new struct A64 defined as:

```
struct A64 {
     int         aaa;
     ptr64       bbb;
     char        c;
     ptr64       ddd;
}
```

The code segment of the device driver for this ioctl is similar to this:

```
          .
          .
          .
     if (IS64U) {
          /* The caller is a 64-bit application */
          x = (struct A64 *) arg;
          .
          .
          .
     } else {
          /* The caller is a 32-bit application */
          x = (struct A *) arg;
          .
          .
          .
     }
```

The following naming conventions are used to create the device driver's 64-bit equivalent structure:

- The 64-bit equivalent structure used by the device driver is included in the same shipped header file.
- It has a comment indicating that this is used for device drivers only and not applications.
- The name of the 64-bit equivalent structure is that of the original structure but with a 64 appended (for example, for sc_iocmd, it will be sc_iocmd64).

For the device driver to manipulate the 64-bit addresses, new 64-bit kernel services are provided. These kernel services support 32-bit unremapped addresses as well as 32-bit remapped addresses and 64-bit addresses. Thus all

64-bit-enabled drivers make the global kernel service replacements provided in Table 13.

Table 13. Old and New Kernel Services Used by Device Drivers

Old Kernel Service	New Kernel Service
copyin()	copyin64()
copyout()	copyout64()
xmattach()	xmattach64()
pinu()	xmempin()
unpinu()	xmemunpin()

The xmempin()/xmemunpin() calls use the same arguments as pinu()/unpinu(), with the exception that the third argument for xmempin()/xmemunpin() is the address of the cross memory descriptor from xmattach()/xmattach64() instead of a segflag. The xmattach64() call uses the same arguments as xmattach(), with the exception that the first argument for xmattach() is of type unsigned long long (ptr64) instead of type char *.

Note: xmattach64() enables xmempin(), so that the 64-bit address can be recast as a 32-bit address when passed to xmempin(). The xmdetach() call is used to undo xmattach64().

3.2.5 Loader

For AIX 4.3, the same basic loader system calls are provided to 64-bit programs. That is, there are 64-bit versions of execve(), fork(), exit(), load(), and unload() that are aware of the 64-bit user address space. There are also 64-bit versions of knlist() and sysconfig(), although these just interface to the existing 32-bit services. There is no 64-bit version of ptrace(), but 64-bit processes can be debugged by 32-bit debuggers. Finally, the loadquery(), loadbind(), and __loadx() functions are no longer system calls for 64-bit programs but are implemented in libc.a.

Four new external functions are added to the loader in AIX 4.3 to support 64-bit processing:

- ldr_init64()

  This function is called during kernel initialization when a 64-bit system boots.

- ldr_config64()

  This function is called by sysconfig() when a 64-bit machine is configured to run 64-bit processes.

- ldr_gettextusage64()

  This function computes the number of real memory pages used by the main executable of a process. It only needs to be called if the main executable is loaded in multiple working segments.

- ldr64()

  This is a new system call exported to the special 64-bit process that relocates shared objects. No other process has access to this system call.

On AIX 4.3, the same algorithm is used for loading 64-bit modules, but the work is split between kernel code and user-space code. The kernel part of the 64-bit loader is responsible for mapping the modules of a process into the 64-bit user address space. The user-space part processes the symbol tables and performs the relocation of the data sections.

Kernel extensions are still in XCOFF32 modules and they are entirely loaded and resolved by the kernel. The user-space processing of the shared library segments is handled by a privileged process running in 64-bit mode called the shared library loader assistant process or SHLAP.

The user-space processing of privately-loaded modules is handled by code that is loaded into a system-wide segment that is shared by all 64-bit processes. This code is called user-space loader assistant (USLA) and runs in the process of loading the module. The USLA is implemented as a loadable module that is prelinked at a fixed address, so that it will not have to be relocated. When an execve() system call leaves the kernel, it transfers control to the USLA that performs symbol resolution and relocation for privately-loaded modules. After load() calls, library code will be responsible for calling the USLA to complete the relocation of any newly-loaded modules.

Because the kernel is not performing symbol resolution and relocation for 64-bit processes, only a small portion of a 64-bit module needs to be addressable in the kernel. The kernel only needs to examine the XCOFF header, the section headers, the loader section headers, and the loader section import IDs. Even for extremely large programs, the size of these areas will be small. Only the import ID section is variable length, and its length depends on the number of dependents a module has, not on the size of the module itself. These portions of a module can be read into allocated memory, avoiding addressability problems inherent in the existing 32-bit loader.

3.2.6 Virtual Memory Manager

The AIX 4.3 design point is a 60-bit user address space. The areas impacted in the VMM are the address space code, the shared memory code, teaching VMM code to understand remapped addresses, and the remapping services themselves.

3.2.6.1 Executing a 64-Bit Program

The size of a user-address space only changes as a result of exec(). A 32-bit program may exec a 32 or 64-bit program, and conversely, all combinations are possible.

The VMM provides support routines for exec() processing to switch between a 32-bit and 64-bit executable. The routine vm_makeme64() is called when the exec()'d program is a 64-bit program. This routine pins and initializes the 64-bit u-block extension, initializes the 64-bit address space structures, initializes the Address Space Register (ASR) in the Machine State (MST) extension with the real address of the segment table, sets the sbreak and stack sizes, and marks the process as a 64-bit executable.

The routine vm_makeme32() is called whenever a 64-bit program execs(). It is called even if the program to be executed is 64-bits. This routine initializes a 32-bit user address space from the 64-bit one, clears the 64-bit MST extension

address and marks the process as a 32-bit executable. The segstate structure is handled later by shm_asinit(), and the 64-bit u-block extension is freed in the subsequent call to vm_cleardata(). The vm_cleardata() call also initializes the sbreak value for the private segment and adjusts the storage protect key accordingly in the external page tables covering the user region. There is no service to re-initialize a 64-bit adspace to a newly-created 64-bit adspace, so it is necessary to call vm_makeme32, followed by vm_cleardata() and vm_makeme64(), when a 64-bit program exec()'s another 64-bit program.

3.2.6.2 Address Space Management

The address space management code is significantly impacted for 64-bits. The code was updated to understand segment numbers, or effective segment IDs, above the first sixteen IDs.

32-Bit Address Space Programming Interfaces

The following 32-bit services that operate on the process address space are exported to kernel extensions, so the AIX 4.3 versions of these services are binary-compatible with prior versions. For internal base kernel use, these address space services are extended to handle 64-bit address spaces but only by code that has been modified to be 64-bit aware. This means only by code that knows how to compute an appropriate adspace_t for a 64-bit address space:

```
caddr_t    as_att(adspace_t * adsp, vmhandle_t srval, caddr_t addr)
int        as_det(adspace_t * adsp, caddr_t addr)
vmhandle_t as_geth(adspace_t * adsp, caddr_t addr)
vmhandle_t as_getsrval(adspace_t * adsp, caddr_t addr)
void       as_puth(adspace_t *adsp, vmhandle_t srval)
void       as_seth(adspace_t * adsp, vmhandle_t srval, caddr_t addr)
adspace_t  *getadsp()
```

To provide compatibility for 32-bit kernel extensions, the 32-bit getadsp() kernel service is modified to determine if it is running under a 64-bit user address space, and if so, it will return the first adspace_t. This represents ESIDS 0-15. This could enable some extensions to run under the 4 GB boundary for 64-bit.

All of the 32-bit services listed may be used by a kernel extension or device driver, but they will only operate on addresses below 4 GB, even when under a 64-bit process. The service to compute an adspace_t for 64-bit, getadsp64(), is not exported from the kernel. Thus, these routines are not enabled outside the kernel to operate above the 4 GB line.

Kernel services and drivers should use the new 64-bit address space services described in the following.

For 64-bit address spaces (internal to the kernel, where there is getadsp64()), the address arguments specified preceding as caddr_ts are actually 32-bit quantities that are treated as offsets into the appropriate adspace_t. The only reason for keeping enablement of these services for 64-bit inside the kernel is that, on some system calls, there should be some performance improvement by only computing an adspace_t once.

64-Bit Address Space Programming Interfaces

The following additional address space services are provided for use by the 64-bit kernel extension and by other base kernel code that has been modified to be 64-bit aware.

All of the following services are exported:

```
unsigned long long as_att64(vmhandle_t srval, int offset)
int                as_det64(unsigned long long addr64)
vmhandle_t         as_geth64(unsigned long long addr64)
vmhandle_t         as_getsrval64(unsigned long long addr64)
int                as_puth64(unsigned long long addr64, vmhandle_t srval)
int                as_seth64(unsigned long long addr64, vmhandle_t srval)
int                IS64U
```

The address space programming model for 64-bit introduces a copy of all the 32-bit interfaces appropriately scaled for 64-bit addresses. All of the 64-bit services work properly under a 32-bit user address space or under a kproc as well as a 64-bit user address space:

One additional non-exported service is provided:

```
adspace_t   *getadsp64(unsigned long addr64)
```

The getadsp64() service exists to provide a bridge between the 64-bit and 32-bit services. The adspace_t returned by getadsp64() may be passed to any of the 32-bit services, and it will work properly. The one exception to the example is that 32-bit as_att() will not support attaching anywhere other than the first adspace_t. The advantage of using getadsp64() is the performance improvement of only computing an adspace_t once per system call.

The concept of an adspace_t really does not exist with the new 64-bit interfaces; getadsp64() is the only exception, and it is only for use with the 32-bit interfaces internal to the kernel. The advantage of not having an adspace_t externalized is that the width of an adspace_t is no longer surfaced to extensions. This saves the extensions the overhead of having to compute another adspace_t every 4 GB.

Kernel extensions writing new code to enable 64-bit support that need the address space services, should use the new *64 services instead of the current 32-bit services. The *64 services handle all the general cases for 64-bit and 32-bit address spaces. The 32-bit services will only work for addresses less than 4 GB outside the kernel.

How to Determine if this is a 64-Bit Address Space

The IS64U macro will return true if the user address space for the current process is 64-bits. This macro is valid only in kernel mode. If used inside the kernel, this will return the value of U.U_64bit directly. If used outside the kernel in an extension, this will generate a subroutine call to the new kernel service: _as_is64(). _as_is64() will simply return the value of the variable. U.U_64bit is managed by exec() in vm_makeme64/32. IS64U is defined in user.h.

3.2.6.3 Shared Memory Management

The shared memory code is impacted for 64-bit support since it must attach shared memory segments at large addresses. The functions shmat() and mmap() will behave as follows regarding segment number (ESID) allocation:

- If no fixed address is specified, then allocation takes place from the shmat()/mmap() pool at ESIDs: 0x70000000 - 0x7FFFFFFF.
- If a fixed address is specified by the user, then the allocation will be allowed as long as the ESID is less than 0x80000000 and is not ESID 0-2,13, or 15.

- The shared memory allocator internally allocates anywhere in the address space. The reason for this is that other areas of the kernel, for example the loader, need to insert segments at ESIDs greater than 0x80000000. Therefore, shm_insert() is allowed to insert anywhere, but the higher-level user-interfaces perform the validation for the user level.

Shared Memory User and Exported Kernel Programming Interfaces

The shared memory component has numerous system calls surfaced to the user. All of these system calls are registered in the 64-bit libc.a and the 64-bit kernel extension. The following is the list of the 32-bit shared memory system calls:

```
void* shmat(int shmid, const void *address, int shmflag)
int   shmctl(int shmid, int command, struct shmid_ds *buffer)
int   shmdt(const void *address)
int   shmget(key_t key, size_t size, int shmflag)
int   disclaim(char *address, uint len, uint flag)
```

The following are the new, 64-bit-ready interfaces for the shared memory services. These interfaces are called directly from the 64-bit kernel extensions only:

```
ptr64   _shmat64(int shmid, ptr64 address, int shmflag)
int     _shmdt64(ptr64 address)
int     _disclaim64(ptr64 addr,unsigned len,unsigned flag)
```

The _shmat64(), _shmdt64(), and _disclaim64() calls do not require parameter remapping, since they are 64-bit-enabled. They do, however, require 64-bit libc.a to split their address parameters into two adjacent general purpose registers for processing in 32-bit mode. the shmctl() call does require parameter remapping on the pointer to the shmid_ds. Additionally, shmget() takes a size_t as input. This typedef is an unsigned long, which has different widths in 32-bit and 64-bit programs.

The prototype to shmget() will not change for 64 bits. The low 32 bits of the 64-bit size will be passed to the kernel, with the size being range-checked for 32 bits in the library-side. There will be no increase in size of the supported memory region. The key_t parameter to shmget() is currently a long. It will be changed to always be an int. This will be true for 32-bit and 64-bit code to make it invariant. This allows predictable message-passing between 32-bit and 64-bit processes.

3.2.6.4 User Data and Stack Management

The 32-bit programming interfaces for adjusting a program's data size are brk() and sbrk():

```
int     brk(void *enddatasegment)
void *  sbrk(int increment)
```

There is a change to the sbrk() interface required by 64-bit mode and UNIX98 standards. For UNIX98, sbrk() needs to take a long on input.

This poses a breakage for those who want to have the UNIX95-behavior of sbrk() that obeys the preceding prototype, taking an int in 64-bit mode. The problem is that the 64-bit library wrapper for sbrk() has no way of determining whether it was passed a 32-bit value or a 64-bit value. The compiler will not ensure that the high 32-bits of a register are 0 for int's. Since most programmers going to 64-bit with their applications will require some porting effort to do so, changing sbrk() to the

UNIX98 interface will add very little extra work. This does not mean that they have to change everything to UNIX98 conformance, just sbrk() in this case.

The sbrk() function prototype in <sys/unistd.h> was changed to pass a long for all 64-bit compilations; that is, if __64BIT__ is set. For 32-bit compilations, the sbrk() prototype will be conditionally compiled to generate the appropriate UNIX95 or UNIX98 prototype since the data-width between int and long does not change for 32-bit. Code that wants to run 64-bit must either make sure it passes a long, or if it obeys UNIX98, it is required to include <sys/unistd.h>. The standards require header file inclusion. The prototype for sbrk(), which defines it as taking a long, is as follows:

```
void *   sbrk(intptr_t increment)
```

`intptr_t` is a new type. Defined in <sys/inttypes.h>, it maps to a long.

3.3 Application Development

Section 3.2, “64-Bit Core Design” on page 33, explained the design issues of AIX 4.3 with respect to 64-bit application support. The changes in the core design of AIX have impacts on various components of the software development environment. This section describes what decisions have been made to provide a migration path from 32-bit to 64-bit applications. It shows the modifications that have been made to the most important tools in the software development area, such as the compiler, linker, and archiver.

3.3.1 C Compiler

This section discusses the implementation of 64-bit capabilities in the C for AIX compiler. The C compiler provides supporting functions that can enable the usability of 64-bit C syntax and semantics.

Each program compiled for execution on AIX is intended for execution in one particular *target execution mode*: 32-bit mode or 64-bit mode. The default compilation and assembly mode is 32-bit. This is the ILP32 model. The change from the default 32-bit to 64-bit mode is under user control. In the compiler, the option -q64 is used to change the compilation mode.

The default execution mode is not directly controllable by the user processes but can be examined indirectly (for code dynamically targeted to multiple environments) through the pointer or long type size. The compiler provides porting assistance options wherever there are statements that can be ambiguously interpreted for the LP64 environment.

When running 32-bit processes on 64-bit platforms, the execution is transparent and identical to executing on a 32-bit platform with no loss of performance. When trying to run 64-bit processes on 32-bit platforms, the execution will fail in an obvious manner.

The 64-bit implementation in the C front end does not change the default behavior of the compiler. The compiler only changes the behavior of code when compiled in 64-bit mode. Code that was compiled in 32-bit mode that has no requirements for large address spaces (pointers) or large object sizes (arrays and dynamic heaps) will not need to be recompiled to work in 32-bit mode on a 64-bit

platform. You may recompile the code in 64-bit mode to check performance implications in 64-bit mode on a 64-bit platform.

While most code will recompile and execute properly in 64-bit mode, some code will behave differently, or may not function at all, due to nonportability deliberately or accidentally written into the code. Common causes of behavior changes are due to mixed use of long and int types in operators, especially comparison operators that will change the code execution path. Although the usual operand promotion rules do not change, the changed size of long types may yield surprising and unexpected results. Function arguments and return types from functions need to be checked for their actual value. Many library functions return long types and take long types that are implicit, such as size_t and ptrdiff_t. Structures, structure alignments, member alignments, bit fields, and enums are guaranteed to change when compiled in 64-bit mode (especially if they contain long and pointer types).

3.3.1.1 Compiler Mode

The generation of 64-bit instructions and 64-bit XCOFF is called the 64-bit compilation mode. The compiler invocation for setting the 64-bit versus 32-bit mode evaluates several sources. They are:

- Internal default
- Environment variable OBJECT_MODE
- Configuration file
- Command line
- Source file

The compiler evaluates the options in the given order of the items. The last one takes precedence. For example, if the environment variable OBJECT_MODE exists, it will replace the internal default of the compiler. Table 14 provides a list of OBJECT_MODE settings and the compilation mode behavior.

Table 14. Settings for OBJECT_MODE and the Resulting Compiler Behavior

OBJECT_MODE setting	Compilation Mode Behavior
not set	32-bit mode
32	32-bit mode
64	64-bit mode
32_64	fatal error and stop (unless there is explicit user setting in the config file or command line) with message: `1501-054 OBJECT_MODE=32_64 is for mixed-mode and is not a valid setting for the compiler.`
anything else	fatal error and stop (unless there is explicit user setting in the config file or command line) with message: `1501-055 OBJECT_MODE setting is not recognized and is not a valid setting for the compiler.`

This option allows the code to function in a 32- or 64-bit environment without excessive use of new option names. This will maintain compatibility with other tools that can exhibit 32/64-bit mode behavior since they will all use the OBJECT_MODE environment variable. It also maintains compatibility with machines without 64-bit capability that want to compile in 64-bit mode. In all

cases, the user is free to override the environment variable with an explicit option in the config file or the command line.

32-bit mode is invoked by specifying -q32 on the compiler command line and is the default if OBJECT_MODE is not set. This option is equivalent to a direct expansion into the -qarch=com option. For the compilers that do not have 64-bit yet, use of the -q32 and/or the -q64 option will cause the following warning (this is the usual warning on unrecognized options):

```
1501-055 Option -q32, -q64 is not recognized and is ignored.
```

Problems with #pragma arch Suboptions in Source Files

The -q32/64 option has no pragma equivalence because the compilation mode must be determined before the compiler driver exits and invokes the compiler. Implicitly expanded options are parsed with the rest of the command line to produce a final compilation mode. From this compilation mode, the options are passed separately to the compiler, linker, and assembler. However, since the ARCH suboption has an equivalent #pragma arch suboption in the source file, the individual files may be compiled in a different mode than what was decided by the command line.

It was decided to disallow the setting of a #pragma arch suboption in a source file. This is a change in Version 4.0 of the C compiler that means a loss of backward compatibility with previous C compiler versions.

Mixed-Mode Compilation and Two-Step Compile and Linking

When you cause a mixed 32- and 64-bit compilation mode, your XCOFF objects will not bind. This will become obvious if the compile and link occurred in one step. However, you may not know this if the compile and link occurred in different steps. In other words, if you compiled and produced 64-bit objects, you need to remember to link using the 64-bit mode (when linking using xlc), otherwise the objects will not link. If the objects are in mixed XCOFF mode, then they will never link, and you must recompile completely, making sure that all objects will be in the same mode.

There is a set of new configuration file attributes that are used in place of the normal attributes whenever the compiler determines that the 64-bit mode is enabled. These new attributes are:

- crt_64
- gcrt_64
- mcrt_64

The new definitions for these attributes are:

crt_64 Path name of the object file passed as the first parameter to the linkage editor. If you do not specify either -p or the -pg option, the crt_64 value is used. The default depends on the compiler being used.

gcrt_64 Path name of the object file passed as the first parameter to the linkage editor. If you specify the -pg option, the gcrt value is used. The default depends on the compiler being used.

mcrt_64 Path name of the object file passed as the first parameter to the linkage editor if you have specified the -p option. The default depends on the compiler being used.

Note: The invocation of 64-bit mode using the -q64 option (either explicitly or using a stanza) automatically implies linkage in 64-bit mode. The compiler driver automatically and quietly generates the correct linker options (-b32 or -b64) to call the binder or the correct assembler option (-a32 or -a64) when calling the assembler. Therefore, these options do not need to be set by the user.

Predefined __64BIT__ Macro

When the compiler is invoked to compile for 64-bit mode, the preprocessor macro __64BIT__ is predefined. When it is invoked in 32-bit (default) mode, this macro is not defined. The variable can be tested through:

```
#if defined(__64BIT__)
```

or

```
#ifdef __64BIT__
```

to select lines of code (such as printf statements) that are appropriate for 64 or 32-bit mode. The ability to choose execution mode (of the final executable) at compile time and the existence of the __64BIT__ macro implies there is no need for an application to determine its execution mode at run time.

When the compiler is invoked to compile for 64-bit mode, this macro is set to a value of 1 internally, so that the C preprocessor and compiler will recognize it. It cannot be redefined or undefined. Any attempt at redefinition will fail.

3.3.1.2 Fixed-Width Types

There is a a set of types that maintain their width regardless of the compilation mode of the compiler. These types may be used if the program relies on an exact and unchanging size for the types.

Programs that exchange formatted messages are, for example:

- An X-windows server and client executing in different modes.
- Processes running in different modes that share data (using shmat() to jointly access and change common memory areas).
- Data files written by applications running in one mode and read by applications running in a different mode.

All of these demand the availability of fixed-width types.

ANSI introduced two sets of types. One is the signed fixed-size integral type:

- int8_t
- int16_t
- int32_t
- int64_t

The other is the unsigned fixed-size integral type:

- uint8_t
- uint16_t
- uint32_t
- uint64_t

These ANSI types are defined through the header <inttypes.h>. Note that the signed or unsigned are explicitly coded into the typedefs and not left to chance. Although it is unlikely that the defaults for short/int/long are unsigned, it is possible on some machines. Furthermore, by forcing the keyword, this would have the same error behavior in all cases if the user were to add a sign qualifier to the ANSI types, as in signed int8_t.

3.3.1.3 Structure Alignment and Bitfields

The LP64 specifications will change the size, member alignment, structure alignment, and bitfield sizes and alignment of most structures implicitly. Structures with only long and pointer types will at least double in size depending on the alignment mode.

Sharing data between 64-bit and 32-bit processes will not be possible unless the fixed-width types are used, or the structure is devoid of pointer and long types. Special attention needs to be paid to unions that attempt to overlay int types with long types or pointer types.

For the details of alignment in different modes and in combination with different compiler flags, consult the compiler reference manual.

Table 15 provides the different alignments found in 32- or 64-bit modes.

Table 15. Alignment of Basic Data Types in 32- and 64-Bit Mode

Type	32-Bit	64-Bit
char	1	1
short	2	2
int	4	4
long	4	8
long long	8	8
float	4	4
double	8	8
pointer	4	8

According to ANSI, a bit field will have a type that is a qualified or unqualified version of one of int, unsigned int, or signed int. Therefore, the ANSI mode cannot change the type.

Bitfields in ANSI mode can only be signed int or unsigned int. In extended mode, common mode, or k&r mode, non-integer bitfields are tolerated. When a non-integer bitfield is tolerated, it means that any type other than int will be converted to int.

The extended mode bitfields are updated to long types to admit 64-bit width in 64-bit mode. If a long type bitfield of length greater than 32-bits is used in 32-bit extended mode, the following message is given:

```
1506-003 (S) Width of a bit-field of type "long" cannot exceed 32.
```

If a long type bitfield of length greater than 64-bits is used in 64-bit extended mode, the following message is given:

```
1506-003 (S) Width of a bit-field of type "long" cannot exceed 64.
```

Bitfields are packed into the current word. Adjacent bitfields that cross a word boundary will start at a new storage unit. This storage unit is a word in power, full or natural alignment in 32-bit mode, but is a double word in 64-bit mode. In 64-bit mode, adjacent declarations of bitfields of type long can now be contained into one storage unit. Since long bitfields of greater than 32-bits were not permitted in 32-bit mode, this does not change and is not a portability problem.

Note that the packed alignment option just reduces the alignment ceiling to one, two, four, or eight bytes depending on the packed=1|2|4|8 setting and leaves the remaining alignment parameters unchanged.

3.3.1.4 Enum Support

Enum constants are always of type int, except when the range of these constants is beyond the range of int, in which case, they have type unsigned int. Enum variables may be smaller depending on the mode of -qenum=small|int|1|2|4|8 option.

small Specifies that enumeration occupies a minimum amount of storage (either 1, 2, 4, or 8 bytes) depending on the range of enum constants.

int Enumeration occupies 4 bytes and are represented by int.

1 Enumeration occupies 1 byte and are represented by char.

2 Enumeration occupies 2 bytes and are represented by short.

4 Enumeration occupies 4 bytes and are represented by int.

8 Enumeration occupies 8 bytes and are represented by long.

enum=int and enum=4 are not the same. The enum=4 allows signed and unsigned variant. The enum constants will usually be typed int, even in 64-bit mode, to enhance compatibility with 32-bit programs. Only when the range chooses an unsigned long or long, the constant will use unsigned long or long types respectively. In 32-bit mode, -qenum=8 will yield an warning message:

```
1506-749 (W) Enum=8 is not valid in 32-bit mode, setting enum=4 instead.
```

3.3.2 XL Fortran Version 5

XL Fortran Version 5.1 introduced a new compiler option, -q64. This allows the object code to run in 64-bit mode. The programming conventions are similar to C. For a better understanding of Fortran tuning on POWER3 processors, see *RS/6000 Scientific and Technical Computing: POWER3 Introduction and Tuning Guide*, SG24-5155.

3.3.3 System Libraries

AIX 4.3 provides a 32-bit Application Binary Interface (ABI) and a 64-bit ABI. The 32-bit ABI consists of the entire pre-AIX 4.3 ABI and provides binary compatibility at the same level as maintained by previous releases.

The dual ABI means two different version of all the Application Program Interfaces (APIs). The mechanism for this are two separate versions of all the objects in a given library. The objects are distinguished by distinct names. The linker is able to distinguish which object to use for a given symbol based on the differing object formats (32-bit and 64-bit).

The following libraries and APIs are not supported in the 64-bit environment:

- lib300.a - obsolete ASCII graphing library
- lib300s.a - obsolete ASCII graphing library
- lib4014.a - obsolete ASCII graphing library
- lib450.a - obsolete ASCII graphing library
- libIN.a - Interactive Systems library from RT days
- libPW.a - obsolete Programmer's Workbench library
- libcur.a - obsolete IBM-invented curses extensions
- libplot.a - obsolete ASCII graphing library
- libbsd.a - nonstandard BSD APIs (others are in libc.a)

The preceding APIs are also obsolete in 32-bit environments and will not be supported in the future. Also, the back level X11 compatibility and libcurses libraries do not have 64-bit versions.

The following functions will not be provided in the 64-bit version of libc.a:

- NC*
- NL*
- _NC*
- _NL*
- isj*
- jis*
- compile
- step
- advance

The asterisk represents a wild card.

3.3.4 Linker

Compilers, the assembler, and the binder create XCOFF64 object files when invoked in 64-bit mode. The AIX 4.3 linker links these object files in the same way that it links XCOFF32 object files.

The AIX linker supports the development of 64-bit applications, libraries, and kernel extensions. For 64-bit applications and libraries, the linker is able to read and write XCOFF64 files, performing internal processing appropriate for 64-bit mode. For 64-bit kernel extensions, the linker is able to mark exported symbols with storage-mapping class XMC_SV, XMC_SV64, or XMC_SV3264.

In this section, *mode* indicates the *linking mode*, which means whether an XCOFF32 or XCOFF64 file is generated as the output file. Mixed-mode linking is not allowed.

Archives may contain both XCOFF32 and XCOFF64 members. Depending on the mode, members in the appropriate format are processed, while other XCOFF members are silently ignored. Archives containing XCOFF64 members use a new

archive file format that provides for separate global symbol tables for XCOFF32 and XCOFF64 members (see Section 3.3.5, “Archiver” on page 64). This new archive format is also used for all archives created on AIX 4.3, so the binder reads the new archive format even when running in 32-bit mode.

The following new command line flags and options in import files are introduced:

- **-b32** option

  Specifies 32-bit linking mode. In this mode, all input object files must be XCOFF32 files, or an error is reported. Only XCOFF32 archive members are processed. Other archive members are ignored. For import files specifying the mode of certain symbols, 64-bit imports are ignored.

- **-b64** option

  Specifies 64-bit linking mode. In this mode, all input object files must be XCOFF64 files, or an error is reported. Only XCOFF64 archive members are processed. Other archive members are ignored. For import files specifying the mode of certain symbols, 32-bit imports are ignored.

- **32** option in an import file

  This option can be used in an import file to specify that subsequent symbols should be processed when linking in 32-bit mode but ignored when linking in 64-bit mode. If no 32 or 64 option is specified, all symbols are processed in both 32 and 64-bit modes.

  Note: The syntax for import file options is a pound sign (#) followed by a blank followed by a list of options.

- **64** option in an import file

  This option can be used in an import file to specify that subsequent symbols should be processed when linking in 64-bit mode but ignored when linking in 32-bit mode. If no 32 or 64 option is specified, all symbols are processed in both 32 and 64-bit modes.

- **no32** or **no64** option in an import file

  This option overrides a previous 32 or 64 option. Subsequent symbols are processed in both 32 and 64-bit modes.

- **OBJECT_MODE** environment variable

  If the -b32 or -b64 options are not used, the OBJECT_MODE environment variable is examined to determine the linking mode. If the value of OBJECT_MODE is 64, 64-bit mode is used. If the value is 32_64, the linker prints an error message and exits with a non-zero return code. Otherwise, 32-bit mode is used.

If both -b32 and -b64 options are specified, the last specified option is used. If neither option is specified, the mode is determined from the value of the environment variable OBJECT_MODE.

New keywords are recognized in import and export files. The keywords are svc64 and svc3264, with synonyms syscall64 and syscall3264. For ease of use, svc32 and syscall32 are added as well. They are equivalent to svc and syscall. All these keywords may be in upper- or lower-case. The keywords are ignored in import files. In export files, a symbol exported with the svc64 keyword is given storage-mapping class XMC_SVC64 in the loader-section symbol table. Similarly,

symbols exported with svc3264 are assigned a storage-mapping class, XMC_SVC3264. The existing flags and options -T, -D, -S, -bD, -bS, -bmaxdata, -bmaxstack, -bpD, and -bpT will accept 64-bit values as arguments. The 64-bit values are passed to the binder in the respective binder subcommands, regardless of the mode. The binder reports errors for used values that are too large for 32-bit mode. Depending on the options specified, some values are never used and do not result in an error.

3.3.5 Archiver

The AIX 4.3 the ar command handles the archiving of 64-bit XCOFF object modules in addition to the current 32-bit object modules. An archive file in AIX 4.2 supports only a single global symbol table to reference the symbols contained in all object-file modules within the archive. To support the two formats of object files, it is important that the symbols of 64-bit objects be distinguishable from those of 32-bit objects. This is not an issue for the old (pre-AIX 4.3) archive file format since 64-bit modules are not stored in these archives. For the AIX 4.3 archive format, however, there are two global symbol tables: one for 32-bit object symbols and one for 64-bit object symbols. The ar command is able to recognize each type of object file and store its symbols in the appropriate table.

The ar command maintains compatibility with the previous archive file format. If given an archive file of the old format, ar still adds, deletes, reorders, and lists members without altering the format of the archive file except in two cases: when the user explicitly requests conversion to the AIX 4.3 format by using the -o option, or when the user adds a 64-bit object to the archive. For the latter case, a 64-bit object cannot be handled by the old-format archive, so conversion is required. A mechanism is provided for ar to refuse the 64-bit object instead of converting the archive format.

When creating a new archive, the 4.3 ar command automatically uses the new format. For files that are not XCOFF objects of either type, ar processes them as usual. If such files are added to a nonexisting archive, the new format is used in creation. If ar is given an old-format archive, it is not reformatted (unless the user requests it). The new maximum size of an archive has increased from $(10^{11} - 1)$ to $(10^{19} - 1)$ bytes.

A new flag has been added, -X, which requires an argument of either 32, 64, or 32_64. This flag indicates to ar whether to accept only 32-bit objects or only 64-bit objects (in addition to any non-object files, which are always valid) or both. If both -X32 and -X64 are specified, ar treats it as if -X32_64 were specified and accept both object types. If only one of the options is specified, ar ignores all object files in the archive that are not of the specified type. If such objects are specified on the command line, an error message is issued, but other acceptable objects are still processed. If the -X option is given with an unrecognized argument, an error message is printed with the usage statement, and ar exits.

A new environment variable, OBJECT_MODE, is recognized by ar to determine the XCOFF file type(s) acceptable for processing. The values of OBJECT_MODE=32, OBEJECT_MODE=64, and OBJECT_MODE=32_64 all have the equivalent function of their -X flag counterparts. If both the environment variable and the -X flag are specified, the flag will take precedence over the environment variable. If no -X flag is given and OBJECT_MODE is unset or is set to an unrecognized value, 32-bit mode is used.

Displaying the symbol table with the -w option shows the symbol table depending on the chosen mode. In 32-bit mode, only the 32-bit symbol table is displayed; in 64-bit mode, only the 64-bit symbol table is displayed. In mixed mode, both are displayed. To distinguish the 32-bit table from the 64-bit table in mixed mode, each symbol table entry is followed by a field containing the characters 32 or 64, respectively. Each field is separated from the previous field by a single tab character. The 32-bit table is printed before the 64-bit table if both are present.

3.3.6 The dbx Debugger

The dbx command provides a symbolic debug program for C, C++, Pascal, and FORTRAN 32-bit and 64-bit programs. It is also able to process and examine core files generated from both 32-bit and 64-bit processes. dbx itself stays a 32-bit program, but its data is expanded to accommodate debugging of 64-bit programs. In case of debugging 32-bit code, there are *wrapper*, or interface routines, that translate from 32-bit formats to dbx's internal 64-bit data formats.

The dbx program:

- Automatically identifies the execution mode of the code
- Understands the new XCOFF64 format
- Understands the new archive format
- Understands the new coredump format
- Supports M:N threads debugging
- Accepts 64-bit addresses input
- Does arithmetical calculations in 64-bit precision
- Displays 64-bit values

The dbx parser has been changed to accept 64-bit addresses.

Note: To save typing of 16-digit long addresses, the user can set a dbx variable and use it as a base.

Commands will allow 64-bit values for addresses, subscripts, ranges, offsets, and so on. The 32-bit dbx required a suffix ll or ull on 64-bit number input. For example, 0x123456789ull. Any number that was too big for its type was set to the maximum value without any warning message. For ease of use, dbx now assumes long long for any input string greater than 8 digits for hex, 11 digits for octal, and 10 digits for decimal including leading zeroes if any. For convenience, underscores are allowed in any numeric input. For example, 0x1234_4567_890A_BCDE. They are ignored and not counted in the preceding sizes like in the assembler.

3.3.7 Commands and Utilities

All commands that needed to be modified for 64-bit support were modified to work with 32-bit and 64-bit objects (object files or processes). No commands and utilities were converted to 64-bit executables; they remain 32-bit executables.

There are two major reasons for changing commands:

- The new XCOFF64 format
- Data values that might exceed 2^{31}

In general, no command user interfaces or new flags were added to these commands, with the exception of the XCOFF-specific commands and the `lint` command. For most of the commands that deal with XCOFF files, a new flag was added, -X that requires an argument of either 32, 64, or 32_64. This flag indicates whether to recognize only 32-bit objects, 64-bit objects, or both. If both -X32 and -X64 are specified, the command treats it as -X32_64 and recognizes both object types. If only one of the options is specified, the command ignores all object files that are not of the specified type. If such objects are specified on the command line, an error message is issued, but other acceptable objects are still processed. If the -X option is given with an unrecognized argument, an error message is printed with the, usage statement and the command exits.

For the same XCOFF-specific commands, the environment variable OBJECT_MODE determines the XCOFF file type(s) to be recognized. The defined values of OBJECT_MODE=32, OBJECT_MODE=64, and OBJECT_MODE=32_64 all have the equivalent function of their -X flag counterparts. If both the environment variable and the -X flag are specified, the flag takes precedence over the environment variable. If no -X flag is given and OBJECT_MODE is unset, 32-bit mode is used. If OBJECT_MODE is set to an undefined value, an error message is printed, and the command fails unless the value is overridden on the command line.

Chapter 4. Application Development and Pthreads

This chapter details the changes in AIX 4.3 that may have an impact on the work of application developers. Applications may exhibit better performance by using new features and functions that are available in AIX 4.3.

4.1 C Language Standards

AIX Version 4 Release 3 made several changes and additions to conform to the ISO C Language Standard Normative Addendum One. The changes concern the handling of multibyte and wide character text formats.

The changes include new and altered programming interface specifications, many of which are contained in new #include files. For more information, see the National Language Support chapter of *AIX Version 4.3 General Programming Concepts: Writing and Debugging Programs*. The document is part of the online documentation, which is supplied with AIX. If you have not installed the documentation on a local machine, it can also be viewed on the internet using the URL: `http://www.rs6000.ibm.com/doc_link/en_US/a_doc_lib/aixgen/`

4.2 IEEE POSIX and UNIX98 Conformance

AIX 4.3 is now aligned with the following standards:

- ISO/IEC 9945-1:1996 that incorporates ANSI/IEEE Std POSIX 1003.1-1990, 1003.1b-1993, 1003.1c-1995, and 1003.1i-1995 (1003.1b-1993 and 1003.1i-1995 are realtime extensions; 1003.1c-1995 is a threads extension).
- ISO C Amendment 1: 1995 (multibyte support).
- The Open Group UNIX98 specification that adds:
  - Extended threads functions over POSIX threads, based on industry input from Sun, Digital, HP and DCE.
  - Dynamic linking extensions to permit applications to share common code across many applications and ease maintenance of bug fixes and performance enhancements for applications.
  - N-bit cleanup (64-bit and beyond) to remove any architectural dependencies in the UNIX specification. This is of particular relevance with the IBM move to 64-bit UNIX.
  - Year 2000 alignment to minimize the impact of the millennium rollover.

4.2.1 Realtime Options

AIX 4.3 does not support the realtime optional parts of the IEEE POSIX and UNIX98 specifications. In particular, the routines provided in Table 16 are not supported in AIX Version 4.3.

Table 16. Unsupported Real-Time Routines

clock_getres()	clock_gettime()
clock_settime()	fdatasync()
lio_listio()	mlock()

mlockall()	mq_close()
mq_getattr()	mq_notify()
mq_open()	mq_receive()
mq_send()	mq_settattr()
mq_unlink()	munlock()
munlockall()	nanosleep()
sched_get_priority_max()	sched_get_priority_min()
sched_getparam()	sched_getsceduler()
sched_rr_get_interval()	sched_setparam()
sched_setscheduler()	sched_yield()
sem_close()	sem_destroy()
sem_getvalue()	sem_init()
sem_open()	sem_post()
sem_trywait()	sem_unlink()
sem_wait()	shm_open() function
shm_unlink()	sigqueue()
sigtimedwait()	sigwaitinfo()
timer_create()	timer_delete()
timer_getoverrun()	timer_gettime()
timer_settime()	

4.2.2 Unsupported Threads Options

AIX 4.3.2 does not support the optional pthread interfaces provided in Table 17.

Table 17. Unsupported Optional Threads Interfaces

pthread_attr_getinheritsched()	pthread_attr_setinheritsched()
pthread_mutex_getprioceiling()	pthread_mutex_setprioceiling()

4.2.3 Dynamic Linking Extension

The dynamic linking extension that came out of the Aspen group comprises a set of four routines and a header file to provide a portable API for manipulation of an implementation-defined class of files, such as shared libraries. These routines are based on those introduced in UNIX System V Release 4.

Use of dynamic linking allows several benefits for application developers:

- The ability to share commonly-used code across many applications, leading to disk and memory savings.
- It allows the implementation of services to be hidden from applications.
- It allows the re-implementation of services. For example, to permit bug and performance fixes or to allow multiple implementations selectable at runtime.

4.2.3.1 dlopen()

The dlopen() function is used to dynamically load a module into a process' address space. The value returned by dlopen() is a handle that can be passed to dlsym() to look up symbols in the loaded module. The handle can also be passed to dlclose() to allow the module to be removed from the address space.

Synopsis: `#include <dlfcn.h> void *dlopen(const char *pathname, int flags);`

If the `<pathname>` is /unix, dlopen() returns a handle that can be used to look up symbols in the current kernel image, including all kernel extensions. If `<pathname>` is NULL, a handle for the main executable is returned. Otherwise, `<pathname>` names a module that will be loaded.

If `<pathname>` contains a slash character(/), the pathname is used directly, whether it is an absolute or a relative path. Otherwise, a search for the named module is made. Directories to be searched are listed in:

1. Value of LIBPATH when the process was first loaded
2. Current value of LIBPATH that can be modified during execution with the `setenv` command

The new module and its dependents are actually loaded with the load() system call. If the main program was built with the -brtl option, the runtime linker processes the loaded modules. Next, initialization routines are called for modules loaded for the first time.

If dlopen() succeeds, it returns a handle that can be used for calls to dlsym() and dlclose(). Otherwise, dlopen() returns NULL and sets errno. If errno is set to ENOEXEC, additional information can be obtained by calling dlerror().

4.2.3.2 dlsym()

This function returns the address of a symbol in a module opened by dlopen().

Synopsis: `#include <dlfcn.h> void *dlsym(void *handle, const char *name);`

The argument `<handle>` must be a value returned by dlopen() that has not been passed to dlclose(). The argument `<name>` is the name of a symbol or the special value RTLD_EP. For functions, the symbol name should not begin with a period.

If the `<name>` is RTLD_EP, the address of the entry point of the module is returned. If there is no entry point, the address of the data section of the module is returned. The returned value may be passed to loadbind().

In general, the module denoted by `<handle>` and its original dependents are searched in breadth-first search order, based on the import file IDs listed in each module. If a module is linked with the -brtl option or the -G flag, the dependency list will contain all modules listed on the command line in the same order. Otherwise, all dependent modules will be listed in an unspecified order.

If dlsym() succeeds, it returns the address of the desired symbol. Otherwise, NULL is returned.

4.2.3.3 dlclose()

This function is used to unload a module loaded by dlopen(). The function is implemented by calling unload(). If this is the last use of the module, it is removed

from the address space. Termination functions are called by unload() before the modules are actually unloaded.

The following is a synopsis of getdate():

```
#include <dlfcn.h> int dlclose(void *handle);
```

If dlclose() succeeds, 0 is returned. Otherwise, errno will be set to EINVAL, and EINVAL will be returned as well.

4.2.3.4 dlerror()

This function is used to return error information about the most recent call to dlopen(), dlsym(), or dlclose() call. If dlopen() fails and sets errno to ENOEXEC, dlerror() will return a pointer to a buffer describing reasons for the failure. In all other failing cases, errno will have been set, and dlerror() will return the formatted string corresponding to errno.

Synopsis: `#include <dlfcn.h> char *dlerror(void);`

Error information is reset after a call to dlerror(). Therefore, if two consecutive calls are made to dlerror(), the second call will return a pointer to a null string.

Note: The dlerror() function is not thread-safe since the string may reside in a static area that is overwritten whenever an error occurs.

4.2.4 Year 2000

The following APIs and commands were changed in accordance with the UNIX98 specification:

4.2.4.1 getdate()

The following is a synopsis of getdate():

```
struct tm *getdate(const char *string);
```

The entry for getdate() states the following with respect to the format code %y:

```
    "%y     year within century (00-99)"
```

%y is now defined such that, when a century is not otherwise specified, values in the range 69-99 refer to the twentieth century, and values in the range 00-68 refer to the twenty-first century. The %C specifier has been added to the interface to denote the century and interprets the %y specifier in the absence of a century as noted in the section above.

4.2.4.2 strptime()

The following is a synopsis of strptime():

```
char *strptime(const char *buf, const char *format, struct tm *tm);
```

The entry for strptime() states the following with respect to the format code %y:

"%y is the year within century [00,99]; leading zeros are permitted but not required

%y is now defined such that, when a century is not otherwise specified, values in the range 69-99 refer to the twentieth century, and values in the range 00-68 refer to the twenty-first century.

4.2.4.3 date Command

Century handling has been added as follows:

```
# date mmddhhmm[[cc]yy]
```

cc is the century specifier.

4.2.4.4 prs Command

The `prs` command is part of SCCS and has been changed such that the -c option

```
-c cutoff
```

indicates the cut off date-time, in the form:

YY[MM[DD[HH[MM[SS]]]]]

The YY specifier is a two digit specifier to the year and, therefore, does not denote the century. YY is now defined such that values in the range 69-99 refer to the twentieth century, and values in the range 00-68 refer to the twenty-first century.

4.3 M:N Pthreads (4.3.1)

AIX 4.3.1 replaced the previous 1:1 threads implementation model with an M:N version. The M:N model complies with the UNIX98 pthreads standard, which includes the POSIX pthreads standard. Previous releases of AIX Version 4 complied with Draft 7 of the POSIX pthreads standard. AIX 4.3.1 is binary compatible with previous releases. The UNIX98 implementation is the default for application development, but you can use the cc_r7 or xlc_r7 compiler interfaces to develop new applications using Draft 7 pthreads. Users may need to alter existing source code to obtain the required function on AIX 4.3.1 using the default UNIX98 pthreads library.

4.3.1 Porting Application from Draft 7 Pthreads

There are very few differences between Draft 7 and the final standard.

- There are some minor errno differences. The most prevalent is the use of ESRCH to denote the specified pthread could not be found. Draft 7 frequently returns EINVAL for this failure.
- Pthreads are joinable by default. This is a significant change since it can result in a memory leak if ignored.
- Pthreads have process scheduling scope by default.
- The subroutine pthread_yield has been replaced by sched_yield.
- The various scheduling policies associated with the mutex locks are slightly different.

4.3.2 The M:N Model

In the M:N model, there are two underlying types of pthreads. Those with PTHREAD_SCOPE_SYSTEM, or system scope contention, otherwise known as global threads, and those with PTHREAD_SCOPE_PROCESS, or process scope contention. These threads are known as local threads. There are also two types of thread schedulers in the system. The AIX kernel scheduler schedules all kernel threads. There is also a user thread scheduler, which schedules the local pthreads in a process.

Global threads are mapped 1:1 to kernel threads, and hence, are scheduled exclusively by the AIX kernel scheduler. The 1:1 threads model used by prior releases of AIX Version 4 only uses global threads.

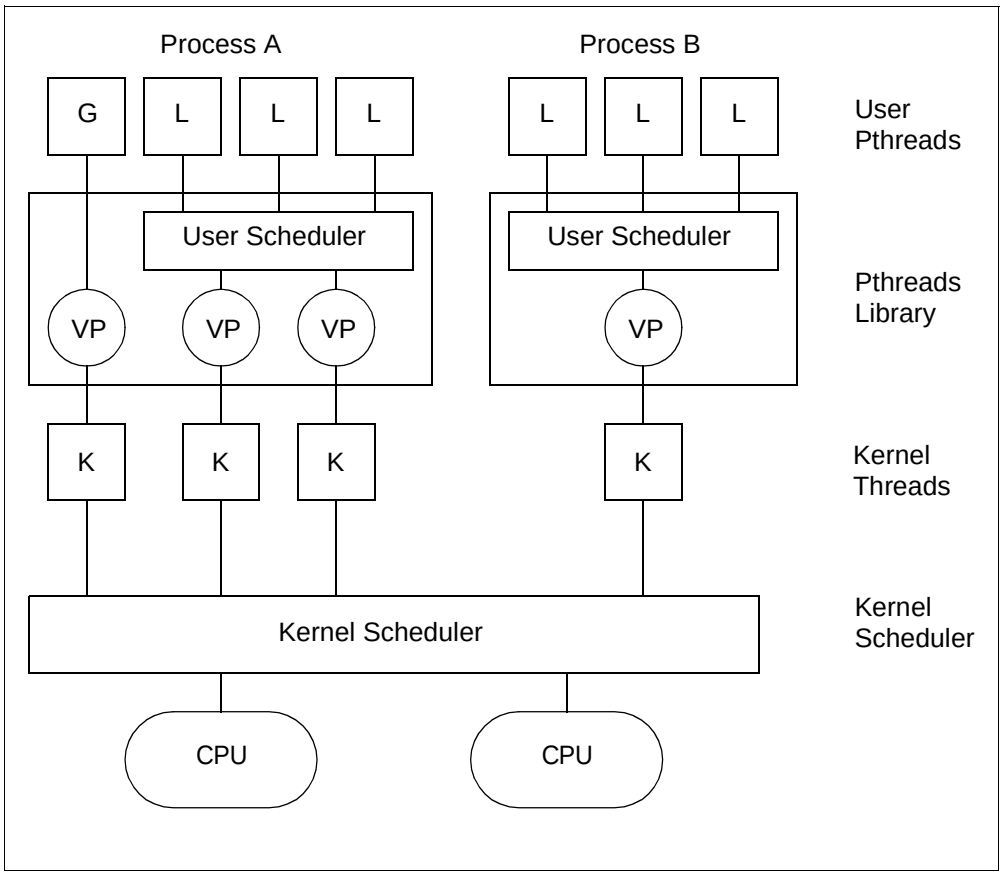


Figure 8. M:N Threads Model

The local pthreads are multiplexed over a set of kernel threads by the user scheduler, which is part of the pthreads library.

4.3.3 User Scheduler

The user scheduler is run on a dedicated *hidden* pthread, which is created when the pthreads library is initialized in M:N mode at process startup. There is one user scheduler for each process using the M:N model. The hidden pthread is created with system scope contention and therefore is scheduled directly by the kernel scheduler.

The user scheduler maintains a runqueue of runnable local pthreads and dispatches them on available kernel threads. Each kernel thread is represented in the pthreads library by a *virtual processor* structure (VP). There is a 1:1 mapping between VPs and kernel threads.

The user scheduler catches a SIGWAITING signal. Applications should not catch this signal, it is only for system use.

Each time a local pthread is created, terminates, or goes to sleep in the library, the user scheduler examines the ratio of kernel threads to active and sleeping user pthreads. If they are not consistent with the required values, then VPs, and hence, kernel threads, are created or destroyed as required. A VP that is to be deleted first places the user pthread it was running on to the queue of runnable pthreads maintained by the library. It then adds itself to the list of zombie VPs, and marks the underlying kernel thread for termination. The user scheduler traverses the list of zombie VPs on a regular basis and deletes the redundant VP structures.

Time slicing of local threads is initiated by the AIX scheduler, which sets a flag in the currently running kernel thread once it has obtained a full timeslice. On return from the clock tick interrupt handler, if the timeslice flag is set, the thread will call the user scheduler. The user scheduler places the current local thread on the local pthreads library runqueue and then selects the highest priority thread to run. If there are no threads on the run queue, then the current thread continues to run.

The user scheduler controls which pthreads are woken when a pthread event occurs. For example, when a mutex lock is released. The sleeping pthreads may have system-wide (global) or process-wide (local) contention scope. The user scheduler favors pthreads with system-wide scope over those with process-wide scope, regardless of their priorities. Priority is only used to decide between pthreads with the same contention scope. If they have the same priority, then the pthread that has been waiting the longest will be woken first.

When a local pthread makes a system call, it may block in the kernel waiting for a response from the system call. In this instance, the kernel thread and VP are not available to run another local pthread.

Consider a process with N+1 local threads, and N VPs, where one thread writes data to a pipe, and N threads read data from the pipe. The process would encounter a deadlock situation when the N threads reading from the pipe were blocked in the kernel. There would be no VP available for the N+1 thread to run on to write data to the pipe. This situation is avoided by a special check in the routine that a thread calls when about to block in the kernel. If the thread about to block is on the only VP of the process that is not already blocked, then the user scheduler is activated and instructed to create a new VP and kernel thread to run another local thread.

4.3.4 Mutex Locks

In previous versions of AIX, when a mutex lock is blocked, a pthread attempting to get the lock sleeps in the kernel. The internal structure of the mutex lock has been changed so that the list of threads waiting for the mutex is maintained in the user address space by the pthreads library. This is to allow the user scheduler to achieve relatively uniform levels of multiplexing over the remaining pthreads

sharing the VPs. When the mutex lock is freed, the user scheduler examines the list of threads waiting for the mutex and activates one of them.

4.3.5 Tuning

The M:N pthreads implementation provides several environment variables that can be used to affect application performance. If possible, the application developer should provide a front end shell script to invoke the binary executables in which the user may specify new values to override the system defaults. The following environment variables can be set by end users and are examined at process initialization time.

AIXTHREAD_SCOPE

This variable can be used to set the contention scope of pthreads created using the default pthread attribute object. It is represented by the following syntax:

```
AIXTHREAD_SCOPE=[P|S]
```

The value P indicates process scope, while a value of S indicates system scope. If no value is specified, then the default pthread attribute object will use process scope contention.

AIXTHREAD_MNRATIO

This variable allows the user to specify the ratio of pthreads to kernel threads. It is examined when creating a pthread to determine if a kernel thread should also be created to maintain the correct ratio. It is represented with the following syntax:

```
AIXTHREAD_MNRATIO=p:k
```

where `k` is the number of kernel threads to use to handle `p` pthreads. Any positive integer value may be specified for `p` and `k`. These values are used in a formula that employs integer arithmetic, which can result in the loss of some precision when big numbers are specified. If k is greater than p, then the ratio is treated as 1:1. If no value is specified, the default ratio depends on the default contention scope. If system scope contention is the default, the ratio is 1:1. If process scope contention is set as the default, the ratio is 8:1.

AIXTHREAD_SLPRATIO

This variable is used to determine the number of kernel threads used to support local pthreads sleeping in the library code on a pthread event. For example, attempting to obtain a mutex. It is represented by the following syntax:

```
AIXTHREAD_SLPRATIO=k:p
```

where `k` is the number of kernel threads to reserve for every `p` sleeping pthreads. Notice that the relative positions of the numbers indicating kernel threads and user pthreads are reversed when compared with AIXTHREAD_MNRATIO. Any positive integer value may be specified for `p` and `k`. These values are used in a formula that employs integer arithmetic, which can result in the loss of some precision when large numbers are specified. If k is greater than p, then the ratio is treated as 1:1. If the variable is not set, then a ratio of 1:12 is used.

The reason for maintaining kernel threads for sleeping pthreads is that, when the pthread event occurs, the pthread will immediately require a kernel thread to run on. It is more efficient to use a kernel thread that is

already available than it is to create a new kernel thread once the event has taken place.

AIXTHREAD_MINKTHREADS
This variable is a manual override to the AIXTHREAD_MNRATIO. It allows you to stipulate the minimum number of active kernel threads. The library scheduler will not reclaim kernel threads below this number.

SPINLOOPTIME
This variable controls the number of times the system will try to get a busy lock without taking a secondary action, such as calling the kernel to yield the processor. This control is really intended for MP systems where it is hoped that the lock is held by another actively running pthread and will soon be released. On uniprocessor systems, this value is ignored.

YIELDLOOPTIME
This variable controls the number of times that the system yields the processor when trying to acquire a busy mutex or spin lock before actually going to sleep on the lock. This variable has been shown to be effective in complex applications where multiple locks are in use.

4.3.6 Maximum Number of Threads

The maximum number of threads a single process can create has been increased to 32767. To obtain this limit, it is necessary to increase the data limit for the process since stacks are malloced out of the heap. If may also be necessary to use the large program model.

4.3.7 Combined Thread-Safe Libraries

Non-thread-safe and thread-safe libraries have been combined into one set of libraries, thereby turning thread-safety on by default.

AIX Version 4.2 libc.a (non-thread-safe) and libc_r.a (thread-safe).

AIX Version 4.3.1 libc.a which is thread-safe.

Libraries, such as X11R6, which link with libc.a are not thread-safe by default; they are thread-aware.

New libraries that are thread-safe include:

- libbsd.a
- libm.a
- libmsaa.a
- librts.a
- libodm.a
- libs.a
- libdes.a
- libXm.a
- libXt.a
- libX11.a

These thread-safe libraries enable a convenient programming model for exploiting SMPs and simplify exploitation of threads by applications, middleware, and other API providers.

4.4 Pthreads Suspend and Resume (4.3.2)

The pthreads library has been enhanced to provide the ability to suspend and resume individual threads. This function is added to AIX 4.3.2 to assist in the porting of applications from other platforms.

The pthreads implementation on AIX 4.3.2 complies with the UNIX98 standard. The four new API functions are not part of this standard, and this is indicated by appending `_np` to their names to indicate that they are NON-POSIX compliant.

The four new user functions are:

- `int pthread_suspend_np(pthread_t thread);`
- `int pthread_continue_np(pthread_t thread);`
- `int pthread_attr_setsuspendstate_np(pthread_attr_t *attr, int suspendstate);`
- `int pthread_attr_getsuspendstate_np(pthread_attr_t *attr, int *suspendstate);`

The pthread_suspend_np and pthread_continue_np functions are used to immediately suspend and resume execution of the thread indicated by the function argument.

The pthread_attr_getsuspendstate_np and pthread_attr_setsuspendstate_np functions are used to get and set the value of the new suspendstate member of the pthread_attr_t structure. The suspendstate argument can be set to either PTHREAD_CREATE_SUSPENDED_NP or PTHREAD_CREATE_UNSUSPENDED_NP. The default value of the suspendstate attribute of a pthread_attr_t structure is PTHREAD_CREATE_UNSUSPENDED_NP.

The new functions work in both the 1:1 and M:N threading environments.

4.5 Preserve Modified Ptrace Data (4.3.2)

AIX 4.3.2 has improved the performance of the `ptrace()` subroutine, which is used by debuggers to control the execution of applications under their control. Debuggers use a private copy of the text pages for the application being traced and any shared libraries it uses. This allows the debugger to modify the text pages to insert breakpoints without affecting any other processes on the system that may be running the same executable or shared library text.

Prior to AIX 4.3.2, when the application being debugged calls the load(), or loadbind() routines to load a private module into its address space, the system loader reloads fresh copies of all the text pages for the application and any required shared libraries. In so doing, any modifications made to the text pages are lost, so the debugger has to reinsert breakpoints after the application calls load() or loadbind().

The function of the ptrace routine has been modified along with the system loader to maintain ptrace altered copies of text pages across calls to load or loadbind. This will improve the performance of the debugger when controlling large

applications that call load or loadbind many times since breakpoints and other changes will not have to be reinserted.

4.6 Direct I/O

Direct I/O is a way of opening JFS files that allows for disk reads and writes using less CPU cycles than normal I/O. The main CPU savings come from avoiding the memory-to-memory copy done by the JFS. As the difference between memory and CPU cycle times increases, the savings achieved by avoiding this copy becomes greater.

Direct I/O offers a big performance gain for applications that do large block I/O (32 KB or greater) to JFS files and a smaller increase in performance for small block I/O. It does not improve raw I/O performance.

With normal I/O, the I/O request is first sent to the device driver. To service the request the device driver uses Direct Memory Access (DMA) to copy the data to or from pages in a file persistent segment. The data is then copied between the persistent segment and userspace through calls to vmcopyin() or vmcopyout(). Thus, a files persistent segment acts as a file cache.

With direct I/O, the data is not cached, but rather, I/O is done directly to a user supplied buffer through cross-memory technology. In other words, DMA is done directly from the disk to user space and conversely through the device strategy routine of the JFS file system.

Optimization was also made to the DMA setup routines. This improves large block I/O to JFS files and to raw logical volumes. However, the benefits of the DMA changes are much less than the benefits of direct I/O.

Take Note

It is important to note that since direct I/O reads are done synchronously and there is no read-ahead benefit, if they are not used correctly, they can also take much longer. The only read-ahead-like semantics that direct I/O can benefit from will be read-ahead performed by the disk device driver (normally 32 KB). For this reason, it is very important for a direct I/O reader to specify large read requests to match the performance of normal cached I/O readers. To match the performance of normal cached I/O readers, a direct I/O reader should issue read requests of at least 128 KB.

Direct I/O is considered *advisory*. This means that if a file is opened for direct I/O, and another application decides to open that same file for normal I/O, the file will be opened using normal cached I/O until direct I/O can be resumed (the normal I/O application closes the file).

Files can also be opened for a deferred update using the O_DEFER flag. If a file is opened with O_DEFER, and a subsequent open for direct I/O is issued, all writes will use normal cached I/O. Similarly, if another application opens the file with O_DEFER while it is already opened for direct I/O, all I/O to the file will be cached.

4.6.1 Opening Files for Direct I/O

A new flag, O_DIRECT has been defined in fcntl.h. When an application opens a file specifying or calling this flag through the fcntl() system call, I/O to this file will be done using direct I/O.

4.6.1.1 Inode Flags

When a file is using direct I/O, the i_flag field in the inode is set with the IDIRECT flag, defined in inode.h. Even so, it is not enough to simply have a flag in the inode to determine if the file is using direct I/O or not. If a normal I/O application opens the file while a direct I/O application currently has it open, then all I/O will be done using normal I/O until the normal I/O reader or writer closes the file. A count of direct I/O readers is maintained to determine if the direct I/O can be resumed. A new field in the inode, i_diocnt, has been added for this purpose. This field indicates if any application has the file opened for direct I/O.

4.6.2 JFS Function Calls for Direct I/O

There are only a few functions that were affected in the JFS for direct I/O to be implemented. These functions are serialized by the inode lock and are described below.

jfs_map() If a file opened for direct I/O is mapped, the IDIRECT flag is reset, and all subsequent I/O will be done using normal I/O. If the mapped file is then closed, direct I/O will be resumed.

jfs_close() Close semantics are closed with direct I/O. When the final close occurs, (checked by the counts on gnode) the IDIRECT flag in the inode is turned off. If a close is initiated by a normal I/O reader or writer and another application opens the file for direct I/O, all cached pages are flushed to disk, and direct I/O is resumed on the file.

jfs_dio() jfs_dio() is called from jfs_rdwr(). If the FDIRECT flag is set, jfs_dio() evaluates if direct I/O can be done for a particular file. This function performs all alignment and file state consistency checking.

If a read or write request cannot be done using direct I/O, jfs_dio() returns a non-zero return code, and the request is done through normal cached I/O.

4.6.3 System Archive Utilities

Archive commands are typical applications that can benefit from the use of direct I/O. Therefore, the standard system commands `tar`, `backup`, `restore`, and `cpio` have been enabled to use direct I/O. Since these commands are *read-once* commands, that is, they do not reference the data again after it has been read and written to media, the copyin() and copyout() characteristics of normal cached I/O consume a lot of unnecessary CPU when these commands are executing. The enabling of these commands has been accomplished by changing all calls to open() and setting the O_DIRECT flag.

4.7 Shared Memory Enhancements

The following enhancements reflect changes to the shared memory function in AIX.

4.7.1 Larger Shared Memory Regions (4.3.1)

The maximum size of a shared memory region created by the shmget() routine and attached to a process' address space by the shmat() routine has been increased from 256 MB to 2 GB. Prior to AIX 4.3.1, it was possible to mmap() a memory mapped file of up to 2 GB, but an anonymous memory region was limited to 256 MB. This meant that a large memory region had to be created in several 256 MB portions and each portion attached individually. AIX 4.3.1 has removed this restriction, so it is now possible to attach a 2 GB memory region with one call to the shmat() routine.

If EXTSHM=ON is set and an application performs a shmget() with a size greater than SEGSIZE-PAGESIZE, the system will use the traditional shmat() and not mmap() as would be the case when EXTSHM=ON.

4.7.2 128 KB Shared Memory IDs (4.3.2)

AIX 4.3.2 now supports 128 KB mem, sem, and shm IDs, up from 4 KB in the previous releases.

4.7.3 Shared Memory Debugging Enhancements (4.3.2)

AIX 4.3.2 has added the facility for additional information to be included in the core file that may be produced by an application program when it encounters certain types of error.

There are two methods of enabling the extra information to be included in the core dump.

- The application can use the sigaction routine to set the SA_FULLDUMP flag for the signal that will cause the core file to be generated.
- Enable full core information as the default, either from the SMIT Change / Show Characteristics of Operating System panel, the WebSM Devices panel by selecting sys0, or by using the command:

  ```
  chdev -l sys0 -a fullcore='true'
  ```

When an application faults, and a full core is generated, the core will include all the shared memory regions of the faulting process that are currently attached.

The dbx debugger has been changed to understand the extra information in the core file and allow the developer to interrogate the user defined variables contained in the shared memory regions of the process at the time of termination.

4.8 DMA Pre-Translation (4.3.2)

DMA pretranslation of memory buffers reduces the cost of setting up DMA operations. Its objective is to reduce DMA setup pathlength for selected and predetermined I/O operations to improve performance. The enhances the

performance of network memory buffers (mbufs), filesystem I/O, raw I/O, and page I/O.

In previous AIX versions, during a DMA operation, the majority of pathlength was spent in page translation/lookup paths to get the virtual to physical address translations for DMA.

The term *pretranslation* refers to the concept of performing the virtual to physical address translations for a data buffer to be involved in a DMA operation once, for the life of the data buffer, instead of for each individual I/O setup for the buffer. In general, a subsystem desiring to benefit from the performance gain of pretranslation calls a new kernel service passing in a buffer address, length, and cross-memory descriptor. The kernel service will attach to the cross memory descriptor pretranslation information for the buffer. Then, whenever the buffer is used for I/O, the DMA services recognize the presence of pretranslation info in the cross-memory descriptor and avoid page table lookups.

There is no change required by device drivers to take advantage of this enhancement, as long as a network driver is using mbufs from the global net_malloc pool and performing dynamic on-the-fly DMA mappings (compared with copying data to premapped buffers).

4.9 Fast fork() Function (4.3.1)

AIX Version 4.3.1 introduces a fast-fork function called f_fork() that is based on IEEE POSIX specifications. The f_fork() call is precisely like fork() except:

- It is required that the child process calls one of the exec functions immediately after it is created. Since fork handlers are never called, the application data, mutexes, and the locks are all undefined in the child process.

The use of f_fork() will significantly enhances the performance of Internet and Web server applications that need to create many short lived child processes.

4.10 New Sockets System Call (4.3.2)

AIX 4.3.2 has added the new send_file() system call. Its use is aimed at applications that transmit file data across the network using a socket connection. It offers a speed improvement over the traditional method of sending a file across the network by avoiding unnecessary data copying where possible.

```
#include <sys/socket.h>

ssize_t send_file(Socket_p, sf_iobuf, flags)

int *Socket_p;
struct sf_parms *sf_iobuf;
uint_t flags;
```

Using send_file eliminates the need to read a file just to send it across the network. Applications can remove the read() call, and therefore avoid redundant transfer of data between kernel space and user space. The send_file call reads file data into the new Network Buffer Cache (NBC). The NBC is allocated from an area of the mbuf pool and uses mbufs for file data storage. The networking subsystem then transmits the data directly from the mbufs in the NBC across the

specified socket. The system call dynamically caches the data in the NBC, thus improving performance for files that are sent frequently across the network, and which do not change often. This feature can be disabled on a file by file basis.

The application sending the data will need to be altered to use the send_file call. The greatest improvement in performance can be gained by using direct I/O to read the file that is to be transmitted. This can be achieved simply by opening the file using the O_DIRECT flag. This flag enables send_file to bypass the JFS cache when reading the file, thus further reducing the number of data transfers required.

The size of the NBC, and various cache tuning parameters can be altered using the no command. The options that can be changed are:

nbc_limit Maximum size of the NBC. Specifies in KB the maximum amount of memory that can be used for the NBC. The default value is derived from the size of the mbuf pool (thewall), which in turn, is determined from the amount of physical memory. If a system has less than 512 MB of memory, the default value of nbc_limit is 0.

nbc_max_cache Maximum size of a cache object in the NBC. Specified in bytes, default value is 131072 (128 KB) bytes.

nbc_min_cache Minimum size of a cache object in the NBC. Specified in bytes, default value is 1.

send_file_duration Specifies the cache validation duration for all the file objects that system call send_file accessed in the Network Buffer Cache. This attribute is in number of seconds, the default is 300 for 5 minutes. 0 means that the cache will be validated for every access.

Cache statistics can be viewed using the command `netstat -c`. Sample output is shown in Figure 9.

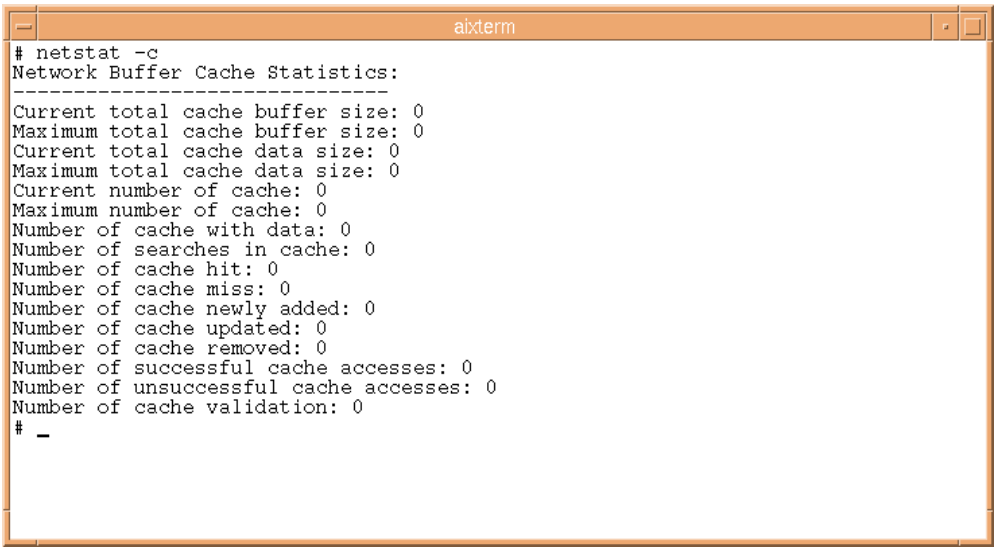


Figure 9. Sample Output from netstat -c Command

The send_file call only supports the TCP/IP protocol. In other words, the sockets of type SOCK_STREAM in address family AF_INET. Both blocking and non-blocking connections are supported. In blocking mode, send_file blocks until all of the file data has been transmitted. In non-blocking mode, or in the event send_file is interrupted, the system call updates parameters in the sf_parms structure to indicate the amount of data that has actually been transmitted.

4.11 Binder Library Enhancements (4.3.2)

The binder library, libld.a, that provides functions to allow an application to create and manipulate XCOFF object files, has been enhanced in AIX Version 4.3.2 to support a cross mode environment.

This allows 32-bit applications to create and manipulate both 32-bit and 64-bit objects using a consistent interface. The changes also allow 64-bit objects to create and manipulate both 32-bit and 64-bit objects. The functions in the library transparently open both 32-bit and 64-bit object files, as well as both small format and large format archive files.

An application need not know the format of an object file before opening it. It can call the ldopen function and then check the magic number of the file or archive member.

Chapter 5. Logical Volume Manager Enhancements

AIX 4.3, AIX 4.3.1, and AIX 4.3.2 received enhancements to the logical volume group scalability, synchronization performance, and on-line mirror backup functions. These changes enhance AIX's image as a robust and powerful operating system for increasingly demanding customer requirements.

In this chapter, the major new features of logical volume manager are described.

5.1 Logical Volume Synchronization

The following commands now support the -P flag to allow the user to specify the number of LPs to sync. in parallel.

- `/usr/sbin/syncvg`
- `/usr/sbin/lresynclv`

The -P flag is followed on the command line by the number of partitions to be synchronized as follows:

```
syncvg [-i] [-f] [-H] [-P NumParalleLPs] {-l|-p|-v} Name[-P num_parallel_lps]
lresynclv [-H] [-P NumParalleLps] -l LVid
```

The valid range for NumParallelLps is 1 to 32. If the number entered is less than one then `num_parallel_lps` defaults to one. If the number entered is greater than 32 then `num_parallel_lps` will be set to 32.

The `mklv` and `chlv` commands were updated in AIX 4.3.0 to allow synchronized updates of volume groups in a concurrent environment. All nodes that share disks must be available at the time the updated command is issued in order for updates to take place. If a system already has an existing LV with the same name as a new one being added to another system, the command will fail. Other conflicts are also detected to provide stable LV updates in a shared environment. All systems must be running AIX 4.3.0 or higher in order to use this enhancement.

5.2 importvg Learning Mode

A new option has been created for the LVM `importvg` command. This new option, -L for learning mode, is executed on a shared volume group in a cluster. It allows the LVM actions of creation, deletion, or extension performed on one cluster node to be propagated to other nodes connected to the same shared volume group.

```
importvg [-V MajorNumber] [-y  VolumeGroup] [-f] [-c] [-x] | [-L VolumeGroup] [
-n] [-F] PhysicalVolume
```

The -L flag takes a volume group and learns about possible changes performed to that volume group. Any new logical volumes created as a result of this command inherit the ownership, group identification, and permissions of the /dev special file for the volume group listed in the -y flag.

To use this feature, note the following:

- The volume group must not be in an active state on the system executing the -L flag.

- The volume group's disks must be unlocked on all systems that have the volume group varied on and operational. Volume groups, and their disks, may be unlocked, remain active, and used through the `varyonvg -b -u` command.
- If an active node has both added and deleted logical volumes on the volume group, the -L flag may produce inconsistent results. The -L flag should be used after each addition or deletion rather than being deferred until after a sequence of changes.

Figure 10 shows an example of a multi-tailed system.

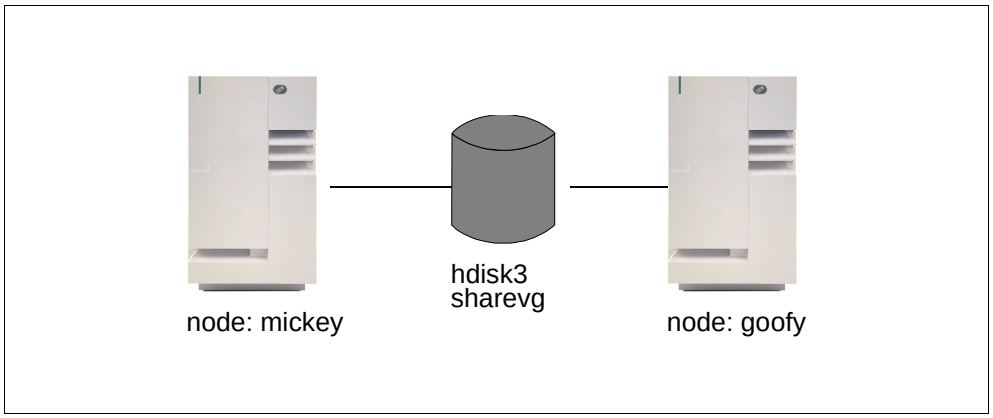


Figure 10. Importvg -L Example

There are two machines, goofy and mickey, that share one disk. The volume group sharevg is created on the shared disk. Both goofy and mickey are aware of the sharevg volume group. The sharevg in node mickey is varied on and in node goofy, it is varied off.

On node mickey, to make the sharevg unlocked, enter:

```
#varyonvg -b -u sharevg
```

On node goofy, to read the LVM information made by node mickey, enter:

```
# importvg -L sharvg datavg hdisk3
```

On node mickey, to return to the normal mode and release the lock, enter:

```
# varyonvg sharevg
```

It should be noted that the volume group sharevg remained on line during the entire operation, therefore, not affecting production work.

5.3 importvg Fast Mode

A new option -F has been added to `importvg` command. The command syntax is shown in the following example.

```
importvg [-V MajorNumber] [-y  VolumeGroup] [-f] [-c] [-x] | [-L VolumeGroup] [
-n] [-F] PhysicalVolume
```

It provides a fast version of importvg that checks the Volume Group Descriptor Areas (VGDA) of only the disks that are members of the designated volume group. As a result, if a user exercises this flag, they must ensure that all physical volumes in the volume group are in a good and known state. If this flag is used on a volume group where a disk may be in missing or removed state, the command may fail, or the results may be inconsistent. This flag has the advantage of avoiding a lengthy search for missing disks that happens during normal `importvg` processing. Administration of large SSA disk arrays will greatly benefit from the terse search the -F option provides.

5.4 Raw LV Online Mirror Backup Support (4.3.1)

The LVM in AIX 4.3.1 provides a snap shot capability for raw mirrored logical volumes. One mirror of a mirrored logical volume can be used to archive the data on the raw logical volume without splitting the mirror copies from each other (only the logical partitions that have changed during the system backup need to be resynchronized).

Table 18 lists the new options added to `chlvcopy` command.

Table 18. chlvcopy New Options in AIX 4.3.1

Flag	Description
-b	Mark a mirror copy as an online backup copy.
-c	Identify which mirror copy used as on-line backup copy. The allowed values of copy are 1, 2, or 3. If this option is not specified, the default for copy is the last mirror copy of the logical volume.
-B	Unmark a mirror as an online backup copy.
-f	Force LV copy to be marked as backup even if there are stale partitions.

5.4.1 Removal of 1016 PPs per Physical Volume Limit (4.3.1)

The support for greater than 1016 PPs per physical volume has been added in AIX Version 4.3.1. To support a VG exceeding the limit of 1016 PPs using the same VGDA and VGSA areas, the number of disks supported in the volume group has been reduced.

The -t flag was added to the `chvg` and `mkvg` commands to convert and create a volume group with multiples of 1016 partitions per disk. This reduces the total number of disks that can be added to the volume group by same fraction. Once a volume group is changed or created to hold more than 1016 physical partitions per disk, it cannot be imported into AIX versions earlier than 4.3.1.

The -t factor allows (factor * 1016) PPs per physical volume. For example, a partition size of at least 16 MB would be needed to create a volume group with a 10 GB disk. Or with at factor size of 2, a smaller partition size of 8 MB can be used. However, this limits the total number of disks that can be added to the volume group. If a factor value is used, a maximum of MAXPVS/factor disks can be included in the volume group.

The relationship of factor -t, PP numbers per physical disk, and the number of disks allowed in one VG is provided in Table 19.

Table 19. Factor -t

Factor t	PP numbers	Number of disks in VG
1	1016	32
2	2032	16
3	3048	10
4	4064	8
5	5080	6
6	6096	5
7	7112	4
8	8128	4
16	16256	2

Following is an example of a 4.3 GB disk. The PP size 8 MB is needed for creating the volume group vg1 without the factor -t:

```
# lsvg vg1
VOLUME GROUP:   vg1                  VG IDENTIFIER:00091974e54218d7
VG STATE:       active               PP SIZE:      8 megabyte(s)
VG PERMISSION:  read/write                TOTAL PPs:      537 (4296 megabytes)
MAX LVs:        256                        FREE PPs:       537 (4296 megabytes)
LVs:            0                    USED PPs:       0 (0 megabytes)
OPEN LVs:       0                    QUORUM:         2
TOTAL PVs:      1                    VG DESCRIPTORS: 2
STALE PVs:      0                    STALE PPs:      0
ACTIVE PVs:     1                    AUTO ON:        yes
MAX PPs per PV: 1016                 MAX PVs:        32
```

When using the factor -t to create vg1, the PP size of 4 MB can be used. Then, the maximum PPs per physical volume becomes 2032, and the maximum physical volumes allowed in the volume group is 16. Following is an example:

```
# mkvg -t 2 -y vg1 hdisk4
0516-1193 mkvg: WARNING, once this operation is completed, volume group vg1
        cannot be imported into AIX 430 or lower versions. Continue (y/n) ?
y
0516-631 mkvg: Warning, all data belonging to physical
        volume hdisk4 will be destroyed.
mkvg: Do you wish to continue? y(es) n(o)? y
vg1

# lsvg vg1
VOLUME GROUP:   vg1                  VG IDENTIFIER:  00091974e54743e9
VG STATE:       active               PP SIZE:        4 megabyte(s)
VG PERMISSION:  read/write           TOTAL PPs:      1075 (4300 megabytes)
MAX LVs:        256                  FREE PPs:       1075 (4300 megabytes)
LVs:            0                    USED PPs:       0 (0 megabytes)
OPEN LVs:       0                    QUORUM:         2
TOTAL PVs:      1                    VG DESCRIPTORS: 2
STALE PVs:      0                    STALE PPs:      0
```

```
ACTIVE PVs:      1                      AUTO ON:        yes
MAX PPs per PV: 2032                    MAX PVs:        16
```

Note

When using the -t flag with `chvg` and `mkvg`, it must be entered from the command line.

5.5 Physical Partition Support (4.3.1)

The support for physical partition sizes of 512 MB and 1024 MB have been added to AIX 4.3.1. Volume groups created with this new size cannot be imported into previous versions of AIX.

In pre-AIX 4.3.1 versions, if you add a new volume group, the physical partition sizes in megabytes you are allowed to chose are: 1, 2, 4, 8, 16, 32, 64, 128, and 256.

For AIX 4.3.1, and later releases, if you add a new volume group, the physical partition sizes in megabytes you are allowed to chose are: 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, and 1024 MBs.

If you select a physical partition size of 512 or 1024 MBs on a AIX 4.3.1 or later system, the volume group created cannot be imported on older versions of AIX.

5.6 Big Volume Groups (4.3.2)

A new volume group (VG) format is added in AIX 4.3.2, which increases the maximum number of disks that can be included in a volume group from 32 to 128. The maximum number of logical volumes in this new volume group format is increased from 256 to 512. This means:

- The maximum physical volume (PV) number in one volume group is increased from 32 to 128 (1024 in the future)
- The maximum logical volume (LV) number in one volume group is increased from 256 to 512 (1024 in the future)

Table 20 provides information about LVM restrictions in several AIX versions.

Table 20. Limitations of LVM

Version	Volume group (per system)	Physical volume (per volume group)	Physical partition (per physical volume)	Logical volume (per volume group)	Logical partition (per logical partition)
AIX 4.3.2	255	128	1016*-t factor	512	32,512
AIX 4.3.1	255	32	1016*-t factor	256	32,512
AIX 4.3.0	255	32	1016*-t factor	256	32,512
AIX 4.2	255	32	1016	256	32,512

The LVM in AIX 4.3.2 supports both the small VG configurations of the previous versions of AIX and the new big VG configuration. A migration path is provided to convert old volume groups to the new volume group format, provided there are sufficient free partitions on each of the physical volumes in the volume group to be allocated.

The following sections explain the changes for the bigger VGDA/VGSA, which describe a volume group to a system completely. The changes needed for commands, library, and the LV device driver are also discussed.

5.6.1 Changes to LVCB

The original design of the VGDA and VGSA limited the number of disks that can be added to the volume group at 32 and the total number of logical volumes at 256 (including one reserved for LVM internal use). With the increasing use of disk arrays, the need for the increased capacity for a single volume group is greater.

Following are the basic concepts of VGDA, VGSA, and LVCB.

VGDA Stands for volume group descriptor area. 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 VGDA is stored on each physical volume of the volume group.

VGSA Stands for volume group status area. 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. The VGSA is stored on each physical volume of the volume group.

LVCB Stands for logical volume control block. The LVCB is the first 512 bytes of a logical volume. This area holds important information, such as the creation date of the logical volume, information about mirrored copies, and possible mount points in the journaled filesystem (JFS). Certain LVM commands are required to update the LVCB as part of the algorithms in LVM.

The Logical Volume Control Block has been moved from the first block of the logical volume to inside the VGDA for better preservation. Though database programs that use logical volumes as raw devices skip this block, obliteration of the LVCB has caused confusion and loss of information such as intra-policy, inter-policy, upperbound, and so on. Since other subsystems, such as diagnostics, IPL, and ROS, do use the LVCB without using the LVM access routines, the LVCB will be maintained at both places.

5.6.2 General Enhancements for Big VG

The following sections describe the general limitations and updates required to implement big VG support on AIX.

5.6.2.1 Commands Changes

To support the big VG format, some new options have been added to the commands `mkvg`, `chvg`, `importvg`, `mklv`, and `chlv` commands.

The mkvg Command

The following lists some of the major changes to the `mkvg` command.

- The new option -B creates a big VG format volume group. This can accommodate up to 128 physical volumes and 511 logical volumes (one reserved for LVM internal use).
- If you do not use the -B option, the `mkvg` command will create the a VG with 1016*factor(-t) physical partitions and 32/factor(-t) disks per volume group.
- The option -G creates the volume group with enough space reserved at the beginning of the disk to expand the VGDA to include 1024 disks in the future without having to migrate the physical partitions.

Following is an example of the `mkvg` command. To create a big VG, testvg, on hdisk1, enter:

```
# mkvg -B -y testvg hdisk1
```

To see the attributes of this VG, enter:

```
# lsvg testvg
VOLUME GROUP:   testvg              VGIDENTIFIER:061515169c44a3e
VG STATE:       active              PP SIZE:        4  megabyte(s)
VG PERMISSION:  read/write          TOTAL PPs:      80  (320megabytes)
MAX LVs:        512                 FREE PPs:       80  (320 megabytes)
LVs:            0                   USED PPs:       0  (0 megabytes)
OPEN LVs:       0                   QUORUM:         2
TOTAL PVs:      1                   VG DESCRIPTORS: 2
STALE PVs:      0                   STALE PPs:      0
ACTIVE PVs:     1                   AUTO ON:        yes
MAX PPs per PV: 1016                MAX PVs:        128
```

This example shows the new limit values for big VG: MAX PVs is 128 and MAX LVs is 512.

The chvg Command

The option -B converts a small VG to a big VG. Once all the logical volumes are in closed/synced state (file systems unmounted), and if all the physical volumes are in the ACTIVE state in the volume group, the -B flag can be used to convert the small VG to a big VG format. This operation expands the VGDA/VGSA to change the total number of disks that can be added to the volume group from 32 to 128.

If you want to convert the rootvg, you will get the following error message:

```
# chvg -B rootvg
0516-1212 chvg: rootvg cannot be converted to the big volume group format.
0516-732 chvg: Unable to change volume group rootvg.
```

If both -t and -B flags are specified, factor will be updated first, and then the VG is converted to the big VG format (sequential operation).

First create a small VG, testvg, on hdisk1:

```
#mkvg -y testvg hdisk1
```

To see the small VG information, enter:

```
#lsvg hdisk1
VOLUME GROUP:   testvg            VG IDENTIFIER:  00615151692724b1
VG STATE:       active            PP SIZE:        4 megabyte(s)
VG PERMISSION:  read/write        TOTAL PPs:      81 (324 megabytes)
```

```
MAX LVs:        256             FREE PPs:       81 (324 megabytes)
LVs:            0               USED PPs:       0 (0 megabytes)
OPEN LVs:       0               QUORUM:         2
TOTAL PVs:      1               VG DESCRIPTORS: 2
STALE PVs:      0               STALE PPs:      0
ACTIVE PVs:     1               AUTO ON:        yes
MAX PPs per PV: 1016            MAX PVs:        32
```

To convert a small VG into a big VG:

```
# chvg -B test
0516-1224 chvg: WARNING, once this operation is completed, volume group test
cannot be imported into AIX 431 or lower versions. Continue (y/n) ?
y
0516-1164 chvg: Volume group testvg changed. With given characteristics testvg
can include up to 128 physical volumes with 1016 physical partitions each
physical volume.
```

To see the attributes of this VG, enter:

```
# lsvg test
VOLUME GROUP:   test            VG IDENTIFIER:  00615151692724b1
VG STATE:       active          PP SIZE:        4 megabyte(s)
VG PERMISSION:  read/write      TOTAL PPs:      81 (324 megabytes)
MAX LVs:        512             FREE PPs:       79 (316 megabytes)
LVs:            0               USED PPs:       2 (8 megabytes)
OPEN LVs:       0               QUORUM:         2
TOTAL PVs:      1               VG DESCRIPTORS: 2
STALE PVs:      0               STALE PPs:      0
ACTIVE PVs:     1               AUTO ON:        yes
MAX PPs per PV: 1016            MAX PVs:        128
```

As shown, the number of TOTAL PPs remain unchanged. The number of free PPs are reduced by two. These two PPs are reserved for the larger VGDA/VGSA.

If you do not have enough space on your disk, suppose disk1 on the small VG is full, such as:

```
# lspv hdisk1
PHYSICAL VOLUME:    hdisk1              VOLUME GROUP:     testvg
PV IDENTIFIER:      00615151648abe10    VG IDENTIFIER:006151516a0af1a9 PV
STATE:              active
STALE PARTITIONS:   0                    ALLOCATABLE:      yes
PP SIZE: megabyte(s)                     LOGICAL VOLUMES:  2
TOTAL PPs:          81 (324 megabytes)   VG DESCRIPTORS:   2
FREE PPs:           0 (0 megabytes)
USED PPs:           81 (324 megabytes)
FREE DISTRIBUTION:  00..00..00..00..00
USED DISTRIBUTION:  17..16..16..16..16
```

The following example shows what happens if you want to convert the small testvg into big testvg:

```
# chvg -B testvg
0516-1214 chvg: Not enough free physical partitions exist on hdisk1 for the
expansion of the volume group descriptor area. Migrate/reorganize to free up 2
partitions and run chvg again.
0516-732 chvg: Unable to change volume group testvg.
```

You must migrate or reorganize the volume group using `migratepv` or `reorgvg` to free up enough physical partitions for the system to expand the VGDA/VGSA.

The importvg Command
The option -R restores the ownership, group ID, and permissions of the logical volume special device files. These values will be restored only if they were set using -U, -G, and -P flags of `mklv` or `chlv` commands. The -U, -G, and -P flags are for root to define the ownership, group, and permissions of the LV you are creating respectively. This flag is applicable only for big VG format volume groups.

The mklv Command
If you create a logical volume in a big VG, you can use the following three new options (using root privileges):

- Option -U specifies the user ID for logical volume special file
- Option -G specifies the group ID for the logical volume special file
- Option -P specifies the permissions (file modes) for the logical volume special file.

The chlv Command
The three new options are the same with `mklv` command (using root privileges):

- Option -U specifies the user ID for logical volume special file
- Option -G specifies the group ID for the logical volume special file
- Option -P specifies the permissions (file modes) for the logical volume special file.

Note

When using the above new options in `mkvg`, `chvg`, `importvg`, `mklv`, and `chlv` commands, the commands must be entered from the command line. There are no `smit` or `wsm` interfaces for them.

5.6.2.2 Header File Changes
The following header file was changed to support big VGs:

- lvmrec.h

5.6.2.3 Default Maximum PPs for Each Physical Volume - 1016
No matter if you create a big VG or small VG, the `mkvg` command will still be using 1016 as the default value for the number of physical partitions per physical volume. If you use the -t (factor) option together with the big VG option, you can create the volume group with the desired partition size and number of partitions.

The -t volume group factor was first introduced in AIX 4.3.1. See 5.5, “Physical Partition Support (4.3.1)” on page 87 for reference. The number of physical partitions calculated is 1016 * t factor per physical volume. The size for each of the physical partition is up to 1024 MB.

The `mkvg` command, by default, creates a volume group that can accommodate 255 logical volumes and 32 physical volumes (disks). These limits can be

extended to 511 logical volumes and 128 physical volumes by specifying the -B flag.

5.6.3 Small VG to Big VG Conversion

To convert the small VG to a big VG, a number of free physical partitions are needed to expand the VGDA/VGSA. Depending on the size of the physical partition and the current size of the VGDA, the number of partitions required are calculated. Since the first partition starts immediately after the end of the secondary VGDA, if it is occupied by a logical partition, it will be migrated to another free physical partition on the disk. This first physical partition will then be removed from the list of available partitions (not be moved or allocated for any reason), and the remaining partitions will be renumbered. After the conversion, the total number of physical partitions on the disk will not change, except that the extra partitions allocated for the VGDA are marked non allocatable.

You should be aware of the following items when you perform a conversion from a small VG to a big VG:

- All the disks in the volume group should have enough free PPs for the conversion to be possible.
- All the logical volumes in the volume group must be in closed/synced state.
- All physical volumes must be available and be in the ACTIVE state.
- The volume group is varied on in management mode to prevent opening of any logical volumes.
- The ownership/permissions of special device files will only be preserved if `mklv` or `chlv` are used with the -U, -G, or -P flags.

Currently, the `migratepv` command does not allow the migration of individual physical partitions. The conversion needs to free up just enough physical partitions from the beginning of the disk to elsewhere. The current implementation will try to migrate the partitions within the physical volume, and the user must move the partitions to other disks in the volume group.

5.6.4 Big VG Limitations

The following list are the limitations of a big VG:

- A big VG is not enabled for concurrent access. This is posed by the communication path used by the concurrent logical volumes. It will be prohibitively slower for the big VG to communicate across nodes due to an increase in the number of disks.
- The rootvg cannot be converted to the big VG format.
- A big VG cannot be imported or activated on pre-AIX 4.3.2 levels.

5.7 Concurrent Online Mirror Backup and Special File Support (4.3.2)

AIX 4.3.1 provided support for an on-line backup mechanism for a mirrored raw logical volume. But it lacked support for file system access and restricts concurrent mode access for the volume group.

AIX 4.3.2 enhances the capabilities of the online backup in AIX 4.3 to support:

- Concurrent mode volume groups
- Filesystem and database access

Note that filesystem access does not mean JFS access. This enhancement to the LVM still requires additional steps (such as unmounting a file system) to prevent data loss.

This new feature is used for HACMP concurrent mode (mode 3). While in concurrent mode, you can designate a mirror copy in a VG as a backup copy to archive the data on the raw logical volume without affecting the other mirror copies. It improves the system availability to end users.

Use a second LV and special device to allow access to the backup mirror copy. All I/O and ioctls coming to this second LV would be routed to the actual logical volume to be serviced. A number of changes were made to VGDA to support this new type of LV.

If the LV contains a filesystem, there will be two serial writes to support the new mountpoint. In order to support this function, there are updates to the LVCB and the superblock of the new filesystem.

5.7.1 Limitations

Following are some limitations:

- The original logical volume cannot be removed while the on-line backup copy exists. No changes are allowed to the original logical volume structure, or attributes, while the on-line backup exists. But you still can make changes to the filesystem mounted over the logical volume.
- All partitions of a logical volume must be fresh before you mark a mirror copy as an on-line backup. Only one copy may be designated as an on-line backup copy.
- This function is currently documented only in the man pages. Use at your own risk.

5.7.2 Commands Changed

Using the `chlvcopy` command, you can mark, or unmark, a mirror copy as an on-line backup copy and change the backup mirror copy characteristics for a logical volume.

The syntax is:

```
chlvcopy -B | { -b [ -c copy ] [ -f ] [ -P ] [ -l newlvname ] [ -w ] } LV name
```

Note

Although the chlvcopy command can mark on-line backup copies on logical volumes that are open (including logical volumes containing mounted file systems), this is not recommended unless the application is at a known state at the time the copy is marked as a backup. The backup copy is internally consistent at the time the chlvcopy command is run, but consistency is lost between the logical volume and the backup copy if the logical volume is accessed by multiple processes simultaneously, and the application is not at a known state. When marking an open logical volume, data may be lost or corrupted. Logical volumes should be closed before marking on-line backup copies in order to avoid a potential corruption window.

The -P,-l, and -w are new options for AIX 4.3.2.

If the persistence flag -P is not set to prevent the loss of backup data, the volume group should be set to not automatically varyon, and the -n flag should be used with varyonvg to prevent stale partitions from being resynced. If the persistence flag -P is set, the following applies: in the event of a crash while an on-line backup copy exists (or multiples exist), the existence of copies is retained when the system is rebooted.

Use the -l or -P flag to prevent the volume group from being unstable on prior releases of AIX.

Table 21 lists the new options of chlvcopy command in AIX 4.3.2.

Table 21. New Options for chlvcopy Command in AIX 4.3

Flag	Description
-P	Maintains information about the existence of an online backup copy across a reboot and also allows other nodes (in a concurrent mode environment) to be aware of the existence of the online backup(s).
-l	New name of the backup logical volume. If one is not provided, one will be created by the system.
-w	Allow backup copy to be writable (default is to create the backup copy as READ ONLY)

To create an on-line backup, perform the following steps:

1. Unmount the filesystem from the mirrored LV
2. Execute chlvcopy -b -c <#> -l <newlvname> <original_lvname>
3. Remount the original filesystem
4. Execute mount -o ro /dev/<newlvname> /<backupfs>
5. Backup the backupfs
6. Execute unmount /<backupfs>
7. Execute chlvcopy -B original_lvname

In general, use chlvcopy the same as you would splitlvcopy.

Chapter 6. System Management and Utilities

Web-Based System Manager is an AIX system administration tool for administering an AIX host locally or over the Internet. Web-Based System Manager is the next step in the evolution of AIX system administration tools. This evolution has included System Management Interface tool (SMIT), Motif SMIT, Distributed SMIT, and Visual System Manager (VSM). SMIT and VSM have been major contributors to customer satisfaction regarding AIX system management and usability. It is an objective for Web-Based System Manager to encompass the system administration capabilities and surpass the usability of these predecessors.

The objectives of Web-Based System Manager are:

- Simplification of AIX administration by a single new interface.
- Enable AIX systems to be administered from almost any client platform.
- Enable AIX systems to be administered remotely.
- Provide an administrative environment that exploits the similarities of the Windows 95/NT and AIX CDE desktop environments so that users of the system will find a large degree of look and feel consistency between Web-Based System Manager and these two primary desktop environments.

Web-Based System Manager provides a comprehensive system management environment and covers most of the tasks in the SMIT user interface.

Note: SMIT continues to fulfill the need for system administration from an ASCII terminal.

6.1 Overview of Existing AIX Systems Management

Since the introduction of AIX Version 3.1 there have been a number of system management tools available to help system administrators manage their installations. These include SMIT, DSMIT, and VSM.

6.1.1 SMIT Overview

SMIT was introduced in AIX Version 3.1. It provides a menu-driven interface for approximately 160 local system management tasks. In SMIT, the user is guided by a series of interactive menus and dialogs that automatically build, execute, and log commands required to accomplish selected operations. SMIT eliminates the need to know, or learn, command-level syntax for system management tasks. SMIT is easily extendable, and many LPPs and customers have added their own SMIT menus and dialogs.

Figure 11 shows the default SMIT menu, as seen on an X-based display.

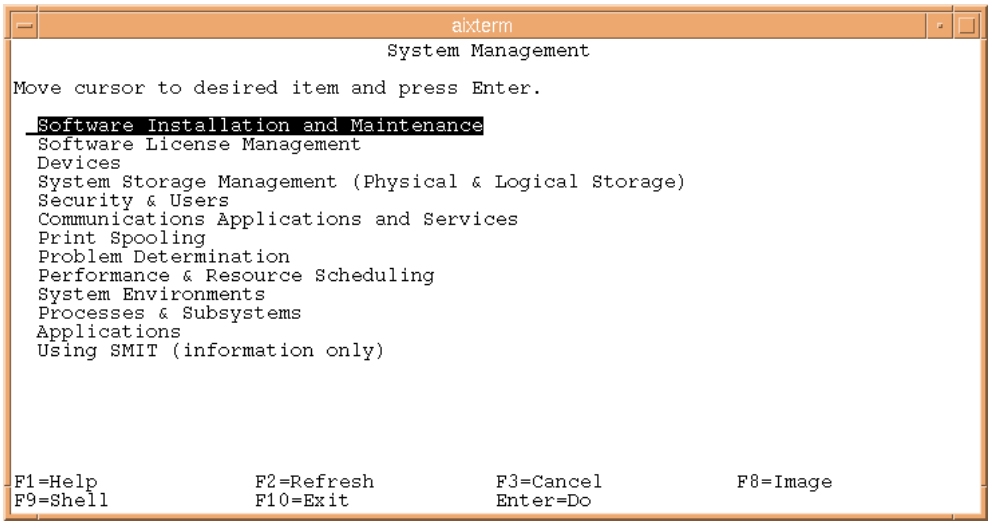


Figure 11. Default SMIT Menu

SMIT provides a character-based interface and a graphical interface. The character-based interface, ASCII SMIT (as shown in Figure 11), can be run on a dumb terminal, while the graphical interface, Motif SMIT (as shown in Figure 12), requires an X Windows compatible graphical display. Motif SMIT provides a point and click interface to the SMIT menus.

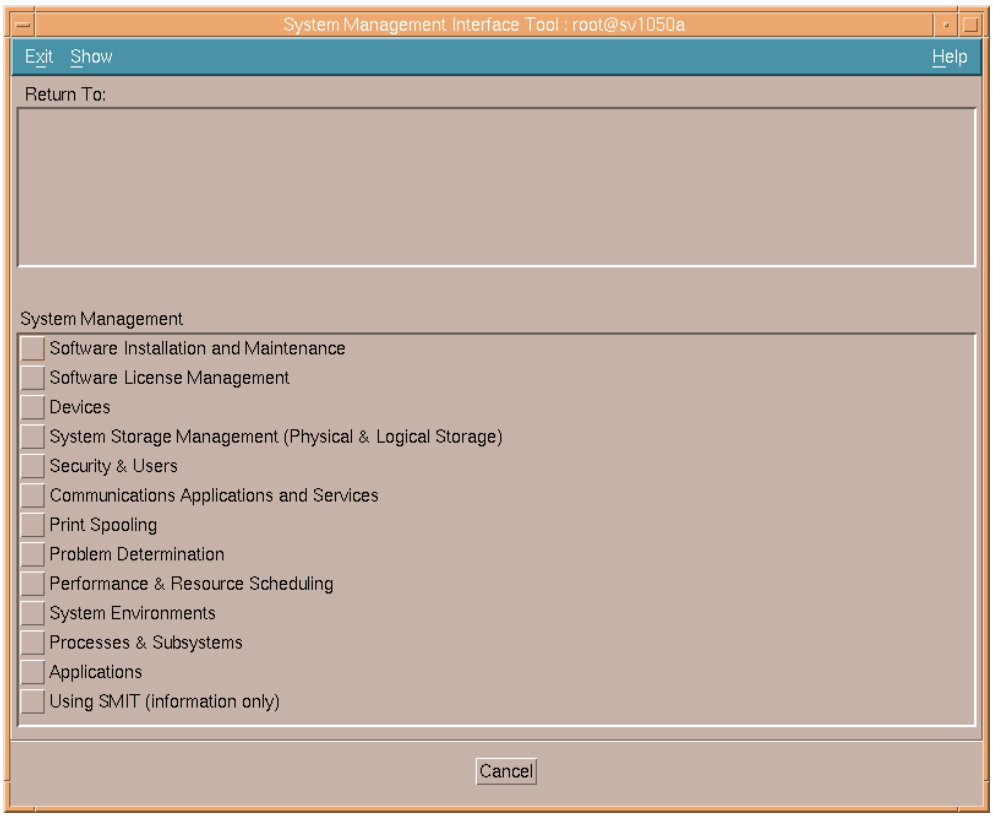


Figure 12. Default Motif SMIT Menu

6.1.2 DSMIT Overview

DSMIT (Distributed SMIT) makes the function of SMIT available in a distributed systems environment. DSMIT allows the administrator to perform SMIT tasks across multiple systems simultaneously from a single point of control. Concurrent and sequential modes of execution are supported. DSMIT provides both the ASCII and graphical user interfaces of SMIT.

DSMIT is a Licensed Program Product (LPP) originally introduced on AIX 3.2.5 and later enhanced and released for AIX V4. In the most recent version, Version 2.2, DSMIT provides an ongoing secure operation including the secure modification of the security configuration and updates of passwords and keys. DSMIT security is based on well established cryptographic routines and DSMIT-specific (modeled after Kerberos 5) communication protocols. DSMIT security features include integrated sign on, authentication, data integrity, data confidentiality, and logging.

Like SMIT, the DSMIT menus and dialogs are easily extendable. Furthermore, DSMIT provides a command line interface that allows the user to run commands, scripts, and programs of their choice on distributed systems. The command line interface allows the user to exploit the capability of DSMIT beyond the provided menus and dialogs without adding additional menus or dialogs. The command line interface also supports interactive commands in the sequential mode of execution. For example, you can run ksh and perform interactive tasks over the secure DSMIT connection.

DSMIT also extends the function of SMIT to heterogeneous systems, with agents available for managing SunOS 4.1.3, Solaris 2.3, and HP-UX 9.0, although the DSMIT agents have not remained current with new releases of Solaris and HP-UX.

6.1.3 VSM Overview

VSM (Visual System Manager) is a graphical user interface that enables you to perform system management tasks through the direct manipulation of objects (icons). Due to VSMs drag and drop graphical interface, you do not need to have a complete understanding of the AIX commands.

VSM was originally introduced as part of AIX 3.2.5 and was enhanced on AIX Version 4. VSM is composed of independent application programs that currently include:

- Device Manager
- Print Manager
- Storage Manager (as shown in Figure 13)
- Users and Groups Manager
- Install and Update Software Manager
- Set Date and Time
- Schedule a Job
- Remove or View Scheduled Jobs
- Maintain Installed Software
- RAID Device Manager

- NIM Install Manager

Figure 13 shows a sample VSM dialog.

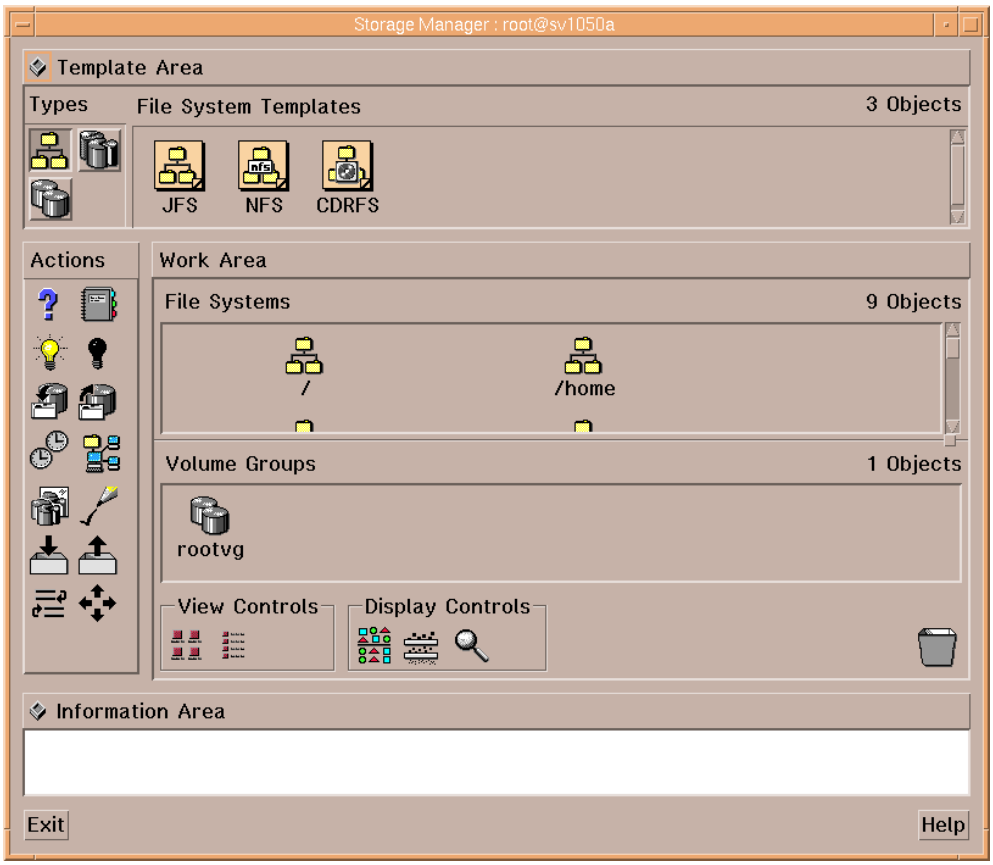


Figure 13. Sample VSM Storage Manager

6.2 Web-Based System Manager Architecture

Web-Based System Manager enables a system administrator to manage an AIX machine either locally from a graphics terminal or remotely from a PC or RS/6000 client. Information is entered through the use of GUI components on the client side. The information is then sent over the network to the Web-Based System Manager server, which runs the commands necessary to perform the required action.

Web-Based System Manager is implemented using the Java programming language. The implementation of Web-Based System Manager in Java provides:

- Cross-platform portability. Any client platform with a Java 1.1-enabled Web browser is able to run a Web-Based System Manager client object.
- Remote administration. A Web-Based System Manager client is able to administer an AIX machine remotely through the Internet.
- A richer and more flexible GUI environment than is available with either HTML forms or Java Script.

Java programs can be delivered as applets that require a Web browser to download the executable code or as stand-alone applications that are stored locally and run independently of a browser or viewer.

Web-Based System Manager has been packaged in a browser-based applet mode, and for local execution, an application mode has been implemented. The application mode uses the AIX Java Virtual-Machine that, in turn, executes the Java applications as threads on the system.

Note: When referring to Java applications, the term *application* is used differently than the conventional use of application as in word processing application or in the discussion on application-oriented user interfaces below. Java application refers to the manner in which Java code is invoked.

6.2.1 Web-Based System Manager Components

Web-Based System Manager includes the following components:

- Backups
- Devices
- File Systems
- Network (interfaces for configuring network communications)
- Printer Queues
- Processes
- Registered Applications
- Software (installable software, software installed, and objects related to installation)
- Subsystems
- System (user interface, console, date/time, language, OS characteristics, system logs, and dump devices)
- Users
- Volumes

6.2.2 Web-Based System Manager User Interface

The Web-Based System Manager user interface is an Object-Oriented User Interface (OOUI). OOUIs are distinguished from traditional, application-oriented user interfaces, in that the user focuses on readily identifiable things on which the user works. In an application-oriented environment, the user focuses on a tool for manipulating the work. Some examples may clarify the distinction. In a document processing context with an application-oriented interface, the user focuses on the tool (a word processing program). While in OOUI, the user focuses on the object of the task itself (the document). In a system management context, an application-oriented interface would require the user to learn management tools (for example, a Print Manager application), while an OOUI would enable the user to directly manipulate a representation of the managed object (for example, a printer or group of printers).

In the evolution of AIX system management user interfaces, SMIT was an application-oriented interface, and VSM was a mixed application/object-oriented

interface. Web-Based System Manager is intended to significantly increase the object-orientation of system management of AIX.

The reasons for this approach, as opposed to an application-oriented user interface are:

- By focusing on objects rather than tools for manipulating objects, OOUIs are more direct and require less learning than application-oriented GUI's.
- OOUIs (especially if implemented using object-oriented programming techniques) have a more consistent user interface than application-oriented interfaces, further reducing the amount of learning required by the user.
- OOUIs follow the current trends in user interface development. User interface styles such as CDE, OS/2, and Windows reflect a trend of increasing object-orientation.

6.2.3 Web-System Manager Launch Interfaces

Web-Based System Manager has been implemented in a modular fashion so that it can be accessed from a variety of launch points. Some launch points are:

- A Web-Based System Manager launch page running inside a Java-enabled browser.
- A Web-Based System Manager application icon in the CDE application manager.

The launch pad icon in the CDE Application Manager loads the Web-Based System Manager environment with launch icons for all of the Web-Based System Manager applications. Multiple applications can be started without the need to restart the Web-Based System Manager environment each time.

The remote launch pad icon in the CDE application manager enables the administrator to login to another AIX 4.3 host and manage it with Web-Based System Manager.

The command line allows Web-Based System Manager to be in X Windows from an `aixterm` window or in the CDE desktop from a `dtterm` window.

When an applet is invoked from a Web-Based System Manager launch interface it appears as a Java frame (essentially a child widow) above the launch interface. Additional dialogs are always opened as child windows. The initial Web-Based System Manager frame appears in Figure 14.

Figure 14. Web-Based System Manager Launch Interface

6.2.4 Web-Based System Manager User Interface Objects

Many system administration and configuration tasks are performed by interacting with simple objects. Simple objects represent individual managed objects that cannot be further decomposed into a collection of objects.

Each instance of a simple object in the system is represented as an icon in a container's view area. Double-clicking on a simple object opens the object so that administrative tasks may be performed. The Web-Based System Manager user interface consists of the following hierarchy of objects:

Container Objects
Container objects include other container objects and simple objects representing elements of system resources to be configured and managed.

Objects
Objects include the following user interface classes:

Property Notebook Objects
Property notebooks (tabbed dialogs) are used for displaying and changing settings associated with a managed resource or container. Property notebooks are useful because they can organize a large collection of system settings into individual pages. They are used in Web-Based System Manager for viewing configuration settings and for

configuration change tasks in which there is no predefined order of steps for the user to perform.

TaskGuide Objects

TaskGuides are dialogs designed to assist the user in performing complex tasks. Unlike property dialogs, TaskGuides lead the user through a task in an ordered series of steps.

6.2.4.1 Container Objects

Simple objects are viewed and manipulated in container objects. Containers consist of a view area, objects within the view, and a group of actions. Actions apply to the container view area and individual objects in the view area.

Web-Based System Manager containers are specialized, that is, each type of Web-Based System Manager simple object (for example, user) is viewed within its own container type (for example, the Users container). The classification of containers provides rules for actions to be applied to included objects and any specialized views of objects within the container type.

Containers are viewed and manipulated in their own primary windows. Container windows perform the functions of CDE and Windows NT file manager folders. All the container windows support the following behaviors:

Open	Iconized containers may be opened to reveal their contents.
Close	Containers may be closed, returning control to the parent container (or in the case of the top-level container, exiting the application).
Find	Find an object by specifying its name.
Reload Now	Rediscover the objects and their current states.
Stop Loading	Halt the loading, or reloading, of a container.
Select All	Select all objects in a container.
Deselect All	Remove selection from all objects in a container.
Scroll	When containers are sized such that their entire contents cannot be shown, they can be scrolled up/down and right/left.

6.2.4.2 Open Action

Opening a container causes the container view area to be populated with objects.

6.2.4.3 TaskGuides

TaskGuides are the IBM/Lotus equivalent of Wizards. Wizards are a Microsoft tool for assisting users in performing complex tasks. They direct the user through the task using questions, instructions, prompts, and controls specific to each step in the task. TaskGuides are used for rarely-performed, and otherwise complex, tasks, such as printer and queue configuration, installing software, and so forth.

Further information on the use and requirements for TaskGuides may be found in the User Assistance section below.

6.2.4.4 Generic Dialogs

Generic dialogs are used to represent objects where notebook or TaskGuide functions are not necessary.

6.2.5 User Interface Elements

The user interface elements of the Web-Based System Manager container are described in the following sections:

Note: The figures are provided as an example to illustrate the user interface elements. The rendering of details of the title bar, menu bar, and so on, vary depending upon the client system on which the applet is running (Figure 15).

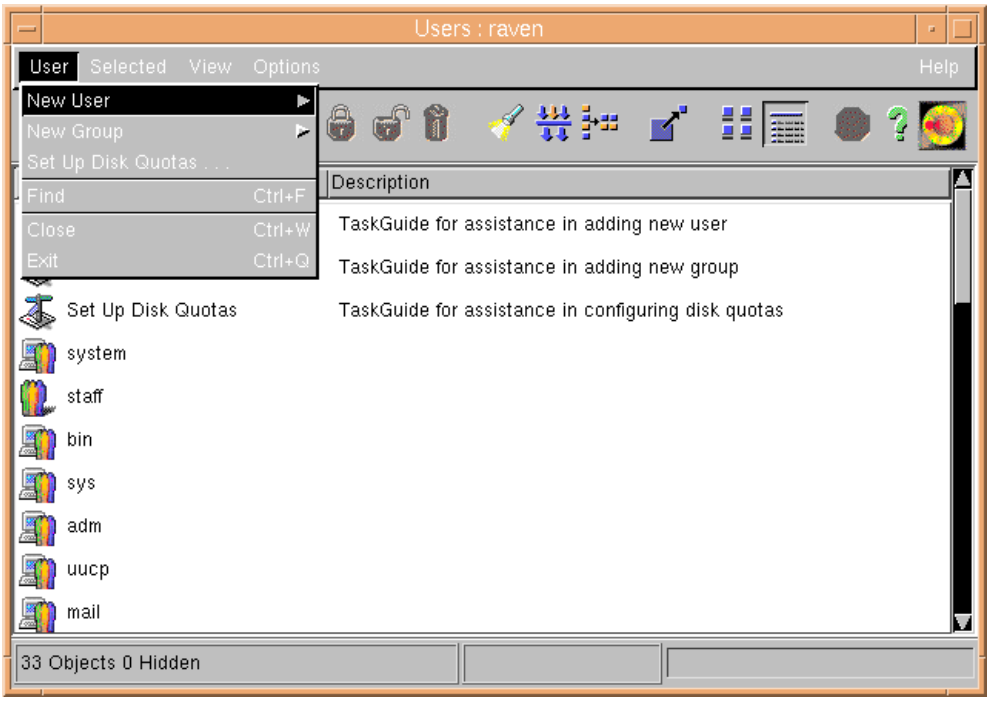


Figure 15. Web-Based System Manager User Menu

The menu bar may be shown or hidden at the user's option. A hidden menu bar may be retrieved through a pop-up menu on the main view area background.

The following menu items are meant to reflect actions that are common across most Web-Based System Managers containers. Additional choices are included for functions that have specific object types.

6.2.5.1 Menu

The object menu contains actions that globally apply to the current container. In each container type, the object menu is titled with the name of the type of object included in the menu. For example, in the users container the object menu is titled User, in the printer's container, the menu is titled Printer, and so on. The basic object menu choices are:

New Create a new instance of the object type contained in current window. This action is equivalent to opening the default template object for the container.

Switch Administrator (applet and remote mode only)
Switch to another user. A dialog box is opened for logging in as another user. Following authentication, the new user's administrative rights are used for actions on the contents of the current container.

Find Opens a search dialog for locating objects. The result of the find action is displayed by providing selection emphasis for visible objects matched in the view area and scrolling the view area to the first object found.

The find dialog also includes a list area for displaying the objects found (because some may not be visible since they are nested in subcontainers) and provide a method of saving the results in a new subcontainer.

Close Close the container window.

Exit Exit Web-Based System Manager. An exit menu choice is present on all Web-Based System Manager containers. When Exit is selected from secondary containers, it generates a confirmation dialog, otherwise, Web-Based System Manager exits.

6.2.5.2 Selected Menu

The selected menu contains actions that are applied against one or more selected objects in the view area. The contents of the Selected menu will differ depending on the type of object container. The selected menu lists only these actions that apply to an object or the set of objects selected. Actions for an object that are temporarily unavailable (for example, *start* when the object has already been started) are dimmed.

Note: The pop-up menu for an object in the view area is roughly equivalent to the selected menu. Additionally, the enabled/disabled menu choices for a pop-up menu on a given object will be equivalent to the enabled/disabled menu choices on the selected menu when the same object has selection emphasis.

Figure 16 shows an example of the Web-Based System Manager Selected menu.

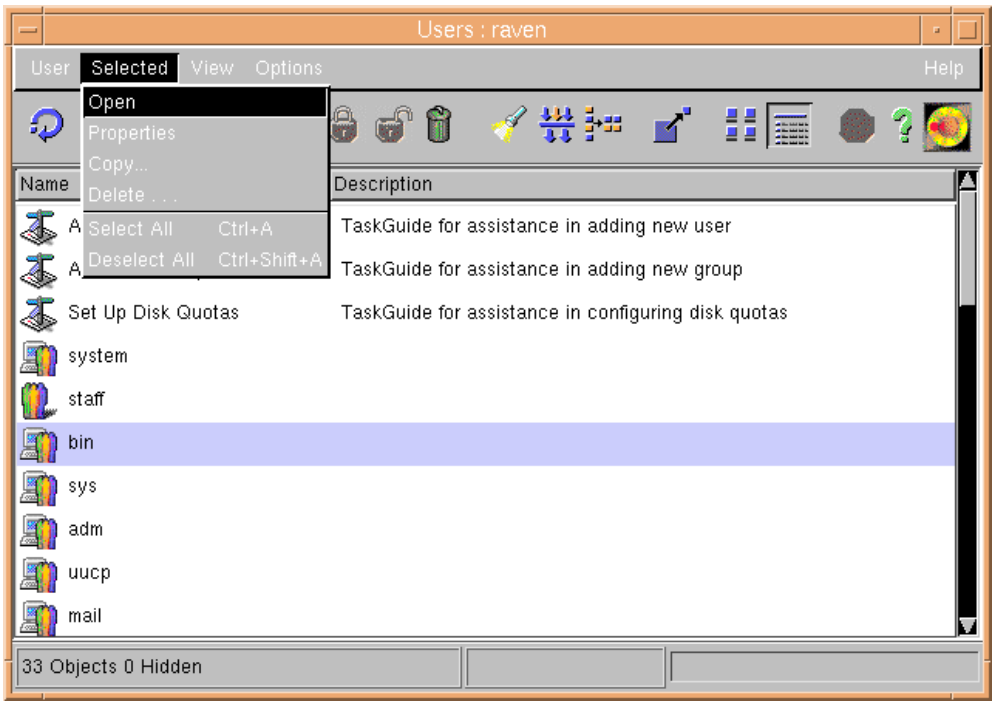


Figure 16. Web-Based System Manager Selected Menu

The basic Selected menu choices are:

Open Opens a selected container or TaskGuide.

Properties Opens a properties notebook dialog for the selected object.

Delete Deletes the selected objects.

Select all Selects all of the objects in the container.

Deselect all Deselects all of the objects in the container.

6.2.5.3 View Menu

The view menu contains options that change the way objects are presented in the view area. The view menu is shown in Figure 17.

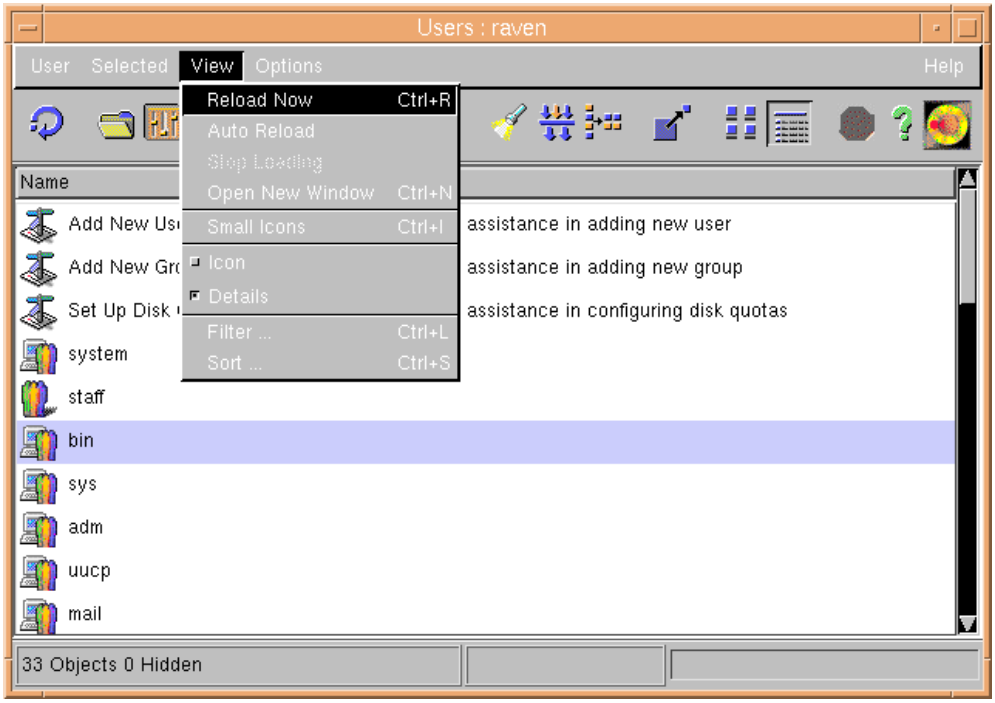


Figure 17. Web-Based System Manager View Menu

The View menu choices are:

Reload Now
Updates the view area with the latest data from the target system. This is analogous to the Netscape *Reload* function.

Stop Loading
Halts an update of the view.

Open New Window
Opens another instance of the current container window. This is equivalent to the Netscape *New Web Browser* action and the CDE file manager *Open New View* action.

Large/Small Icons
Selects either small or large icons for the display.

Icon View
Displays icons in a grid arrangement.

Tree

Changes the presentation of icons to tree view.

Details

Changes the presentation to a tabular view that displays small icons in the first column, object names in the second column, and other relevant properties in one or more additional columns (for example, object description, status, and so on). The exact information displayed in the details view will vary depending upon the application.

Tree Details

Changes the presentation of icons to a tree that also lists properties of each node.

Filter

Opens a dialog box for filtering objects based on user-specified criteria. (Filter is available only in icon and detail views).

Sort

Open a dialog box for sorting the objects based on user-specified criteria. (Filter is available only in icon and detail views).

6.2.5.4 Options Menu

The Options menu contains choices that specify the inclusion, or exclusion, of main user interface elements in the primary window, such as tool bar, status line, and so on. These menu items are selected by a check box for each menu choice, as shown in Figure 18.

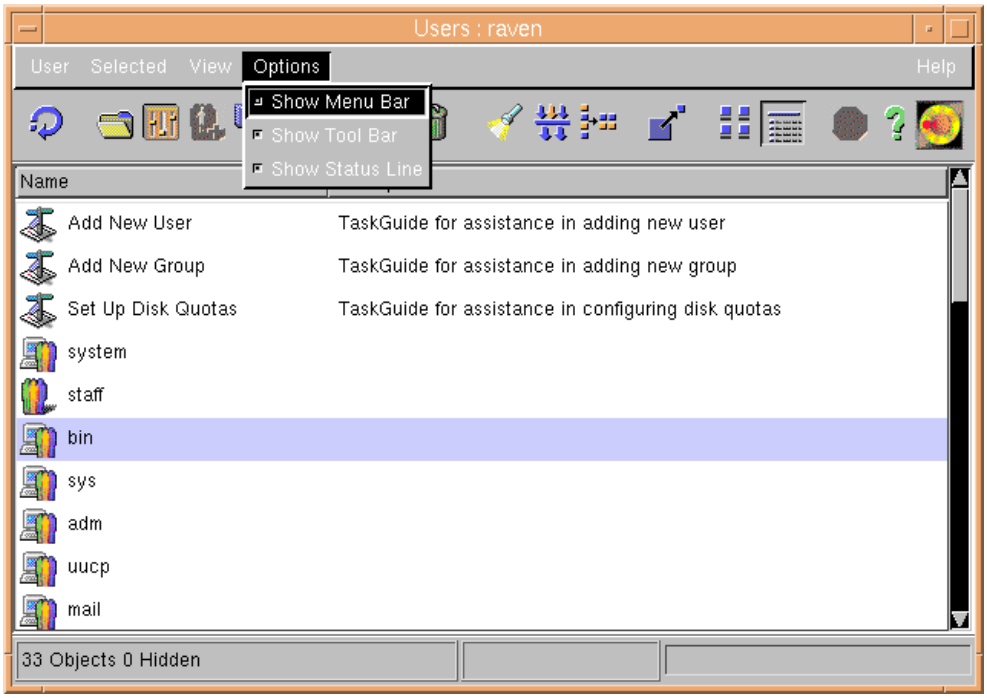


Figure 18. Web-Based System Manager Options Menu

The Options menu choices are:

Show Menu Bar (check box)

Default position is checked. Unchecking removes the menu bar from

the window. The menu bar is restored through a pop-up menu choice on the view area background.

Show Tool Bar (check box)
Default position is checked. Unchecking removes the tool bar from the window and subsequent subcontainers windows.

Show Status Line (check box)
Default position is checked. Unchecking removes the status line from the container and subsequent containers.

6.2.5.5 Help Menu

See Section 6.2.7.2, “Container Help Menu Contents” on page 110.

6.2.5.6 Pop-Up (Context) Menus

Pop-up menus are available for each object in a view area. When the cursor is positioned over an object, the Selected menu for that object is presented in a pop-up menu. When the cursor is over the main view area background, the pop-up menu contains the Options menu contents.

When a group of dissimilar objects is selected, the pop-up menu for the collection reflects only the actions that are applicable to all of the collection.

6.2.5.7 Dynamic Status Icon

A dynamic status icon is used to indicate the status of:

- Communications
- Processing on the target system

6.2.5.8 Tool Bar

Frequently used actions are represented on a tool bar. The tool bar can be displayed, or hidden, at the user's option.

The contents of the tool bar consists of some icons common to all containers (for example, Reload Now, Stop Loading, View Type) and other icons unique to specific container types. For example, the Users container includes a *Change Password* icon.

6.2.5.9 Main View Area

The main view area displays the objects and containers within the current container.

6.2.5.10 Command Buttons

The following command buttons are included on the background panel of the dialog. They are:

OK Applies all of the parameters specified on each of the tab pages visited and closes the dialog box.

Apply Applies all of the parameters specified on each of the tab pages visited and leaves the dialog box open.

Cancel Closes the dialog box without applying any parameters.

Help Launches a help window for the tab page currently visible.

6.2.5.11 Status Line

The status line is used to display current status information about the container. The status line may be shown, or hidden, at the user's option. Examples of information displayed in the status line are: number of objects shown in the view area, number of objects hidden, and loading status.

6.2.5.12 Container Views

Container views present a variety of representations of a group of objects that can be altered according to user needs or preferences. Many containers are able to present more than one view. Because different view types may be more or less appropriate for different object types, there is no one default view for all object containers.

Examples of standard views are listed in the following:

Icon Icon view arranges icons for managed objects in a grid. This view is useful for displaying a large number of objects in a small area (Figure 19).

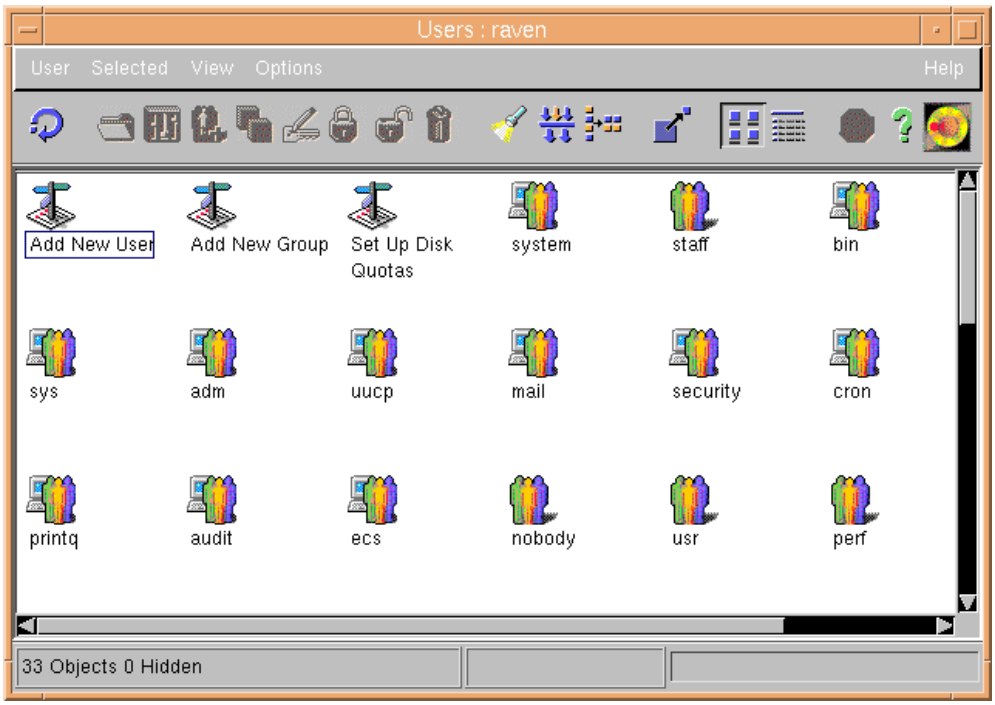


Figure 19. Web-Based System Manager Icon View

Details The icons in the view are displayed in a grid or table with the object icons, or names, in the columns on the far-left and additional property information in the remaining columns. See Figure 20 for an example of the details view for system devices.

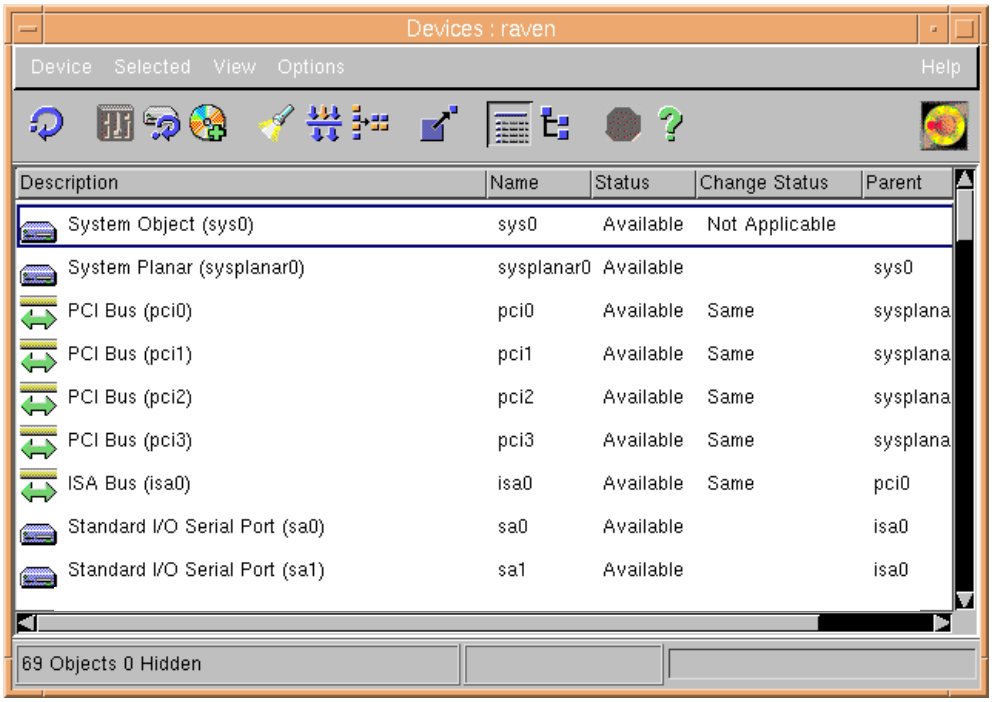


Figure 20. Web-Based System Manager Details View

Tree A tree view displays a hierarchical relationship with parent and child nodes. The tree view is useful for displaying users and groups, printers and queues, bundles and file sets, and devices. See Figure 21 for an example of a tree view for system devices.

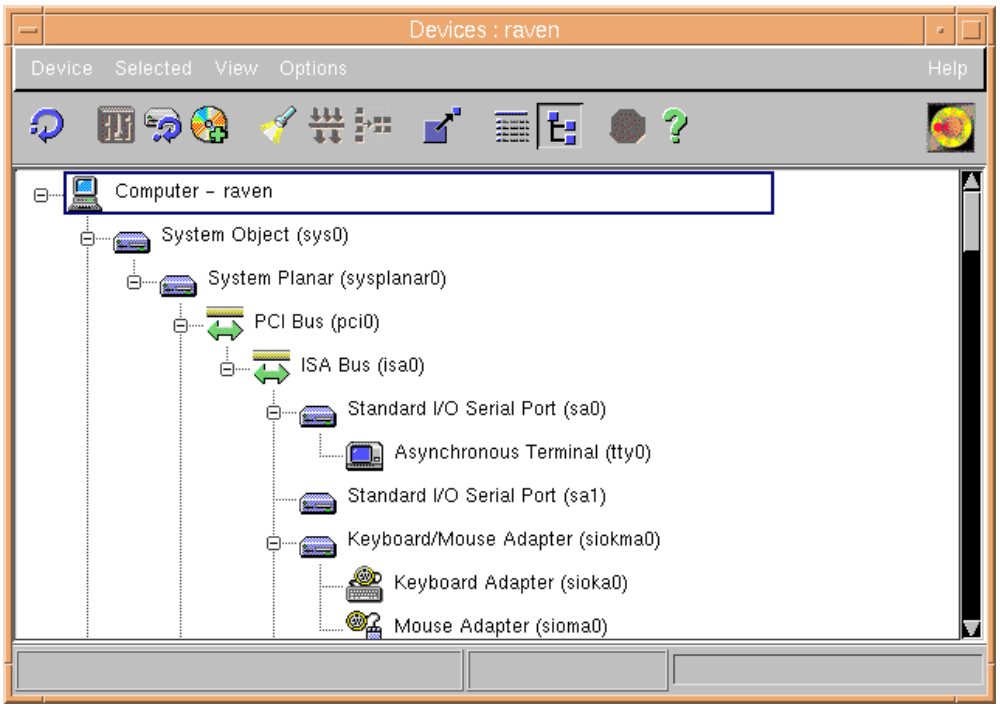


Figure 21. Web-Based System Manager Tree View

Note: The Web-Based System Manager UI architecture includes single-rooted and multiple-rooted tree views and trees of an arbitrary number of levels.

6.2.6 Message Boxes

Where it is necessary to alert the user to various conditions, a message box is used for the following purposes:

- Informational messages
- Warning messages
- Critical conditions
- Confirmation prompts

6.2.7 User Assistance

User assistance includes on-line information designed to aid users in performing unfamiliar tasks. Web-Based System Manager user assistance includes: on-line books, help, TaskGuides, hover help, and context help. These are described below.

6.2.7.1 Help

General help for each Web-Based System Manager application is provided through a container-level help for that application. The general help also contains an overview section common to all Web-Based System Manager applications.

All help text is written in HTML and is accessed through the Web browser on the client system.

6.2.7.2 Container Help Menu Contents

Container help menu choices are provided for the following:

- Contents (contents of extended or reference help)
- What's this? (places the application in context-sensitive help mode)
- Search for help on topic (Web page for accessing the help search engine)
- How to use Help
- About Web-Based System Manager (product information)

6.2.7.3 Context Sensitive Helps

Context-sensitive help is provided in a child window that pops up above the user interface element for which help is requested. It is available when the user:

- Selects **What's this?** from the application help menu. This enters context-sensitive help mode.
- Clicks on a tool bar icon containing a question mark to get help on the contents of a window.
- Presses the **help** button on a dialog.

Context-sensitive help is available for:

- Fields with a dialog
- Objects in a view area

6.2.7.4 Hover Help

Web-Based System Manager displays hover help when the user pauses the mouse pointer over a toolbar icon.

6.2.7.5 Online Books

Web-Based System Manager users have access to the complete AIX on-line documentation library through hypertext links in extended help.

6.2.8 Navigation

Although the usual method of navigation through Web-Based System Manager is by use of a mouse, it also possible through the keyboard.

6.2.8.1 Keyboard Navigation

To meet the needs of a wide range of users of different abilities and skills, Web-Based System Manager supports keyboard navigation. Specific key assignments for keyboard navigation are similar to Windows and Netscape Navigator.

The following keyboard navigation methods are supported:

Focus Traversal
: Tab and Shift-Tab are used to move forward and backward among controls.

Menu Shortcuts
: Short cuts (or accelerators) are keyboard equivalents for menu commands that are executed by key combinations (such as CTRL-F for Find).

6.2.8.2 Mouse Model

For a three-button mouse, the mouse button functions are:

- Button 1 - select, drag, activate.
- Button 3 - context (pop-up) menu

For a two button mouse, the mouse button functions are:

- Button 1 - select, drag, activate
- Button 2 - context menu

A single-click of button 1 is used for selecting icons and activating button controls.

A double-click of button 1 activates view area icons with their default behaviors. For container objects, the default behavior is to open-contents-view. For simple objects (that is, without contents), the default behavior is open-properties-dialog. For objects that have both properties and contents (for example, UNIX groups), the double-click action is to open-contents-view. Pop-up and Selected menu choices (*Open*=contents, *Properties*=properties dialog) for each action are provided.

6.2.9 Selection and Multiple Selection

A single selection defines to which object the actions in the selected menu, or pop-up menu, apply. Specific actions are enabled, or disabled, depending upon the type of object selected. A single-click of button 1 is used to select an object.

A multiple selection is enabled for various types of objects and actions. Specific actions in the selected and pop-up menu will be enabled or disabled depending upon whether or not multiple selection is allowed for the collection of objects selected. The menu choices enabled are the intersection of the enabled states of the objects in the selected collection.

Most standard multiple selection interaction techniques are supported, including range selection, use of Ctrl-Select to modify a selection, and use of shift-select to select a contiguous range of objects.

6.3 Web-Based System Manager Enhancements (4.3.1)

Web-Based System Manager was introduced with AIX version 4.3.0 as a technology evaluation release. It did not provide all of the function required for system management but was intended to demonstrate to customers the direction of system management products.

The version of Web-Based System Manager shipped with AIX version 4.3.1 contained significant performance enhancements over the previous release. The improvements were mostly due to improvements in the underlying Java run time system.

6.4 Web-Based System Manager Enhancements (4.3.2)

The following sections describe the enhancements to Web-Based System Manager that were introduced by AIX version 4.3.2.

6.4.1 Security Enhancements

AIX 4.3.2 has enhanced the function of Web-Based System Manger to allow remote administration sessions to be carried out using the Secure Socket Layer (SSL) protocol. This allows all data transmitted on the network between the Web-Based System Manager client and the machine being managed to be encrypted and, therefore, prevent unauthorized systems from viewing the data.

The software required to implement this function is included on the AIX Bonus Pack that ships with AIX 4.3.2. The required package is sysmgt.websm.security. Users in the United States and Canada can additionally install the package sysmgt.websm.security-us, which provides stronger encryption facilities.

The configuration process for using the SSL protocol involves generating a public and private key for each machine to be managed. The certificates can be obtained from an external Certificate Authority or generated on a designated Web-Based System Manager server for use in a private network. The keys are installed on the machine being managed and, additionally, on any AIX machines that will use the Web-Based System Manager client to manage the machine in client-server mode. In this scenario, all communication between the client and server takes place using the SSL protocol.

In applet mode, where the Web-Based System Manager client is run in a browser, the client is required to download the Web-Based System Manager servers public key in order to verify the applet files that are being downloaded. For maximum security, the client should connect to the server using the HTTPS protocol.

The encryption facilities provided work in conjunction with a Web server that uses SSL to support the HTTPS protocol. The Lotus Domino Go Web server, that is also provided on the Bonus Pack can be configured to accept requests using the HTTPS protocol, either in addition to, or instead of, the HTTP protocol. Similarly, the Web-Based System Manager server running on the managed machine can be configured to respond to HTTPS requests either in addition to, or instead, of HTTP requests.

A client session with a server that has been configured with the optional security is shown in Figure 22.

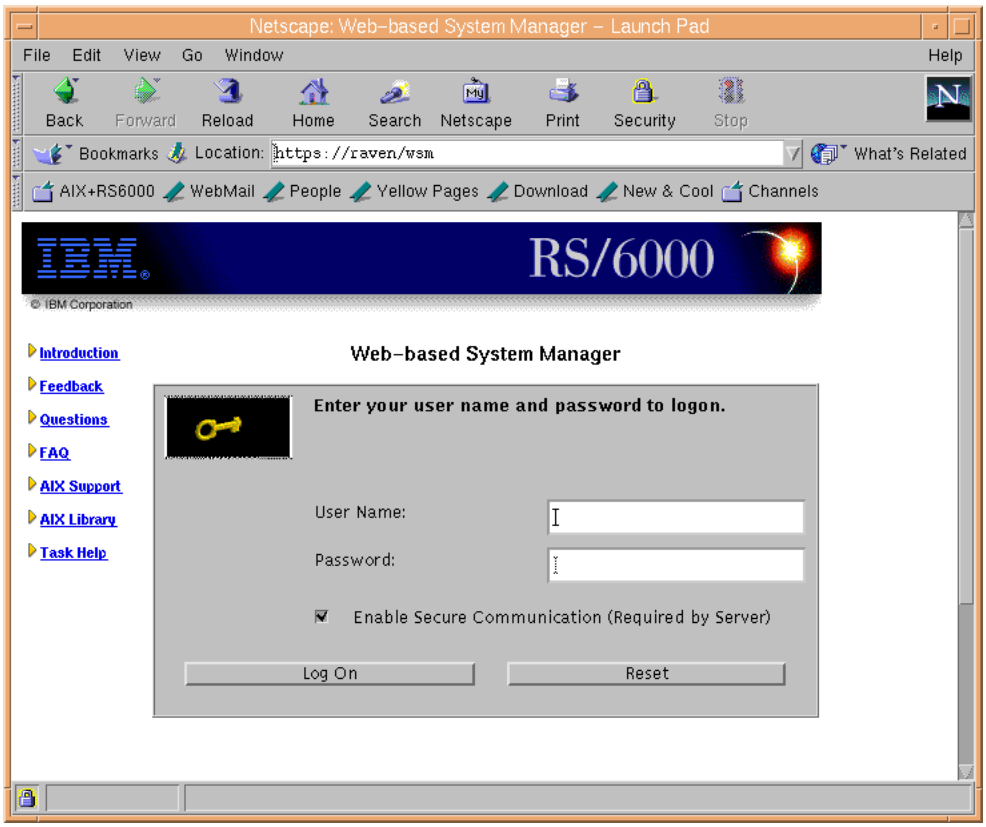


Figure 22. Example of Secure Mode Connection Using HTTPS

The only visible differences when using the secure version are:

- The URL of the initial Web-Based System Manager login page specifies the HTTPS protocol.
- The browser indicates that the Web page being viewed has been obtained using a secure connection. The Netscape Navigator browser shipped on the Bonus Pack indicates this with a locked padlock icon in the lower left corner of the window.
- Web-Based System Manger child windows have the message Secure Connection displayed in the status bar at the base of the window. See Figure 23.

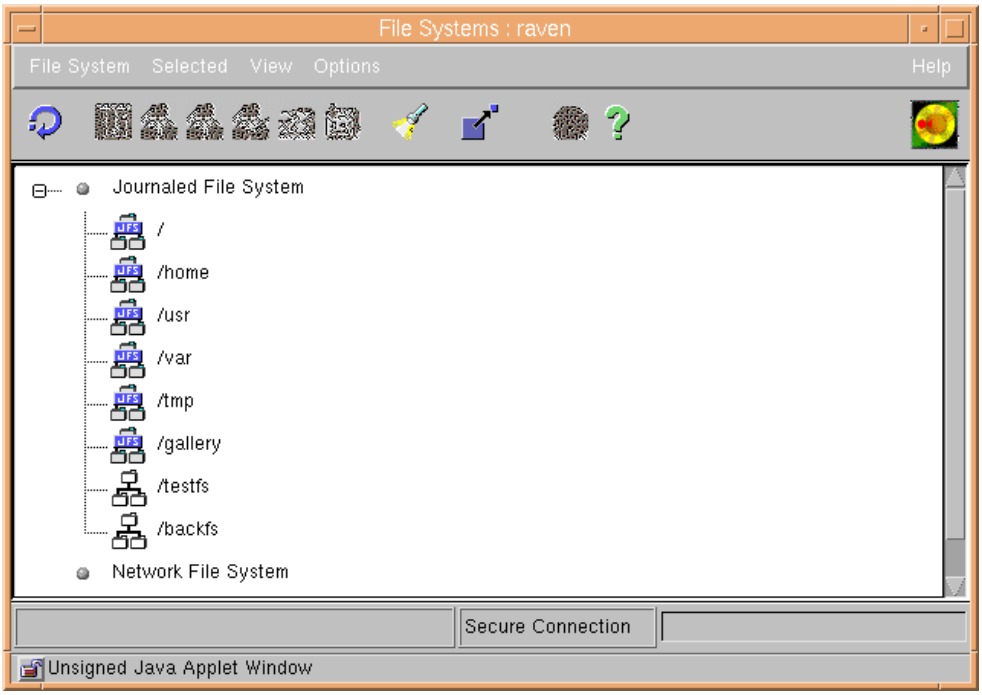


Figure 23. Example of Container Window in Secure Mode

6.4.2 Diagnostics Enhancements

Web-Based System Manager has been enhanced to allow diagnostic and service aid functions to be carried out on devices that support these actions. The menu presented depends on the capabilities of the selected device. For example, it is possible to perform format or certify operations on certain models of disk drives, as shown in Figure 24.

It is also possible to perform Error Log Analysis when running diagnostics on the selected device. The AIX error log can be searched for errors logged against the selected device for errors between 1 and 60 days old.

If a device is marked with a warning triangle (containing an explanation point), the Run Diagnostics selection can be used to determine what is wrong with the device, or if it has just been removed from the system.

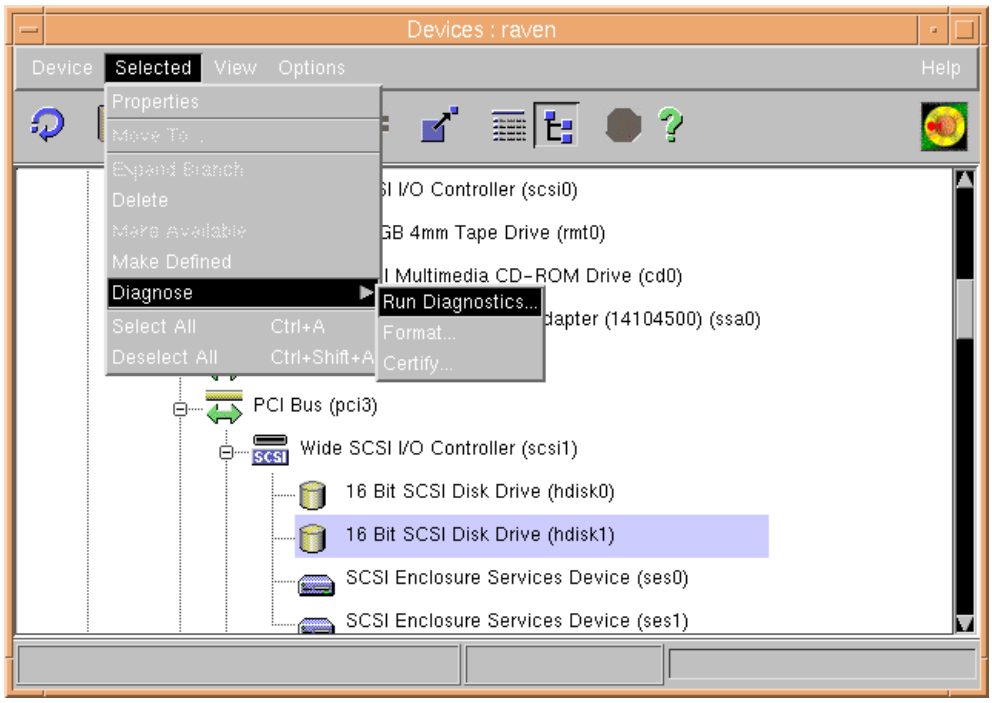


Figure 24. Example of Diagnostics Menu

Using the previous selections, the menu shown in Figure 25 is presented. Here, you may select how the diagnostics are run.

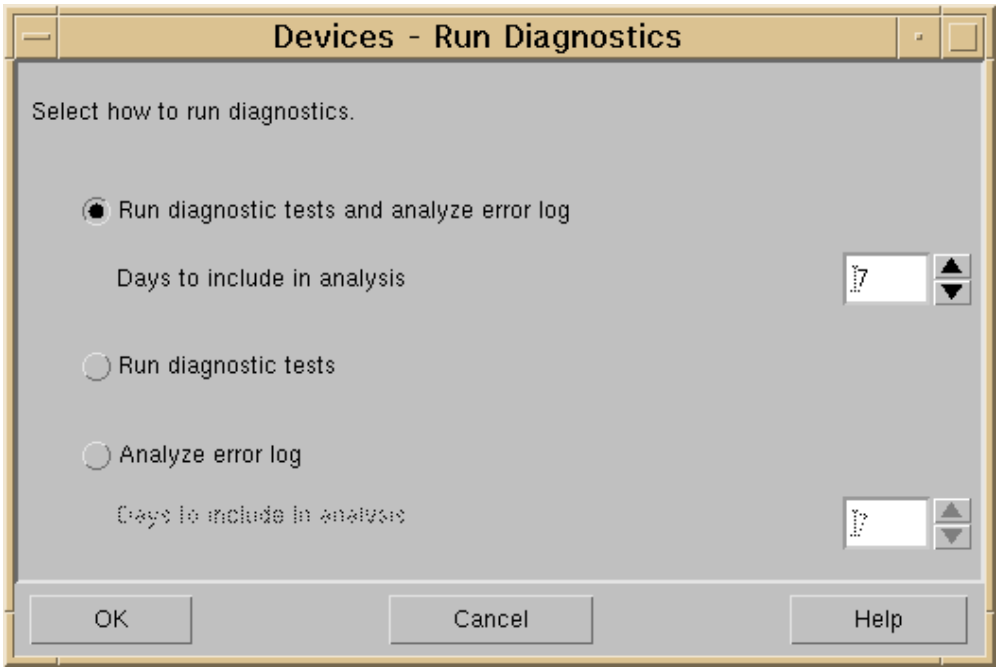


Figure 25. Example of ELA Day Selection Menu

6.4.3 Registered Applications

This function allows a system administrator to register remote applications with Web-Based System Manager. It only supports an application that can be accessed using a URL. For example, it allows a system administrator to register the Netfinity Manager application with Web-Based System Manager. The dialog screen, shown in Figure 26, prompts the user to enter the URL to start the application and shows a list of machines that have the application installed.

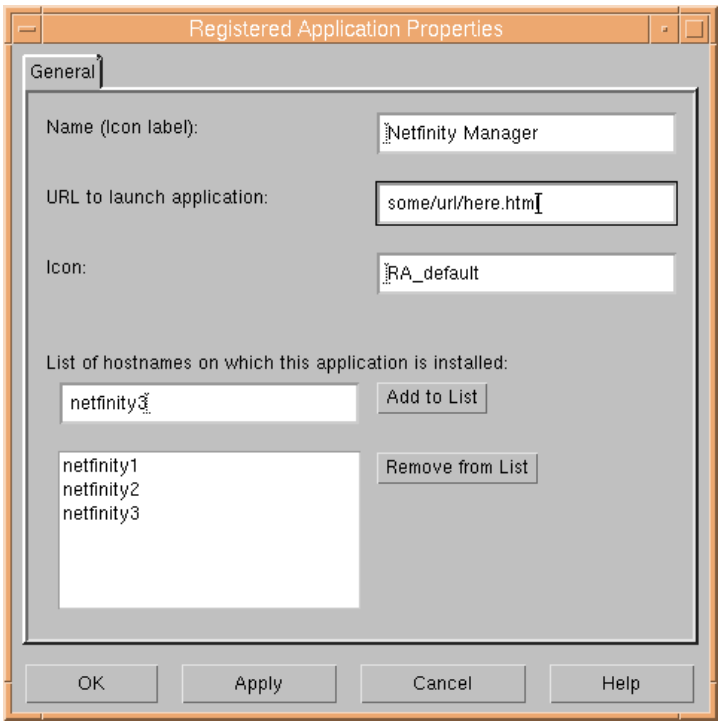


Figure 26. Registered Applications Dialog Box

The registered application then appears on the Registered Applications container as an icon, shown in Figure 27.

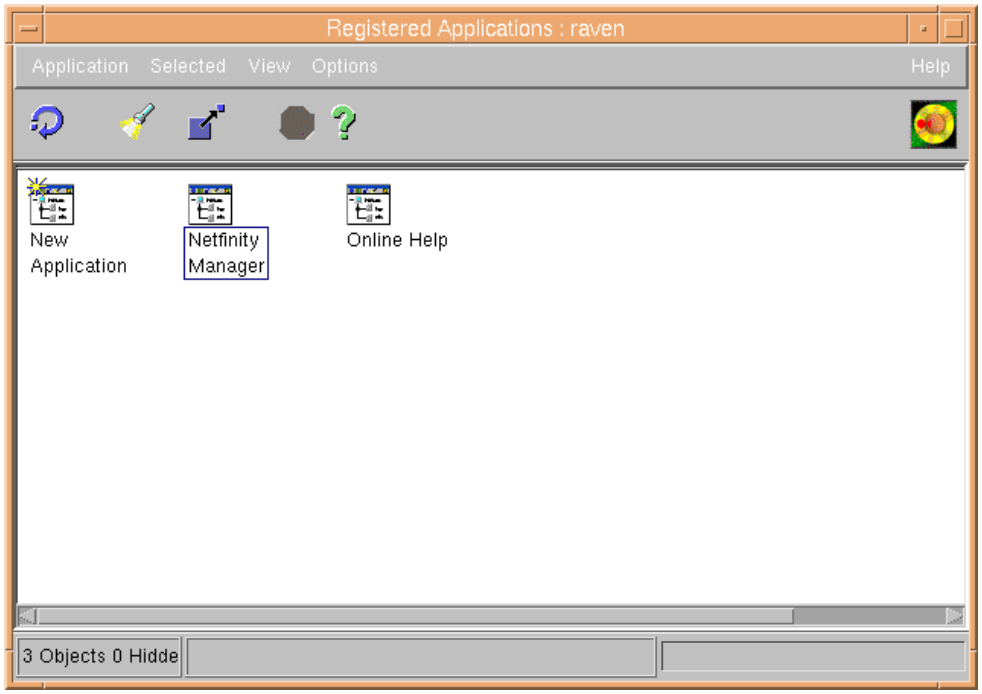


Figure 27. Registered Applications Container

When the icon is opened, the user is prompted to select the required target machine if the application is registered with multiple machines. This is shown in Figure 28. Web-Based System Manager then starts the Netscape Web browser with an initial URL of the registered application on the target machine.

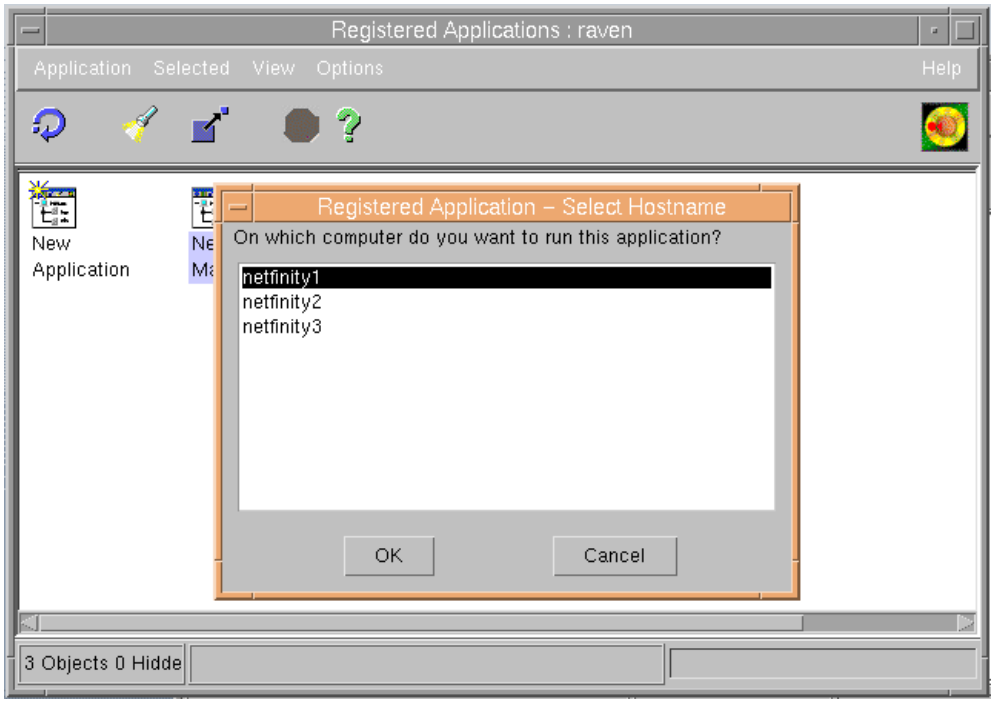


Figure 28. Registered Application Host Selection Dialog

6.5 Daylight Savings Time

Before AIX Version 4.3, daylight savings time could be selected through SMIT, but the date of change for this characteristic was restricted to USA standard. Once the daylight savings YES or NO question had been answered, a list of available time zones was presented, but there was no option within SMIT for specifying the start and end dates for daylight savings.

By default, the daylight saving time starts on the first Sunday of April at 2:00am and stops on the last Sunday of October at 2:00am. AIX Version 4.3 now has the capability of overriding these settings by specifying the start time and stop time in the TZ environment variable.

Additional fields have been added to the SMIT time setting menu to set the TZ variable. A back-end script performs the actual setting of the TZ variable.

6.6 Login Performance

The original design of the UNIX/AIX login system dates back to the early days of UNIX when the number of users to be catered for was relatively small. As such, the login process was perfectly adequate. With the commercial acceptance of UNIX, however, the number of users per system has grown dramatically with tens of thousands of users now being seen on some servers. This increase in the number of users has highlighted some of the deficiencies in the original design of the login process that are now beginning to affect system performance.

A problem exists, which once a user has entered their name and password, the system must then search through the /etc/passwd and /etc/security/passwd files trying to find a match for that user and, if successful, must also update a number of other files. All the files are searched sequentially, and the time consumed can be substantial if there are a large number of records to search through. In extreme cases, if a user's entry is near the end of the files, it is possible for the login attempt to time out before completion. Also, the amount of CPU time being consumed by the login process is a cause for concern. Login CPU usage has been recorded as high as 47 percent on some systems.

The three major bottlenecks that have been identified are:

- Reading the /etc/passwd file
- Reading the /etc/security/passwd file
- Updating the /etc/security/lastlog file

A limited solution was used in previous versions of AIX to address some of these issues by creating a hashed version of the /etc/passwd file. The `mkpasswd` command took the /etc/passwd file as input and created the /etc/passwd.dir and /etc/passwd.pag files. Then during login, if these hashed files existed, they were read instead of the ASCII file. This partial solution provided some relief, but there were still other areas that could also be improved.

A further improvement has been introduced in AIX Version 4.3 that provides indexed access to the data files instead of sequential reads. Indexes are created using the user name and user ID as keys with offsets into the data file. The data

files remain as ASCII files, but the design allows for them to be upgraded to binary files at a later date if this is found to be necessary.

The /etc/passwd and /etc/security/passwd files have been indexed, and the /etc/security/lastlog file has been indexed with a corresponding change in the way that this file is processed.

6.6.1 Indexing of the /etc/passwd File

The actual /etc/passwd file has not been changed. However, two new files have been created, /etc/passwd.nm.idx and /etc/passwd.id.idx, through the `mkpasswd` command. These files are indexes created using the username (string) and the user ID (number) as keys. A record in the index file contains the key, offset of the corresponding record in the data file (/etc/passwd), and the status of the record. A negative offset value implies the corresponding record is deleted.

A hook was also added at the point where the file is read to check for the existence of the index file. If the corresponding index exists, then the index read mechanism is called with the key as a parameter.

6.6.2 Indexing of the /etc/security/passwd File

The /etc/security/passwd file is a text file that contains one stanza for each user. It is searched one line at a time looking for the user name. Similar to the /etc/passwd file, the user name that is physically located at the top of the file is at an advantage compared to the user name at the bottom of the file. This file is also indexed by the `mkpasswd` command. The index is based on the username string as the key and provides an offset into the file where the stanza can be located. Once the stanza is located, it is then searched sequentially for the relevant information.

6.6.3 Indexing and Locking /etc/security/lastlog File

The lastlog file is a text stanza file similar to the /etc/security/passwd file that contains one stanza per user. It is accessed in a similar manner, sequentially looking for a username. However, unlike the /etc/passwd and /etc/security/passwd files, this file is accessed for update, which means that file-locking and crash-recovery must be taken into account. In the existing design, locking is done at a file-level, and the whole file is backed up, representing a major overhead. Note that there are no external commands or system and library calls to access this file. It is accessed internally by the tsmlogin() module to display lastlog information.

In AIX Version 4.3, the /etc/security/lastlog file remains a text file but has been changed in the following ways:

Indexed access

An index called /etc/security/lastlognm.id is built using the username string. This index provides the offset to a stanza. Once a stanza is located, the lines in it are processed sequentially as in AIX Version 4.2. The index is created by the `mkpasswd` command.

Fixed record length

Since a user stanza is updated upon login, if the file needed to be reorganized after every update (because of the variable length text fields), this would cause the record (or stanza) offsets for all stanzas after the changed stanza to be changed by a fixed delta. This would

keep the data and index files synchronized but would be expensive in processing time. The extra processing was avoided by keeping some unused space in each variable length field. By padding the fields with spaces, they can shrink or grow within limits, and a record can be updated without having to rearrange the file and rebuild the index.

Record level lock

Locking is now done at the record-level instead of the file-level by using advisory locks for updates. This prevents having to read and write the entire file. There is no longer a need for the /etc/olastlog file, and it has therefore been removed.

6.6.4 mkpasswd Command

The index system is created when the `mkpasswd` command is executed. It deletes any existing outdated indexes (except for the lastlog index) and builds new indexes. New flags allow complete control of this enhancement.

6.7 Microcode Packaging

In versions of AIX prior to Version 4.3, some IBM microcode entities resided in filesets that were prerequisites of other filesets. This meant that our OEM customers had to ship these filesets even if they were not required. In some cases, customers who developed their own additional features preferred to use their own microcode instead of the IBM-supplied microcode. In order for these OEM customers to replace the IBM microcode, it was necessary for them to modify AIX.

To rectify this situation, the following filesets were modified to remove the IBM microcode so that our OEM customers can ship AIX without having to perform any changes:

- bos.sysmgt.nim.master
- devices.mca.8fc8
  - Common token-ring Software
  - Token-Ring high-performance adapter diagnostics
  - Token-Ring high-performance adapter microcode
  - Token-Ring high-performance adapter software
- devices.mca.df9f
  - Direct-Attached disk diagnostics
  - Direct-Attached disk software
- devices.mca.ffe1
  - 128-port asynchronous adapter diagnostics
  - 128-port asynchronous adapter microcode
  - 128-port asynchronous adapter software

The devices.mca packages were a source of error because even though some systems do not have an MCA bus, other rspc packages regarded these MCA packages as prerequisites. The files needed by the other rspc packages were moved out of the devices.mca.* packages and into a separate fileset.

6.8 On-line Alternate Disk Installation

The `alt_disk_install` command gives users another way to update AIX to the next release or maintenance level without having to schedule an extended period of system downtime.

The update can be performed in two ways:

mksysb image

Installing a mksysb requires a 4.3 mksysb image or 4.3 mksysb tape. The `alt_disk_install` command is called, specifying a disk or disks that are installed in the system but are not currently in use. The mksysb is restored to those disks, such that, if the user chooses, the next reboot will boot the system on a 4.3 system.

Note: If needed, the `bootlist` command can be run after the new disk has been booted, and the bootlist can be changed to boot back to the older version of AIX.

Cloning

Cloning allows the user to create a backup copy of the root volume group. Once created, the copy may be used either as a back up, or it can be modified by installing additional updates. One possible use might be to clone a running production system, then install updates to bring the cloned rootvg to a later maintenance level. This would update the cloned rootvg while the system was still in production. Rebooting from the new rootvg would then bring the level of the running system up to the newly installed maintenance level. If there was a problem with this level, simply changing the bootlist back to the original disk and rebooting would bring the system back to the old level.

Currently, you can run the `alt_disk_install` command on 4.1.4 and higher systems for both of these functions. The bos.alt_disk_install.rte fileset must be installed on the system to do cloning to an alternate disk, and the bos.alt_disk_install.boot_images fileset must be installed to allow a mksysb install to an alternate disk.

The mksysb image that is used must be created before installation and include all the necessary device and kernel support required for the system on which it is installed. No new device, or kernel support, can be installed before the system is rebooted from the newly-installed disk.

Note: The level of mksysb that you are installing must match the level of the bos.alt_disk_install.boot_images fileset. At this time, 4.3.2, 4.3.1, and 4.3.0 mksysb images are supported. AIX 4.3.1 boot images are available only on the 4.3.1 installation media.

When cloning the rootvg volume group, a new boot image is created with the `bosboot` command. When installing a mksysb image, a boot image for the level of mksysb and platform type is copied to the boot logical volume for the new alternate rootvg. When the system is rebooted, the `bosboot` command is run in the early stage of boot, and the system will be rebooted again. This is to synchronize the boot image with the mksysb that was just restored. The system will then boot in normal mode.

At the end of the install a volume group, altinst_rootvg, is left on the target disks in the varied off state as a place holder. If varied on, it will show as owning no logical volumes, but it does, in fact, contain logical volumes. Their definitions have been removed from the ODM because their names now conflict with the names of the logical volumes on the running system. It is recommended that you do not vary on the altinst_rootvg volume group but just leave the definition there as a place holder.

When the system reboots from the new disk, the former rootvg will not show up in an `lspv` listing. The disks that were occupied by the rootvg will show up as not having a volume group. However, you can still use the `bootlist` command to change the bootlist to reboot from the old rootvg if necessary.

When the system is rebooted from the new altinst_rootvg, then `lspv` will show the *old* rootvg as old_rootvg so you will know which disk or disks your previous rootvg was on. There is also a -q option in alt_disk_install that will allow you to query to see which disk has the boot logical volume so you can set your bootlist correctly for cases when old_rootvg has more than one disk.

The alternate root file system is mounted as /alt_inst, so other file systems also have that prefix (/alt_inst/usr, /alt_inst/var). This is how they must be accessed if using a customization script.

Note: If you have created an alternate rootvg with `alt_disk_install`, but no longer want use it, or you want to run `alt_disk_install` commands again, do not run `exportvg` on altinst_rootvg. Simply run the `alt_disk_install -X` command to remove the altinst_rootvg definition from the ODM database.

The reason you cannot run the `exportvg` command (or the `reducevg` command) is that the logical volume names and file systems now have the real names, and exportvg removes the stanzas for the real file system from /etc/filesystems for the real rootvg.

If `exportvg` is run by accident, be sure to recreate the /etc/filesystems file before rebooting the system. The system will not reboot without a correct /etc/filesystems file.

6.8.1 alt_disk_install Command Syntax

The following is an example of the `alt_disk_install` command:

```
alt_disk_install -d device || -C [-i image.data ] [-s script ] [-R resolv_conf
]
[-D] [-B] [-V] [-r] [-p platform ] [-L mksysb_level ]
[-b bundle_name [ [ -I installp_flags ] [-l images_location ] [-f fix_bundle ]
[-F fixes ] [-e exclude_list ] [-w filesets ] target_disks...

alt_disk_install -X
```

The following is a description of `alt_disk_install` flags:

-d device

The value for device can be:

- Tape device - for example, /dev/rmt0.

- Path name of mksysb image in a file system.

Note: -d and -C are mutually exclusive.

-C Clone rootvg.

Note: -d and -C are mutually exclusive.

-i image.data
Optional image.data file to use instead of default image.data from mksysb image or image.data created from rootvg. The image.date file name must be a full path name. For example, /tmp/my_image.data.

-s script Optional customization script to run at the end of the mksysb install or the rootvg clone. This file must be executable. This script is called on the running system before the /alt_inst file systems are unmounted, so files can be copied from the running system to the /alt_inst file systems before the reboot. This is the only opportunity to copy or modify files in the alternate file system because the logical volume names will be changed to match rootvg's, and they will not be accessible until the system is rebooted with the new alternate rootvg. You must use a full path name for script.

-R resolv_conf
Specifies the path name of the resolv.conf file you want to replace the existing one after the mksysb has been restored, or the rootvg has been cloned. You must use a full path name for resolv_conf.

-D Turn on debug (set -x output).

-V Turn on verbose output. This will show the files that are being backed up for rootvg clones. This flag will show files that are restored for mksysb alt_disk_installs.

-B Specifies not running bootlist after the mksysb or clone. If set, then the `-r` flag cannot be used.

-r Specifies to reboot from the new disk when the alt_disk_install command is complete.

-p platform
This is the platform to use to create the name of the disk boot image. The default value is the platform of the running system obtained with the `bootinfo -T` command on 4.1 or the `bootinfo -p` command in 4.2. This flag is only valid for mksysb installs (-d flag).

-L mksysb_level
This level is combined with the platform type to create the name of the boot image to use (IE rspc_4.3.0_boot). This must be in the form V.R.M. The default is 4.3.0. The mksysb image is checked against this level to verify that they are the same.

The following flags are only valid for use when cloning the rootvg (-C):

-b bundle_name
Path name of optional file with a list of packages, or filesets, that will be installed after a rootvg clone. The -l flag must be used with this option.

-l installp_flags
The flags to use when updating, or installing, new filesets into the

cloned alt_inst_rootvg. The default flags are -acgX. The -l flag must be used with this option.

-l images_location
Location of the installp images, or updates, to apply after a clone of the rootvg. This can be a directory full path name or a device name (for example, /dev/rmt0).

-f fix_bundle
Optional file with a list of APARs to install after a clone of the rootvg. The -l flag must be used with this option.

-F fixes Optional list of APARs (such as IX123456) to install after a clone of the rootvg. The -l flag must be used with this option.

-e exclude_list
Optional exclude list to use when cloning rootvg. The rules for exclusion follow the pattern matching rules of the `grep` command. The exclude_list must be a full path name.

-w filesets
List of filesets to install after cloning a rootvg. The -l flag must be used with this option.

The following are supplied as parameters

Target Disks
Specifies the name, or names, of the target disks where the alternate rootvg will be created. The disk, or disks, must not currently contain any volume group definition. The `lspv` command should show these disks as belonging to volume group `None`.

6.8.2 Using alt_disk_install

The following are examples of using the `alt_disk_install` command:

1. To clone the running 4.2.0 rootvg to hdisk3 and apply updates from /updates to bring the cloned rootvg to a 4.2.1 level:

   ```
   #  alt_disk_install -C -F 4.2.1.0_AIX_ML -l /updates hdisk3
   ```

   The bootlist will then be set to boot from hdisk3 at the next reboot.

2. To install a 4.3 mksysb image on hdisk3, then run a customized script (/home/myscript) to copy some user files over to the alternate rootvg file systems before reboot:

```
#  alt_disk_install -d /mksysb_images/4.3_mksysb -s /home/myscript hdisk3
```

6.8.3 Alternate Disk Installation Enhancements (4.3.1)

The `alt_disk_install` command gives users another way to update AIX to the next release or maintenance level without having to schedule an extended period of system downtime. The function is included in the bos.alt_disk_install package, which is shipped on the AIX media. The package is not installed by default during system installation.

AIX 4.3.1 has enhanced the alternate disk installation function by splitting the task into three distinct phases. A system administrator now has greater control

over the task by being able to perform each phase in isolation from the others. It is not required to perform them all at once.

The three phases of the alternate disk install are:

1. Phase 1
   - Create the altinst_rootvg, logical volumes and file systems.
   - Restore or copy files to the /alt_inst file systems.
2. Phase 2
   - Copy a resolv.conf file to the /alt_inst/etc file system if specified.
   - Copy NIM client configuration information if specified.
   - For clones, install any new filesets, updates, and fixes.
   - Run any user specified customization script.
3. Phase 3
   - Manipulate the ODM databases and /etc/filesystems file.
   - Build the boot image.
   - Unmount the /alt_inst file systems and rename the logical volumes and file systems.

Phase two can be performed with phase one or phase three and can also be performed on its own multiple times if required before phase three is run.

The phases performed are controlled by the new -P option that has possible values as shown in Table 22.

Table 22. Possible Values of Phase Value

Flag value	Result
1	Phase one performed
2	Phase two performed
3	Phase three performed
12	Phases one and two performed
23	Phases two and three performed
all	Phases one, two, and three performed

The target disk name is required for all phases, even though the altinst_rootvg has already been created.

Some standard alt_disk_install options, such as the reboot option and the no bootlist option, are not allowed in phase one or phase two.

Specifying an exclude list (-e exclude_list) is not allowed in phase two or phase three.

If a flag is used in a wrong phase, a warning is displayed, but the install does not terminate.

6.8.4 Alternate Disk Installation Enhancements (4.3.2)

In this section, the enhancements made at the AIX 4.3.2 introduction are given.

6.8.4.1 New alt_disk_install 4.3.2 Usage

Listed below are the new command formats for the various tasks given.

Create Alternate Disk: mksysb (-d) or clone (-C):

```
alt_disk_install {-d <device> | -C} [-i <image.data>] [-s <script>]
                 [-R <resolv_conf>] [-D] [-B] [-V] [-r]
                 [-p <platform>] [-L <mksysb_level>]
                 [-b <bundle_name>] [-I <installp_flags>]
                 [-l <images_location>] [-f <fix_bundle>]
                 [-F <fixes>] [-e <exclude_list>] [-w <filesets>]
                 [-n] [-P <phase_option>] <target_disks...>
```

Determine Volume Group Boot Disk (-q):

```
alt_disk_install -q <disk>
```

Rename Alternate Disk Volume Group (-v):

```
alt_disk_install -v <new volume group name> <disk>
```

Wake-up Volume Group (-W):

```
alt_disk_install -W <disk>
```

Put-to-sleep Volume Group (-S):

```
alt_disk_install -S
```

Clean Up Alternate Disk Volume Group (-X):

```
alt_disk_install -X [<volume group>]
```

6.8.4.2 Scenarios for Command Enhancements

To see which disks belong to the original rootvg volume group after the system has been rebooted from the alternate disk, an entry has been added to the database, so that when the system is rebooted from the alternate disk, and the `lspv` command is executed, a volume group name old_rootvg will be listed for the original rootvg disks.

```
# lspv
hdisk0          00006091aef8b687    old_rootvg
hdisk1          00076443210a72ea    rootvg
```

This volume group will be set to NOT varyon at reboot, and should ONLY be removed with:

```
# alt_disk_install -X old_rootvg
# lspv
hdisk0          00006091aef8b687    None
hdisk1          00076443210a72ea    rootvg
```

New function was added to the -X flag to allow for specified volume group name information removal. It is recommended that you always use `alt_disk_install -X` when removing any information about an alternate volume group (altinst_rootvg, old_rootvg, and so on). alt_disk_install manages changes to the ODM that are required. Using `exportvg` or `reducevg` could cause serious problems to your system (like removing base entries in /etc/filesystems). `alt_disk_install -X` and `alt_disk_install -X <new volume group name>` will not remove actual data from the

volume group. Therefore, you can still reboot from that volume group, if you reset your bootlist.

To see which disk is the boot disk, of the old_rootvg volume group, after a reboot from the alternate disk, the -q flag has been added to `alt_disk_install` to determine the boot disk, from the user specified disk and its associated volume group. The command syntax is:

```
# alt_disk_install -q <disk>
```

Given any disk in the volume group, and the command will return the actual boot disk (contains hd5), for that volume group.

```
# lspv
hdisk0          00006091aef8b687    old_rootvg
hdisk1          00076443210a72ea    rootvg
hdisk2          0000875f48998649    old_rootvg
# alt_disk_install -q hdisk0
hdisk1
```

In this case, the boot disk for old_rootvg is actually hdisk1. Therefore, you could reset your bootlist to hdisk1 and reboot to the original rootvg volume group.

```
# bootlist -m normal hdisk1
# reboot -q
```

This query will work on any volume group that has a boot (hd5) logical volume.

Different names for altinst_rootvg are now possible for the case that a user would want to have multiple alternate disks on one system (one with 4.3.2, one with 4.2.1, and so on). The -v flag was added to allow non-rootvg volume group names to be changed. The syntax for this is:

```
# alt_disk_install -v <new volume group name> <disk>
```

For example, on a 4.2.1 system, run alt_disk_install to restore a 4.3.2 mksysb to hdisk2 and hdisk3. Execute alt_disk_install, with the -v flag, to rename the altinst_rootvg volume group name. Then, on the same system, run alt_disk_install to clone the 4.2.1 system to hdisk4 and hdisk5. Finally, rename the cloned altinst_rootvg.

```
# lspv
hdisk0          00006091aef8b687    rootvg
hdisk1          00000103000d1a78    rootvg
hdisk2          000040445043d9f3    None
hdisk3          00076443210a72ea    None
hdisk4          0000875f48998649    None
hdisk5          000005317c58000e    None
# alt_disk_install -d /dev/rmt0 hdisk2 hdisk3
...
# lspv
hdisk0          00006091aef8b687    rootvg
hdisk1          00000103000d1a78    rootvg
hdisk2          000040445043d9f3    altinst_rootvg
hdisk3          00076443210a72ea    altinst_rootvg
hdisk4          0000875f48998649    None
hdisk5          000005317c58000e    None
# alt_disk_install -v alt_disk_432 hdisk2
# lspv
```

```
hdisk0          00006091aef8b687    rootvg
hdisk1          00000103000d1a78    rootvg
hdisk2          000040445043d9f3    alt_disk_432
hdisk3          00076443210a72ea    alt_disk_432
hdisk4          0000875f48998649    None
hdisk5          000005317c58000e    None
# alt_disk_install -C hdisk4 hdisk5
...
# lspv
hdisk0          00006091aef8b687    rootvg
hdisk1          00000103000d1a78    rootvg
hdisk2          000040445043d9f3    alt_disk_432
hdisk3          00076443210a72ea    alt_disk_432
hdisk4          0000875f48998649    altinst_rootvg
hdisk5          000005317c58000e    altinst_rootvg
# alt_disk_install -v alt_disk_421 hdisk4
# lspv
hdisk0          00006091aef8b687    rootvg
hdisk1          00000103000d1a78    rootvg
hdisk2          000040445043d9f3    alt_disk_432
hdisk3          00076443210a72ea    alt_disk_432
hdisk4          0000875f48998649    alt_disk_421
hdisk5          000005317c58000e    alt_disk_421
# alt_disk_install -q hdisk3
hdisk2
# bootlist -m normal hdisk2
# sync
# reboot -q
```

After the system reboot, perform the following steps:

```
# lspv
hdisk0          00006091aef8b687    old_rootvg
hdisk1          00000103000d1a78    old_rootvg
hdisk2          000040445043d9f3    rootvg
hdisk3          00076443210a72ea    rootvg
hdisk4          0000875f48998649    alt_disk_421
hdisk5          000005317c58000e    alt_disk_421
```

A way to wake up a volume group for data access between the alternate disk and the original rootvg and also a way to put the volume group back to sleep is provided.

The syntax for the wake-up function is:

```
# alt_disk_install -W <disk>
```

Note, the volume group that will experience the wake-up will be renamed altinst_rootvg.

The booted volume group's version of AIX must be later or equal to the version of AIX on the volume group that will undergo the wake-up. This may mean that you will need to boot from the altinst_rootvg and wake-up the old_rootvg.

```
# oslevel
4.1.0.0
# lspv
hdisk0          000040445043d9f3    rootvg
hdisk1          00076443210a72ea    None
```

```
# alt_disk_install -d /dev/rmt0 hdisk1
...
# lspv
hdisk0          000040445043d9f3    rootvg
hdisk1          00076443210a72ea    altinst_rootvg
# reboot -q

After rebooting...

# oslevel
4.3.0.0
# lspv
hdisk0          000040445043d9f3    old_rootvg
hdisk1          00076443210a72ea    rootvg
# alt_disk_install -W hdisk0
# lspv
hdisk0          000040445043d9f3    altinst_rootvg
hdisk1          00076443210a72ea    rootvg
```

At this point, you will find the altinst_rootvg volume group varied-on and the /alt_inst file systems will be mounted.

Now, to put the volume group to sleep the command syntax is:

```
# alt_disk_install -S
# lspv
hdisk0          000040445043d9f3    altinst_rootvg
hdisk1          00076443210a72ea    rootvg
```

The altinst_rootvg is no longer varied-on and the /alt_inst file systems are no longer mounted. If desired and the bootlist is reset, this volume group is now ready for booting. If it is necessary for the altinst_rootvg volume group name to be changed back to old_rootvg, this can be done with the -v flag.

```
# alt_disk_install -v old_rootvg hdisk0
# lspv
hdisk0          000040445043d9f3    old_rootvg
hdisk1          00076443210a72ea    rootvg
```

6.9 Printer Support

The AIX spooler subsystem was already significantly enhanced over the basic UNIX spooler. For AIX Version 4.3, the following additional enhancements have been included:

6.9.1 Remote Printing Robustness

Both `rembak` and `lpd` have new flags that allow you to build a log file. A log file is helpful in determining why a daemon failed. Use the following commands to start error logging:

```
startsrc -s lpd -a "-D /tmp/lpddebug"

/etc/qconfig -D /tmp/remback_debug flag backend=/usr/lib/lpd/rembak
```

Support in SMIT was updated to allow `rembak` error logging to be enabled when adding a remote queue.

6.9.2 Remote Print Job Count

On a print server, when a job is received from the print client, `lpd` receives a control file and one or more data files. The control file contains information on the job to be printed, including the name of the corresponding data files. The datafiles contain the actual data to be printed. The control file is used to generate the arguments to the `enq` command. It is subsequently deleted. The datafiles are copied to the spooling directory so they can be processed by `qdaemon`.

The LPR/LPD protocol (RFC 1179) specifically designates the naming convention for print data files sent from print clients to print servers. Part of the name is a three-digit job ID. Due to this specification, problems could arise when a single print client sent more than 1000 print jobs to a print server. The datafiles became non-unique and the older file with the same name would get deleted. This could cause major problems for high-volume printing installations.

AIX version 4.3.0 has modified the function of the print server. When files are received by `lpd`, (before they are copied to the spooling directory) a time stamp is appended to the data file names, thus generating unique file names. This enables a large number of jobs to be submitted. The print server still conforms to the specification of the LPD protocol, which does not stipulate what happens to the data file once it has been received by the server.

6.9.3 Additional Printer Support

AIX Version 4.3 now includes native support for five additional Lexmark dot-matrix printers:

- Lexmark 2380 Model 3
- Lexmark 2381 Model 3
- Lexmark 2390 Model 3
- Lexmark 2391 Model 3
- Lexmark Forms Printer Model 4227

More printer support was specifically introduced in AIX 4.3.1:

- Hewlett-Packard 4000
- IBM InfoPrint 20

More printer support was specifically introduced in AIX 4.3.2 for IBM, Lexmark and Hewlett-Packard printers:

- IBM InfoPrint 32
- Lexmark Optra Color 40
- Lexmark Optra Color 45
- Lexmark Optra Color 1200
- Lexmark Optra K 1220
- Hewlett-Packard 8000
- Hewlett-Packard 8500 Color

The HP 8000 and HP 8500 Color printers and the associated AIX print drivers support A3 paper size.

For all printers, this includes:

- Printer backend colon files for each printer data stream supported by the printer.
- ODM parameters for the printer device driver and diagnostics.
- New message catalog entries for the printer name and new printer attributes (if required).
- Packaging files to make support for the printer's separately installable packages.

More information and colon files can be obtained from Lexmark. In the USA, call 1-800-Lexmark (1-800-539-6275) or visit their Web site at the following URL:

`http://www.lexmark.com`

6.9.4 Print Job Administration Enhancements (4.3.2)

The print queue administration commands have been enhanced to support print queues with more than 1000 jobs. Previous editions of AIX would allow more than 1000 jobs in a print queue. Cancelling or altering a job when the queue size grew more than 1000 became difficult, because job numbers would repeat, and specifying a specific job number would not guarantee that the job selected would be unique and the one desired.

The formatting of the output of the `qchk` command, when used with the -W flag, has been changed to show the six-figure print job number. The lpstat command has been changed to also accept the -W flag to show information in wide format. Use of the -W flag results in output where the lines are over 106 characters in length. It can be quite confusing to read the output when using a screen that is only 80 characters wide. To maintain compatibility for any user scripts that parse the output of these commands, the default format for both remains unchanged from previous versions.

```
# qchk -W
Queue                Dev            Status       Job Files              User
     PP   %  Blks  Cp Rnk
-------------------- -------------- --------- ------ ------------------ --------
-- ---- --- ----- --- ---
ps                   lp0            DOWN
                                    QUEUED      2228 /etc/passwd        root
                1   1   1
                                    QUEUED      2229 /etc/passwd        root
                1   1   2
                                    QUEUED      2230 /etc/passwd        root
                1   1   3
                                    QUEUED      2231 /etc/passwd        root
                1   1   4
#
```

The `enq`, `cancel`, `qpri`, `qcan`, and `lprm` commands have been altered to accept six-figure job numbers.

The enhancement applies only to jobs submitted to local print queues. Jobs submitted to remote printers will still have three digit print job numbers. This is because of a restriction in the lpd protocol.

Although AIX 4.3.2 now generates six digit job numbers for local jobs, the response time of the `qchk` and `lpstat` commands is identical to that on AIX 4.3.0. The time taken to list the jobs on the queue is proportional to the number of jobs. It is suggested that you maintain a queue size less than 1000 unless absolutely necessary, because larger queue sizes will impact performance.

6.10 System Resource Controller Subsystem Enhancements (4.3.2)

Two major enhancements have been introduced to the System Resource Controller (SRC) subsystem in AIX 4.3.2. They are aimed at increasing the reliability and scalability of both the various subsystems that are controlled by SRC and the SRC itself. The following sections explain these enhancements:

6.10.1 Recoverable SRC Daemon

The SRC is a subsystem controller that facilitates the management and control of complex subsystems. The SRC provides a single set of commands to start, stop, trace, refresh, and query the status of a subsystem. If a subsystem should fail for any reason, the SRC can automatically restart it.

If the SRC itself were to fail for any reason, it would be restarted due to its entry in /etc/inittab, as shown in the following example:

```
srcmstr:2:respawn:/usr/sbin/srcmstr # System Resource Controller
```

The respawned SRC is, however, unable to control or monitor the subsystems started by the previous instance of SRC since they will have been inherited by the `init` process when the original SRC terminated. As a result, the `lssrc` command will show such a subsystem as inoperative, even though it is still running. In addition, the `startsrc` command can be used to start a second instance of the subsystem, even though the subsystem definition explicitly forbids multiple instances.

The SRC in AIX 4.3.2 has been enhanced to allow a respawned `srcmstr` daemon to monitor and control the subsystems started by the previous instance of the daemon. This has been achieved using the following enhancements.

The SRC now keeps an external list of the subsystems under its control in the file /var/adm/SRC/active_list. This file is for use by the SRC system only, therefore the format is unpublished. A respawned `srcmstr` daemon will read the contents of this file to update its internal list of the currently running subsystems. This will allow the `lssrc` command to correctly determine the status of the running subsystems, even though they were not necessarily started by the current instance of the `srcmstr` daemon.

A respawned `srcmstr` daemon uses a new kernel extension to register interest in the termination of certain processes. This allows a respawned `srcmstr` daemon to be informed of the termination of subsystems started by the previous instance of the daemon. A child process is created to communicate with the kernel extension. The child process in turn communicates with the srcmstr daemon. The presence of the child process, called `srcd`, indicates that the `srcmstr` daemon has been restarted.

```
# ps -ef | grep src
    root  4650     1   0   Aug 21      -  0:00 /usr/sbin/srcmstr
    root 24680  4650   0   Aug 21      -  0:00 srcd
    root 25894  7030   2 10:03:38  pts/1  0:00 grep src
#
```

If a subsystem fails for any reason while under the control of a respawned `srcmstr` daemon, it will be restarted if the subsystem policy requires it. In this case the exit code of the subsystem is not available to SRC due to the method used by the

kernel extension to detect process termination. This is still preferable however to the previous function of SRC that would not detect subsystem failure at all in this situation.

If you run the `lssrc` command with the -S flag, you receive a list of the subsystem attributes. The *action* is set to -R (for respawn) or -O (for once). The value of action must be -R to have the subsystem restarted. Also, there is a retry limit. If the subsystem fails more than once within the configured waittime (20 seconds by default), it will not be restarted.

The action and waittime attributes can be set using the `mkssys` command or changed with the `chssys` command.

This new feature of the `srcmstr` daemon can be disabled if required by specifying the -B option when starting the daemon. This is usually performed by an entry in /etc/inittab.

6.10.2 Thread-Safe Routines in libsrc

In previous versions of AIX, some of the libsrc subroutines are neither threadsafe nor reentrant. This prevents other libraries and applications that call these libsrc subroutines from achieving thread-safety and reentrance requirements.

The libsrc subroutines of interest in a threaded environment are those that support communication with an SRC subsystem. In other words, the routines that are used by an application, that may be multi-threaded, to interrogate the SRC.

The libsrc subroutines that update the subsystem configuration data, and those that are used by SRC commands to process input parameters, are not required in a threaded environment. This is because the applications that use these routines, the srcmstr daemon itself along with `lssrc` and related commands, are not threaded applications.

The threadsafe and reentrant routines are shown in Table 23. The new function has been implemented by changing the internals of some routines and by providing new threadsafe and reentrant versions of other routines. The new routines are indicated by the _r extension on the name. Where a new routine has been implemented, the original non-threadsafe version has been retained for use by non-threaded applications and for binary compatibility with previous versions of AIX.

Table 23. Threadsafe Routines in libsrc

New threadsafe routines	Existing routines made threadsafe
src_err_msg_r	srcsrpy
srcrrqs_r	srcstathdr
srcstattxt_r	srcstop
srcstat_r	srcstrt
srcsrqt_r	
srcsbuf_r	

6.11 TTY Remote Reboot (4.3.2)

AIX 4.3.2 has added the ability to communicate with a system that has stopped responding on the network but is still processing device interrupts. The feature allows a system administrator to force a machine to take a predetermined action when a user defined character sequence is entered on a serial port. The feature can only be enabled on native serial ports. Only one serial port on a machine can be configured for remote reboot. Serial ports configured on 8, 16, 64, and 128 port adapter cards are not supported.

The feature is configured by setting two ODM attributes that have been added to native serial ports. The new attributes are reboot_enable and reboot_string. The reboot_enable attribute has possible values of no, reboot, and dump. The reboot_string attribute is used to store a case sensitive user defined string up to 16 characters in length. It is advised to choose an unusual character sequence that would never normally be typed. For example ReEbOoTmE. This allows the serial port to be used for a normal login session, if required, without the possibility of accidentally rebooting the system.

Table 24. Settings of reboot_enable Attribute

Value of reboot_enable Attribute	Result
no	Remote reboot disabled. No action taken if reboot_string is entered.
reboot	Machine will reboot when reboot_string is entered and confirmed.
dump	Machine will dump system image to dump device when reboot_string is entered and confirmed.

The settings appear on the **SMIT Add a TTY**, and **Change / Show Characteristics of a TTY** panels as:

```
REMOTE reboot ENABLE                                    no
REMOTE reboot STRING                                   [#@reb@#]
```

Interrupts must be enabled on the port for this feature to be active. One way to insure interrupts are enabled is to enable login on the port; with the port enabled, getty is running and holding the device open, although the user does not need to be logged in for the system to recognize the reboot string.

If the user defined reboot string is entered when reboot_enable is set to reboot or dump, the user defined string is erased from the screen and replaced with the symbol > that is the confirmation prompt. If the user presses the **1** key on the keyboard, then the predefined action specified by reboot_enable will occur. If the user presses any other key, the user defined string reappears on screen; the subsequent character is appended, and no other action is taken.

An error log entry is made when the remote reboot facility is enabled or disabled on a serial port. An entry is also made when the facility is used to reboot a machine or force a system dump. The entry is created when the machine next starts the errorlog daemon indicating the action taken and the name of the tty device used to initiate the action.

```
---------------------------------------------------------------------------
LABEL:          TTY_RRB
IDENTIFIER:     1960E672

Date/Time:       Fri Aug 28 14:54:41
Sequence Number: 20
Machine Id:      000044091C00
Node Id:         aix4xdev
Class:           O
Type:            INFO
Resource Name:   Remote Reboot

Description
SYSTEM REBOOTED USING TTY REMOTE REBOOT.

User Causes
SYSTEM REBOOTED USING TTY REMOTE REBOOT.

Detail Data
TTY LOGICAL NAME
tty0
---------------------------------------------------------------------------
```

The remote reboot function is intended to be used on remote server machines that do not have a service processor. Ordinarily, the serial port with remote reboot enabled would be connected to a modem to allow remote support staff to reboot the machine if it fails to respond on the network in the normal manner.

It is the system administrators responsibility to provide physical security on any serial port with remote reboot enabled. This is because any user can determine the reboot string by using the `lsattr` command with the appropriate logical device name. It is not possible to enable a password protected reboot string, as this would require the code checking the password to use the crypt() function. Since the code checking the string is running at the highest interrupt priority, any increase in the time taken to service the interrupt may cause other device interrupts to be lost with unpredictable results.

6.12 Network Install Manager Enhancements (4.3.2)

The Network Install Manager (NIM) subsystem has been enhanced in AIX 4.3.2 to offer greater control over NIM operations. The system has been changed to allow more concurrent NIM operations and restrict the number of concurrent NIM operations.

6.12.1 Restrict Concurrent Group Operations

A NIM machine group allows an administrator to use a single command to initiate the same NIM action on many machines at the same time. Depending on the NIM operation and numbers of machines involved, this can sometimes lead to resource constraints. For example, many machines performing a BOS install action could saturate a network segment.

Two new settings are available when performing NIM operations on group resources. Together they allow the administrator to specify how many concurrent operations should be attempted on machines in the group and for how long the NIM server should continue to initiate the operations.

For example, this would allow the administrator of a NIM environment with a machine group of 100 machines to initiate a NIM operation on the group and to specify that no more than 10 machines in the group should have the operation in progress at any one time. This ensures that the network bandwidth is not

exhaustively consumed. When a NIM operation completes on a client machine, the NIM server initiates an operation on the next machine in the group until all group members have been processed, or the time limit has been exceeded. The options are in place for the duration of the NIM operation. Subsequent NIM operations on the group can use different values if desired.

The options are valid only for certain operations when a NIM group is used as the target. The NIM operation will fail with an error message if the options are used for individual machine, LPP, or SPOT targets. The options appear near the end of the following NIM SMIT panels that initiate operations likely to generate large amounts of network traffic:

- Install the Base Operating System on Stand-alone Clients
- Install and Update from LATEST Available Software
- Update Installed Software to Latest Level (Update All)
- Install and Update Software by Package Name (includes devices and printers)
- Install Software Bundle (Easy Install)
- Update Software by Fix (APAR)
- Install and Update from ALL Available Software
- Install mksysb on an Alternate Disk
- Clone the rootvg to an Alternate Disk

Figure 29 shows an example of the new NIM settings within SMIT.

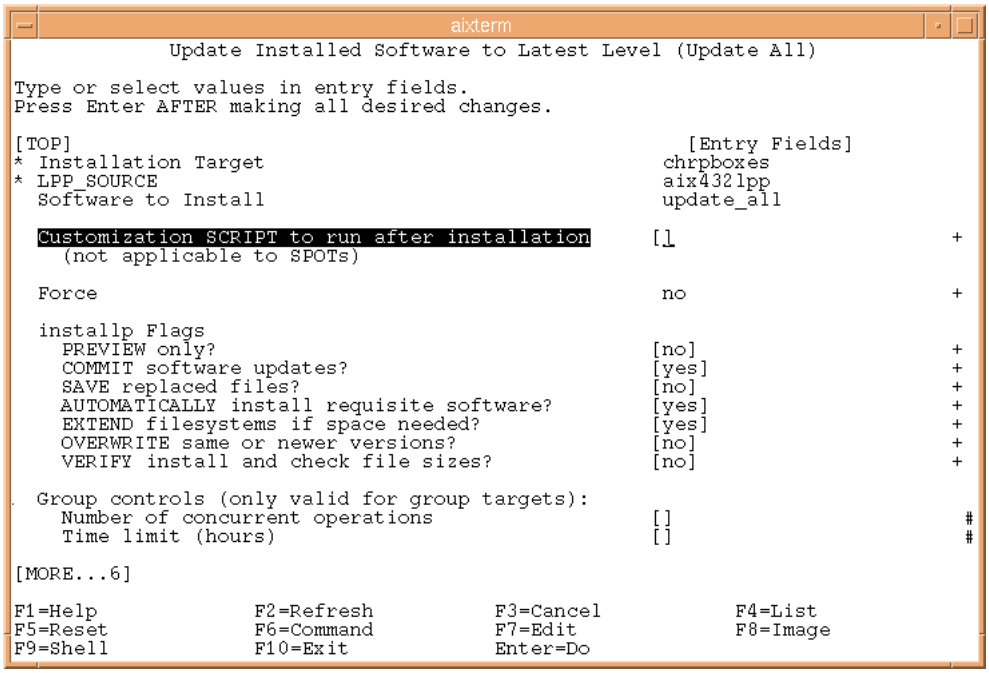


Figure 29. Sample NIM SMIT Panel Showing Group Controls

6.12.2 Resource Lock Contention

The lock granularity of the NIM subsystem has been improved to allow more operations in parallel. Previous versions of NIM would lock an object for the duration of some operations, thus preventing any other operation on the same object.

The locking methodology has been changed to lock the object only for critical parts of the operation. This will allow other operations on the object to complete when in the past they may have waited or timed out. For example, this change will allow a showlog operation to be carried out on a machine resource, which a customer operation is also being carried out. Previously, the machine object would be locked for the entire duration of the customer operation.

6.12.3 Administration Enhancements

The NIM sections of SMIT and Web-Based System Manager have been updated to offer function that was previously only available using the NIM command line interface. This includes support for ATM network types and IEEE 802.3 Ethernet networks.

6.13 Paging Space Enhancements (4.3.2)

AIX Version 4.3.2 now provides support for up to 32 GBs of memory on RS/6000 64-bit SMP servers. Before AIX 4.3.2, paging space was allocated for the executing process at the time the memory was requested or accessed. This required backing paging space allocated for all pages in the real memory, to save the image of the page. On a large-memory machine where paging was never or rarely required, these paging space blocks were allocated but never be used. In this case, resources were wasted.

In AIX 4.3.2, the policy for paging space allocation has been modified to allow a deferred paging space allocation. The allocation of paging space is delayed until it is necessary to page out the page, which results in no wasted paging space allocation. This new paging space allocation method greatly reduces the paging space requirements for systems with large physical memory.

6.13.1 Late and Early Paging Space Allocation

There are three kinds of paging space allocation policies used with AIX. The setting of the PSALLOC environment variable determines the paging space allocation mode.

Early Allocation

If the environment variable PSALLOC is set to early, then the early allocation policy is used. This will cause a disk block to be allocated whenever a memory request is made. If there is insufficient paging space available at the time of the request, the early allocation mechanism fails the memory request. This guarantees that the paging space will be available if it is needed.

Late Allocation in Pre-AIX 4.3.2

If the environment variable is not set, then the default late allocation policy is used and a disk block is allocated only when a page in memory is initially accessed, not when it is allocated.

Late Allocation in AIX 4.3.2

AIX 4.3.2 modifies the late allocation policy so that a disk block is not allocated until it becomes necessary to page out the page from memory into paging space. In AIX 4.3.2, late allocation will not allocate any disk blocks if there is enough real memory and no paging required for a given application set.

In summary, Table 25 shows the different policies used in various AIX versions.

Table 25. Paging Space Allocation Policies

<table>
<tr><td rowspan="2">PSALLOC = early</td><td colspan="2">All AIX versions</td></tr>
<tr><td colspan="2">Paging space is allocated when memory is requested</td></tr>
<tr><td rowspan="2">PSALLOC is not set or set to any value other than early</td><td>Pre-AIX 4.3.2</td><td>AIX 4.3.2</td></tr>
<tr><td>Paging space is allocated when the page in memory is accessed</td><td>Paging space is allocated when the page in memory needs to be paged out</td></tr>
</table>

Paging space slots are only released by process (not thread) termination or by the disclaim system call. They are not released by the free call.

6.13.1.1 Early Paging Allocation Mode Considerations

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, will run in the early allocation environment. The early allocation algorithm causes the appropriate number of paging space slots to be allocated at the time the virtual-memory address range is allocated. For example, with malloc. Interfaces, such as the malloc subroutine and the brk subroutine, will fail if sufficient paging space cannot be reserved when the request is made.

The early allocation algorithm guarantees as much paging space as requested by a memory allocation request. Thus, proper 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 may 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, since 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 will almost always be 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.

6.13.1.2 Late Paging Allocation

If the environment variable PSALLOC is not set, is set to null, or is set to any value other than early, the default late paging space allocation policy is used, and a disk block is allocated only when a page is initially used, not when a memory request is made.

The default late allocation algorithm for paging space allocation assists in the efficient use of disk resources and supports applications of customers who wish to take advantage of a sparse allocation algorithm for resource management.

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.

6.13.2 Commands Affected by Late Paging

The following commands are affected by the change in paging policy.

6.13.2.1 vmstat Command Updates

The avm column reported by `vmstat` command means active virtual pages. In previous AIX versions, the description of avm states "virtual pages are considered active if they are allocated". This is not true for every release of AIX and is changed to "virtual pages are considered active if they have been accessed".

6.13.2.2 lsps Command Updates

If you set the environment variable PSALLOC=early, the -s flag displays a value different from the value returned when using the -a flag for all the paging spaces. In this case, the value of -s flag displays the percentage of paging space allocated (reserved), whether the paging space has been assigned (used) or not. The -a flag specifies the percentage of paging space used. Therefore, the percentage reported by the -s flag is usually larger than that reported by the -a flag.

The following is an example. First, set the paging space allocation to early:

```
#export PSALLOC=early
```

After the system is running for some time, the paging space looks like:

```
#lsps -a
Page Space  Physical Volume   Volume Group    Size   %Used  Active  Auto  Type
hd6         hdisk0             rootvg         256MB      8     yes   yes    lv
#lsps -s
Total Paging Space   Percent Used
      256MB               9%
```

The paging space used reported by using -s (9%) is larger than using -a (8%).

Set the paging space allocation to late:

```
#export PSALLOC=
```

The `lsps` command displays the same percentage value with -a and -s flag (8%).

```
# lsps -a
Page Space  Physical Volume   Volume Group    Size   %Used  Active  Auto  Type
hd6         hdisk0            rootvg         256MB      8     yes   yes    lv

# lsps -s
Total Paging Space   Percent Used
      256MB               8%
```

6.14 Error Message Templates (4.3.2)

When an application wants to log an error to the AIX error log, it writes information about the error, specifically the error identifier, resource name, and error specific data to the /dev/error special file. The error daemon reads from the special file and logs the information in the error log. The `errpt` command, which is used to display error messages, reads the error message template repository to determine how to interpret and display the error data.

On previous versions of AIX, it is not possible for an error message template definition to contain any message text. It can only contain codepoints, which are two byte message numbers used to reference predefined message strings. The codepoints refer to messages defined by the IBM SNA Generic Alert Architecture described in *SNA Formats*, GA27-3136. The architecture imposes restrictions on the message numbering and message length that can be used. The message strings are kept in the codepoint.cat file, which is a specially formatted message catalog.

AIX 4.3.2 has updated the `errpt` command, along with the commands used for updating the error template repository, to understand an additional error template format, which can define messages from a normal format NLS message catalog as well as using the previous codepoint method.

A template may contain all NLS messages, all codepoints, or a combination of both. An NLS message is represented with a message set number, a message number, and a default text string to be printed if the associated message catalog is not present. Each error template also specifies the message catalog to be used for messages referenced by that template.

```
*!sample.cat
* sample error template using NLS messages
+ SAMPLE:
        Err_Type = UNKN
        Class = S
        Report = TRUE
        Log = TRUE
        Alert = FALSE
        Comment = "Sample of msgs in templates"
        Err_Desc = {1, 2, "SAMPLE DESCRIPTION TEXT"}
        Fail_Causes = {0, 0xeb54, ""), (1, 5, "default cause"},
           {3, 5, "default cause 2"}, {3, 6, "default cause3"}
        Prob_Causes = {2, 1, "Bad Operator"}, {2, 2, "Bad Programmer"}
        User_Causes = {2, 5, "User pressing wrong key"}
        User_Actions = {2, 6, "Read the manual"},
           {2, 7, "Take a long long long\n\t\
long, really long, look at the manual."}
        Inst_Causes = {0, 0, ""}
        Inst_Actions= {2, 8, "reinstall"}
        Fail_Actions = {2, 9, "kick it"}
        Detail_Data = 226, {0, 0x8004,""} ,ALPHA
        Detail_Data = 4, {2, 10, "regester value"} ,HEX
```

This dramatically increases the number of messages that can be used by an error template, as it is no longer restricted to the messages defined by the IBM Alert Architecture. The trade off is that the new error messages are no longer alertable.

In addition to increasing the number of messages available to an error template, this enhancement has also increased the number of detail data items up to a maximum of sixteen.

6.15 Remote File Distribution Enhancements (4.3.2)

Previous versions of AIX included Version 5.1 of the `rdist` command, which is used to distribute and maintain identical copies of files on multiple hosts. The `rdist` command on AIX 4.3.2 has been updated to Version 6.1.3, which includes some new features, namely:

- Multiple target hosts are now updated in parallel. This improves the update time when working with large numbers of hosts. This behavior can be controlled by changing the number of hosts updated in parallel, or disabling the feature, in which case, hosts are updated sequentially.
- The new version of `rdist` avoids problems when communicating with a remote host by setting a time-out value. If the remote host fails to respond within a set period during a transfer, `rdist` displays an error message and continues to update other hosts. The previous version of `rdist` would continue to wait until the remote host responded.
- Local and remote error messages are distinctly marked for better clarity.
- The amount of free space can optionally be checked to avoid filling up a filesystem. Before actually installing or updating a file, `rdist` will calculate whether the update would exceed the minimum amount of free space as specified on the command line. If the minimum space would be exceeded by the update, no update is performed and an error message is displayed.
- The client and server portions are split into two distinct programs, `rdist` and rdistd. This lowers the risk of security vulnerabilities since the server rdistd does not need to be setuid to root. It also allows for greater ease in maintaining different versions of `rdist`.

Version 6.1 of `rdist` implements a new protocol for communicating between machines. Both versions of the `rdist` command are shipped with AIX 4.3.2 to allow users to distribute files to machines running either version of `rdist`. The new version is shipped as `/usr/bin/rdist`, and the old version as `/usr/bin/oldrdist`.

When the `rdist` program contacts a target machine, it requests the target to start the `rdist` server side program. Version 6.1 `rdist` will start the rdistd server program. Version 5.1 `rdist` requests the target machine to run `rdist -Server`. If the version 6.1 `rdist` is run with the -Server option, then it will exec a copy of `oldrdist`. In this way, you can get compatibility with hosts running `rdist` Version 5.1 attempting to distribute files to a machine running `rdist` Version 6.1. If a host running `rdist` Version 6.1 wants to distribute files to a host running the `rdist` Version 5.1, then it must run the `oldrdist` program.

6.16 Editor Enhancements (4.3.2)

The `ed` editor program has been enhanced to examine the environment variable EDTMPDIR to determine the directory location for temporary files. This has been done to allow a system to better handle the start up of multiple `ed` sessions by avoiding a bottleneck on the inode for the default temporary directory used by `ed`.

6.17 System Backup Usability Enhancements (4.3.2)

The mksysb and savevg commands have been enhanced to include information in the backup image about block size of the tape device being used to store the backup image.

This change means that when using SMIT to list the contents of the backup, restore individual files, or restore the complete backup. The system can read the information on the tape indicating the block size used to create the image. It can then change the block size of the tape device being used to read the backup to be the same, therefore maximizing the data transfer rate. Once the operation to list, or restore the backup has completed, the system changes the block size of the tape device back to the previous setting.

The following SMIT panels have been updated to include a new option that allows the user to specify whether the system should attempt to determine the tape block size used to create the backup image:

- List Files in a System Image
- Restore Files in a System Image
- List Files in a Volume Group Backup
- Restore Files in a Volume Group Backup

An example of the change is shown in Figure 30. If the option is set to yes, and the tape being read was created on a previous version of AIX that did not include the tape block size information on the tape, then the underlying commands will set the block size to 0 and continue with the required operation.

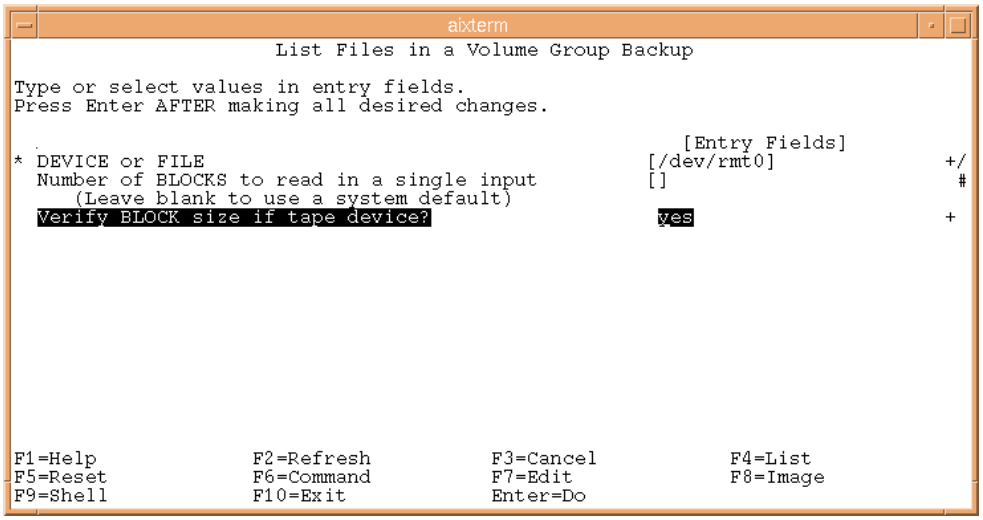


Figure 30. Sample SMIT Volume Group Backup Screen

6.18 Operating System Install Enhancement (4.3.2)

The function of the non-prompted install method has been improved to provide a means of protecting user defined volume groups already on the system. A non-prompted install can be carried out by supplying a customized bosinst.data

file when restoring a mksysb or installing the base operating system from CD-ROM, tape, or NIM server.

The EXISTING_SYSTEM_OVERWRITE variable now has three possible values, provided in Table 26, which determine the action taken.

Table 26. Possible Values of EXISTING_SYSTEM_OVERWRITE

EXISTING_SYSTEM_OVERWRITE=	Action Taken
any	Any disk can be used for the system install. This is the behavior of the 'yes' option on releases prior to AIX 4.3.2.
no	Only disks containing no volume groups (user defined or previous rootvg) can be used.
yes	Only disks in the current rootvg, or containing no volume groups, can be used for the system install

The value of EXISTING_SYSTEM_OVERWRITE is only examined if the bosinst.data file also sets PROMPT=no and INSTALL_METHOD=overwrite.

6.19 New Diagnostic Service Aid (4.3.2)

The diagnostics subsystem in AIX 4.3.2 has been enhanced by the addition of a system memory exerciser, which can be used to check system memory on CHRP systems. The tool is implemented as a service aid and is only available when running on-line diagnostics in service or maintenance modes. Service mode diagnostics are entered when the machine boots in service mode. Maintenance mode diagnostics are entered by first taking the machine to maintenance mode using the `shutdown -m` command.

The machine to be tested is required to have the bos.acct package installed and paging space of at least one and a half times the amount of physical memory. The service aid will exit with an error if the bos.acct package is not installed. If the paging space requirement is not met, the service aid will give a warning message informing the user that it is not possible to test the maximum amount of physical memory.

The service aid will then display the memory exerciser options screen, which allows the user to select the characteristics of the testing that will be performed.

The service aid memory exerciser options screen is shown in Figure 31.

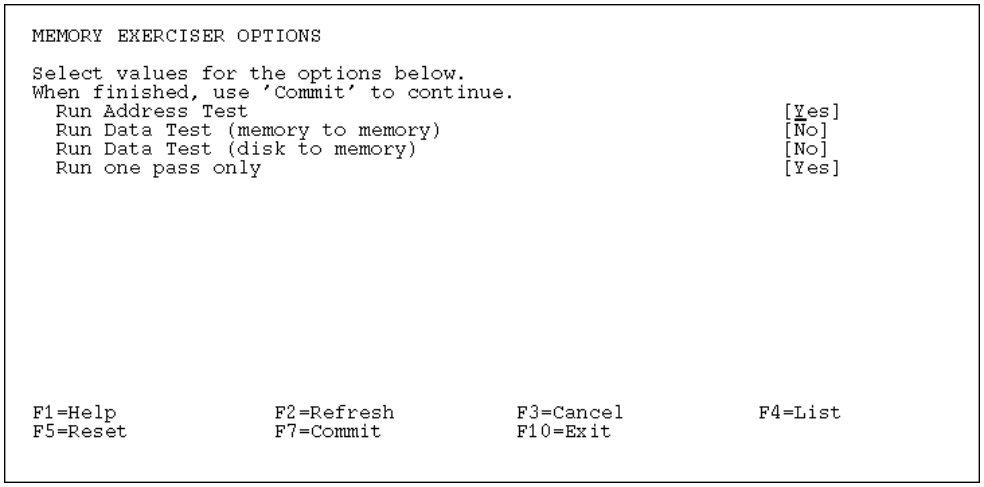


Figure 31. Memory Exerciser Options Menu

Once the desired options have been selected, the system exerciser menu (Figure 32) is displayed, which allows the user to start and stop the exerciser and view the error logs.

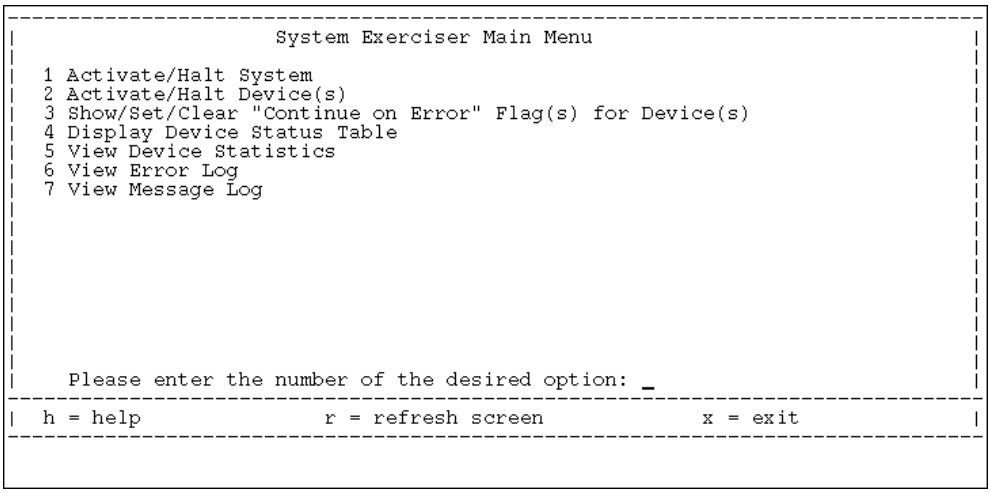


Figure 32. System Exerciser Main Menu

6.20 Performance Toolbox Agent Repacking (4.3.2)

The Performance Toolbox is a Motif-based AIX licensed program product (LPP) that consolidates AIX performance tools into a toolbox framework. Users can easily access tools for system and network performance tuning, monitoring, and analysis. It consists of two major components: Performance Toolbox Manager and Performance Toolbox Agent.

The Performance Toolbox Manager has three packages:

perfmgr.local This package contains the commands and utilities that allow monitoring of only the local system.

perfmgr.network This package contains the commands and utilities that allow monitoring of remote systems as well as the local system.

perfmgr.common This package contains the commands and utilities that are common between the network support and the local support.

The Performance Toolbox Agent has one package:

perfagent.server This package contains the performance agent component required by Performance Toolbox as well as some local AIX analysis and control tools.

The packaging of the previous Performance Toolbox contained two filesets: perfagent.server and perfagent.tool causing installation difficulty. To improve this process, those pieces that are required to be built with the AIX kernel are moved into the perfagent.tools fileset. Then the agent becomes mainly an interface routine to those pieces.

The perfagent.tools fileset is shipped with the AIX 4.3.2 base. For AIX 4.3.2, The Performance Toolbox Agent will prereq perfagent.tools. So the .tools fileset must be installed first.

Note: The on-line *PTX Guide and Reference for AIX 4.3* (perfagent.html.en_US) is available by ordering APAR IX80484 (PTF U458736). In AIX Version 4.3.0, this document is shipped with the Performance Aide CD. In AIX Version 4.3.2, it is included in the base AIX documentation.

Table 27 lists the various minimum file set levels required with a particular AIX level.

Table 27. AIX Level and Required File Sets

AIX Version	File Set
AIX 4.1.5	perfagent.tools 2.1.6.* perfagent.server 2.1.6.*
AIX 4.2.1	perfagent.tools 2.2.1.* perfagent.server 2.2.1.*
AIX 4.3.1	perfagent.tools 2.2.31.* perfagent.server 2.2.31.* (replaced by 32)
AIX 4.3.0	perfagent.tool 2.2.32.0 perfagent.server 2.2.32.0 (prereqs 3.3.32.0 perfagent.tools)

Chapter 7. Networking Enhancements

Internet Protocol Version 6 (IPv6) was first introduced in AIX version 4.3. In AIX version 4.3.2, IPV6 routing is supported. Some important network improvements for Web servers are also supported in AIX version 4.3.2.

In these sections, the networking enhancements in AIX version 4.3, 4.3.1, and 4.3.2 are discussed.

7.1 Internet Protocol Version 6

Internet Protocol Version 6 (IPv6) is the next generation Internet Engineering Task Force (IETF) networking protocol that will become the industry standard network protocol for the internet of the future.

IPv6 extends the maximum number of Internet addresses to handle the ever-increasing Internet user population. IPv6 is an evolutionary change from IPv4 that has the advantage of allowing a mixture of the new and the old to coexist on the same network. This coexistence enables an orderly migration from IPv4 (32-bit addressing) to IPv6 (128-bit addressing) on an operational network.

7.1.1 IPv6 Introduction

This initial release of IPv6 in AIX 4.3.0 is a migration platform to enable user migration to IPv6. The AIX IPv6 migration platform in AIX 4.3.0 supports IPv6 host function only. This means that no gateway support is included, so IPv6 packets cannot be forwarded from one interface to another on the same RS/6000.

The following function is included in this release:

- IPv6 128-bit addressing
- Neighbor Discovery/Stateless Address Autoconfiguration
- Internet Control Message Protocol (ICMPv6)
- Tunneling over IPv4
- IP Security (IPSec)
- Resolver support for /etc/hosts
- Commands/applications enabled for IPv6
- IPv6 Socket Library Support
- System management changes

The following network media is supported in the AIX IPv6 migration platform:

- Ethernet
- Token-Ring
- FDDI

7.1.2 IPv6 128-Bit Addressing

A brief introduction to IPv6 addressing is presented here. For more detail, refer to the appropriate RFC documents. See section 7.1.11, “IPv6 and IPSec-Related RFCs Implementation” on page 166 for a listing of RFCs supported by the AIX IPv6 migration platform in AIX 4.3.0.

7.1.2.1 Text Representation of Addresses

As shown in the following example, an IPv6 address is represented by hexadecimal digits separated by colons, where IPv4 addresses are represented by decimal digits separated by dots or full-stops. IPv6 is, therefore, also known as colon-hex addressing, compared to IPv4's dotted-decimal notation.

IPv6 addresses are 128-bit identifiers for interfaces and sets of interfaces. Note that IPv6 refers to *interfaces* and not to *hosts*, common to IPv4.

There are three conventional forms for representing IPv6 addresses as text strings:

1. The preferred form is `x:x:x:x:x:x:x:x`, where the `x`'s are the hexadecimal values of the eight 16-bit pieces of the address, each separated by a colon.

   Examples are:

   ```
   FEDC:BA98:7654:3210:FEDC:BA98:7654:3210
   1080:0:0:0:8:800:200C:417A
   ```

   Note that it is not necessary to write the leading zeros in an individual field, but there must be at least one numeral in every field (except for the case described in 2).

2. Due to the method used for allocating certain styles of IPv6 addresses, it will be common for addresses to contain long strings of zero bits. To make writing addresses containing zero bits easier, a special syntax is available to compress the zeros. The use of `::` (two colons) indicates multiple groups of 16-bits of zeros. Note that the `::` can only appear once in an address. The `::` can also be used to compress the leading and/or trailing zeros in an address.

   For example, the following addresses:

   ```
   1080:0:0:0:8:800:200C:417A  a unicast address
   FF01:0:0:0:0:0:0:43         a multicast address
   0:0:0:0:0:0:0:1             the loopback address
   0:0:0:0:0:0:0:0             the unspecified addresses
   ```

   may be represented as:

   ```
   1080::8:800:200C:417A       a unicast address
   FF01::43                    a multicast address
   ::1                         the loopback address
   ::                          the unspecified addresses
   ```

3. An alternative form that is sometimes more convenient when dealing with a mixed environment of IPv4 and IPv6 nodes is `x:x:x:x:x:x:d.d.d.d`, where `x` is the hexadecimal values of the six high-order 16-bit pieces of the address, and `d` is the decimal values of the four low-order 8-bit pieces of the address (standard IPv4 representation).

   Examples:

   ```
   0:0:0:0:0:0:13.1.68.3
   ```

```
0:0:0:0:0:FFFF:129.144.52.38
```

or in compressed form:

```
::13.1.68.3
::FFFF:129.144.52.38
```

Note: FFFF is used to represent addresses of IPv4-only nodes (those that do not support IPv6).

7.1.2.2 Types of IPv6 Address

In IPv6, there are three types of addresses:

Unicast

An identifier for a single interface. A packet sent to a unicast address is delivered to the interface identified by that address. A unicast address has a particular scope as shown in the following lists:

- link-local
  - Valid only on the local link (that is, only one hop away).
  - Prefix is `fe80::/16`.
- site-local
  - Valid only at the local site (for example, inside IBM Austin).
  - Prefix is `fec0::/16`.
- global
  - Valid anywhere in the Internet.
  - Prefix may be allocated from unassigned unicast space.

There are also two special unicast addresses:

- `::/128` (unspecified address).
- `::1/128` (loopback address - note that in IPv6 this is only one address not an entire network).

Multicast

An identifier for a set of interfaces (typically belonging to different nodes). A packet sent to a multicast address is delivered to all interfaces identified by that address. A multicast address is identified by the prefix `ff::/8`. As with unicast addresses, multicast addresses have a similar scope. This is shown in the following lists:

- Node-local
  - Valid only on the source node (for example, multiple processes listening on a port).
  - Prefix is `ff01::/16` or `ff11::/16`.
- Link-local
  - Valid only on hosts sharing a link with the source node (for example, Neighbor Discovery Protocol [NDP] data).
  - Prefix is `ff02::/16` or `ff12::/16`.

- Site-local
  - Valid only on hosts sharing a site with the source node (for example, multicasts within IBM Austin).
  - Prefix is `ff05::/16` or `ff15::/16`.
- Organization-local.
  - Valid only on hosts sharing organization with the source node (for example, multicasts to all of IBM).
  - Prefix is `ff08::/16` or `ff18::/16`.
- The `0` or `1` part in these prefixes indicates whether the address is permanently assigned (`1`) or temporarily assigned (`0`).

Anycast

An identifier for a set of interfaces (typically belonging to different nodes). An anycast address is an address that has a single sender, multiple listeners, and only one responder (normally the nearest one, according to the routing protocols' measure of distance). An example may be several Web servers listening on an anycast address. When a request is sent to the anycast address, only one responds.

Anycast addresses are indistinguishable from unicast addresses. A unicast address becomes an anycast address when more than one interface is configured with that address.

Note: There are no broadcast addresses in IPv6, their function being superseded by multicast addresses.

7.1.3 Neighbor Discovery/Stateless Address Autoconfiguration

Neighbor Discovery (ND) protocol for IPv6 is used by nodes (hosts and routers) to determine the link-layer addresses for neighbors known to reside on attached links and maintain per-destination routing tables for active connections. Hosts also use Neighbor Discovery to find neighboring routers that forward packets on their behalf and detect changed link-layer addresses. Neighbor Discovery protocol (NDP) uses the ICMPv6 protocol with a unique message types to achieve the above function. In general terms, the IPv6 Neighbor Discovery protocol corresponds to a combination of the IPv4 protocols Address Resolution Protocol (ARP), ICMP Router Discovery (RDISC), and ICMP Redirect (ICMPv4), but with many improvements over these IPv4 protocols.

IPv6 defines both a stateful and stateless address autoconfiguration mechanism. Stateless autoconfiguration requires no manual configuration of hosts, minimal (if any) configuration of routers, and no additional servers. The stateless mechanism allows a host to generate its own addresses using a combination of locally available information and information advertised by routers. Routers advertise prefixes that identify the subnet(s) associated with a link, while hosts generate an interface-token that uniquely identifies an interface on a subnet. An address is formed by combining the two. In the absence of routers, a host can only generate link-local addresses. However, link-local addresses are sufficient for allowing communication among nodes attached to the same link.

7.1.3.1 NDP Application Kernel Support

For kernel function, a new version of the netinet kernel extension has been provided that contains code to handle both IPv4 and IPv6. New special processing handled by the kernel (or NDP applications) includes:

- Interface initialization
  - Autoconfiguration
  - Duplicate address detection (DAD)
  - Joining the all nodes multicast group
  - Sending router solicitations
  - Responding to router advertisements
- Interface maintenance
  - Managing prefixes (changing, if necessary, when router advertisements are received)
  - Timing out prefixes when advertised lifetime expires
  - Forming an appropriate address for received prefixes
  - DAD as necessary
- NDP support
  - Destination routing (NDP table)
  - Path MTU
  - Determination of neighbor unreachability (NUD)
- Source address selection
  - For multihomed hosts (problems with link-local)
- Routing
  - Uses version 4 routes for compatible addresses (through the Simple Internet Transition [SIT] interface)
  - Maintains multiple default routes, with fast switching based on NUD
- Tunneling
  - See section 7.1.5, “Tunneling over IPv4” on page 152 for more details.
- ICMPv6 - control messages
  - See section 7.1.4, “Internet Control Message Protocol (ICMPv6)” on page 151 for more details.
- Address mapping
  - Incoming v4 packets passed up to v6-aware applications through mapping the address to v6

Note: For more detailed information on this topic, refer to RFCs 1970 and 1971.

7.1.4 Internet Control Message Protocol (ICMPv6)

ICMPv6 is used by IPv6 nodes to report errors encountered in processing packets and to perform other internet-layer functions, such as diagnostics (ICMPv6 ping) and multicast membership reporting.

7.1.4.1 ICMPv6 Message Types

ICMPv6 messages are grouped into two classes:

- Error messages
- Informational messages

Error messages are identified by having a zero in the high-order bit of their message Type field values. Thus, error messages have message Types from 0 to 127. Table 28 lists the different types and their meanings.

Table 28. ICMPv6 Error Messages

Type	Description	Code	Cause
1	Destination unreachable	0	No route to destination
		1	Communication with destination administratively prohibited
		2	Not a neighbor
		3	Address unreachable
		4	Port unreachable
2	Packet too big	0	Packet too big
3	Time exceeded	0	Hop limit exceeded in transit
		1	Fragment reassembly time exceeded
4	Parameter problem	0	Erroneous header field encountered
		1	Unrecognized Next Header type encountered
		2	Unrecognized IPv6 option encountered

Informational messages have message Type values from 128 to 255. Table 28 lists the information types and their meanings.

Table 29. ICMPv6 Informational Messages

Type	Description
128	Echo request
129	Echo reply
130	Group membership query
131	Group membership report
132	Group membership reduction

Note: For more detailed information relating to message types and formats, refer to RFC 1885.

7.1.5 Tunneling over IPv4

The key to a successful IPv6 transition is compatibility with the existing installed base of IPv4 hosts and routers. Maintaining compatibility with IPv4, while deploying IPv6, streamlines the task of transitioning the Internet to IPv6.

In most deployment scenarios, the IPv6 routing infrastructure will be built-up over time. While the IPv6 infrastructure is being deployed, the existing IPv4 routing infrastructure can remain functional and can be used to carry IPv6 traffic.

Tunneling provides a way to use an existing IPv4 routing infrastructure to carry IPv6 traffic.

IPv6/IPv4 hosts and routers can tunnel IPv6 datagrams over regions of IPv4 routing topology by encapsulating them within IPv4 packets. Tunneling can be used in a variety of ways:

- Router-to-Router. IPv6/IPv4 routers interconnected by an IPv4 infrastructure can tunnel IPv6 packets between themselves. In this case, the tunnel spans one segment of the end-to-end path that the IPv6 packet takes.
- Host-to-Router. IPv6/IPv4 hosts can tunnel IPv6 packets to an intermediary IPv6/IPv4 router that is reachable through an IPv4 infrastructure. This type of tunnel spans the first segment of the packet's end-to-end path.
- Host-to-Host. IPv6/IPv4 hosts that are interconnected by an IPv4 infrastructure can tunnel IPv6 packets between themselves. In this case, the tunnel spans the entire end-to-end path that the packet takes.
- Router-to-Host. IPv6/IPv4 routers can tunnel IPv6 packets to their final destination IPv6/IPv4 host. This tunnel spans only the last segment of the end-to-end path.

Tunneling techniques are usually classified according to the mechanism by which the encapsulating node determines the address of the node at the end of the tunnel. In the first two tunneling methods listed above, router-to-router and host-to-router, the IPv6 packet is being tunneled to a router. In the last two tunneling methods, host-to-host and router-to-host, the IPv6 packet is tunneled all the way to its final destination.

7.1.5.1 Tunneling Mechanisms

The following is the path taken when tunneling:

- The entry node of the tunnel (the encapsulating node) creates an encapsulating IPv4 header and transmits the encapsulated packet.
- The exit node of the tunnel (the decapsulating node) receives the encapsulated packet, removes the IPv4 header, updates the IPv6 header, and processes the received IPv6 packet.
- The encapsulating node needs to maintain soft state information for each tunnel, such as the MTU of the tunnel, to process IPv6 packets forwarded into the tunnel.

Note: The information presented above is a brief summary of tunneling from RFC 1933. Refer to the RFC for more detailed information on this topic.

7.1.6 IP Security (IPSec)

IPSec (IP Security) is an IP-layer security mechanism for IP Version 4 (IPv4) and IP Version 6 (IPv6).

IPSec is a security protocol in the IP layer that provides security services to ensure packet authentication, integrity, access control, and confidentiality. A secure tunnel is established between the two systems to perform message encryption and message authentication.

There are two security mechanisms for IP. The first is the Authentication Header (AH) that provides integrity and authentication without encryption. The second is

the Encapsulating Security Payload (ESP) that always provides confidentiality and usually provides integrity and authentication. The protocol formats for the IP AH and IP ESP are independent of the cryptographic algorithm. The use of the RFC-compliant combinations of AH and ESP are supported in the initial release of IPSec for AIX 4.3.0.

IPSec also provides filtering capability without the use of secure tunnels. This may be very useful for setting up filters based on addresses, protocol, interface or port, and so on.

7.1.6.1 Key Management

Key management for AIX/IPSec supports a static key used during the lifetime of a tunnel and a dynamically refreshed session key that is updated periodically by a session key daemon using an IBM proprietary protocol. Master keys for both tunnel types can be manually input by the user or autogenerated using a pseudorandom number generator.

Note: Session key refresh is not supported for IPv6 tunnels.

IPSec for AIX allows for additional key management modules, including automatic master key exchange and key management user interface, as new protocols are defined. The current design does not include any distributed key management engines, such as Internet Security Association Management Protocol (ISAKMP/Oakley) or Simple Key Management for IP (SKIP).

In summary, the major key management elements in this release are:

- Manual master key exchange
- Static session key
- Dynamic session key refresh

Note: Master key access is restricted to root user only.

7.1.6.2 Transforms Provided with IPSec for AIX 4.3.0

The following transforms are provided with IPSec for AIX 4.3.0:

- keyed-md5
- hmac-md5 with _optional_ replay protection
- esp-des-cbc
- esp-des-md5 (with replay protection)
- hmac-sha with _optional_ replay protection

The design of IPSec for AIX allows plug-in and replaceable kernel modules for encryption and authentication.

7.1.6.3 Encapsulation Forms

The following list describes the encapsulation forms:

- IPSec for AIX supports both AH and ESP.
- On outgoing packets, the RFCs recommendations for AH and ESP order of processing is followed.
- On incoming packets, any order and combination of AH and ESP ordering is allowed.

7.1.6.4 Compatibility

Compatibility issues are described as follows:

- Current IP applications are not effected.
- IPSec concurrently interoperates with Internet Connection Secured Network Gateway (SNG) for AIX. Although both SNG and IPSec support IPv4, SNG code supersedes the IPSec code in providing IPv4 secure tunnel support. This means that if SNG is configured on a system, IPv4 will use the SNG tunnel code. Note that files belonging to IPSec fileset are unique, so both products can be installed without overwriting problems.

Note: IPv6 uses the IPSec code for secure tunnel support.

7.1.6.5 AIX/IPSec Kernel Configuration

The kernel configuration for AIX/IPSec comprises the configuration of all IPSec-related kernel extensions. The administrator controls the enablement of IPSec either through SMIT or `mkdev/rmdev` commands. Filter rules are downloaded and tunnels are activated during the configuration. The command to activate the tunnels also starts the session key daemon; therefore, filter rules must be generated before IP Security is loaded. On reboot, the kernel configuration is performed by the `cfgmgr` command.

7.1.6.6 IPSec/IPv4 Configuration

The system administrator has the option of enabling IPSec for either IPv4, IPv6, or both. Enabling IPSec for IPv4 loads all kernel extensions necessary to support IPSec for IPv4. The IPSec.v4 filter module configuration function assigns entry points to the IPv4 ip_fltr_*_hooks, and the IPSec encapsulation module configuration function assigns the ipsec_decap_hook. If IPSec.v4 detects that SNG is installed, it does not load.

The system call, sysconfig (SYS_QUERYLOAD,...), uses the SNG driver tuif.o as a search string to detect SNG kernel extensions. If SNG is installed after the IPSec.v4 is loaded, the SNG code changes the ip_*_hooks to point to SNG entry-points. IPSec.v4 checks to see if kernel extensions common to IPSec.v4 and IPSec.v6 have been loaded and only loads those that have not been loaded already. Once loaded, these modules are not unloaded from the kernel until next reboot. An ODM object in the CuDv, ipsec_v4 database represents the state of IPSec.v4 module. If the IPSec.v4 is defined, then the IPSec.v4 modules are loaded during boot time, and the ODM state changes to available.

7.1.6.7 IPSec/IPv6 Configuration

Enabling IPSec.v6 loads all kernel extensions necessary to support IPSec for IPv6. It checks to see if kernel extensions common to IPSec.v4 and IPSec.v6 have been loaded and only loads those that have not been loaded already. The statements concerning ODM in the preceding IPSec/IPv4 section also apply to IPSec.v6, with the exception that the CuDv object is called ipsec_v6.

7.1.6.8 Cryptographic Support

Cryptographic kernel extensions and supporting crypto-capsulation kernel extensions are selectively loaded based on the state of the ODM object that represents the cryptographic module. This process is similar to the one used for ODM object ipsec_v4, except it only affects the cryptographic modules. By default, the KEYED_MD5, HMAC_MD5, DES, and CDMF cryptographic kernel extensions are enabled. Additional cryptographic modules, as applicable, may be

loaded as well if the administrator has enabled them through SMIT or the cryptographic configuration method from the command line, as shown:

```
# mkdev -d -l des
```

7.1.6.9 IPSec Commands

The following administrative commands have been added. All of the commands require root access to run.

Table 30. IPSec Command Summary

Command	Description
gentun	Adds a tunnel entry
chtun	Changes a tunnel entry
rmtun	Removes a tunnel entry or deactivate a tunnel
lstun	List tunnels
exptun	Exports a tunnel entry
imptun	Imports a tunnel entry
mktun	Activates a tunnel entry
lsipsec	Shows IPSec status
genfilt	Adds a filter rule
chfilt	Changes a filter rule
mvfilt	Moves a filter rule
lsfilt	Lists a filter rule
rmfilt	Removes a filter rule
mkfilt	Activates or deactivates the filter rules
expfilt	Exports filter rule
impfilt	Imports filter rule

7.1.7 Resolver Support for /etc/hosts

IPv6 hosts entries in the /etc/hosts database follow the same standards used for IPv4-only. The hosts file may contain IPv4 and IPv6 addresses as shown in the following example:

An example of the entries in a typical /etc/hosts file:

```
127.0.0.1               loopback   localhost
129.144.248.127         percy
129.144.1.45            jumbuck
::8190:3584             redback
129.144.53.132          redback
::129.144.248.127       percy
fec0:1::11              ethspook
::ffff:129.144.248.127  taipan
fec0:1::                ethpercy
```

7.1.8 Commands and Applications Enabled for IPv6

The AIX IPv6 migration platform in AIX 4.3.0 has enabled a significant subset of TCP applications to support the IPv6 protocol. It is, however, important to note here that applications not supporting IPv6 in this first release will still function normally for IPv4 networks. Lack of IPv6 support simply means that the application has not been made IPv6 aware.

Some of the major applications that have not been made IPv6 aware are:

- DCE/DFS
- NFS/NIS
- HACMP
- AnyNet

The applications shown in Table 31 are able to run on either the IPv4 or IPv6 stacks. The code automatically determines the appropriate IP version to use.

Table 31. Applications Ported to IPv6

Command	Description
autoconf6	Automatically configures IPv6 addresses
crash/ndb	Network debugger updated for IPv6
ftp/ftpd	File transfer
ifconfig	Network interface configuration
inetd	Inetd server
iptrace/ipreport	Trace and dump network packets including IPv6
ndb	Display/update ND cache
ndpd-host	Support for receiving Router Advertisements
netstat	Network statistics
ping	ICMP echo/reply diagnostic
rcp	Remote copy
rexec/rexecd	Remote command execution
rlogin/rlogind	Remote login
route	Add/delete IPv4 and IPv6 routes
rsh/rshd	Remote shell access
tcpdump	Dumps Network packets, including IPv6
telnet/telnetd	Remote login
tftp/tftpd	File transfer

7.1.8.1 netstat

`netstat` displays the contents of various network-related data structures. There are the same number of output formats, but they now show additional IPv6 information. Typical output from the `netstat -ni` command is shown in Figure 33. Note the display of IPv6 addresses.

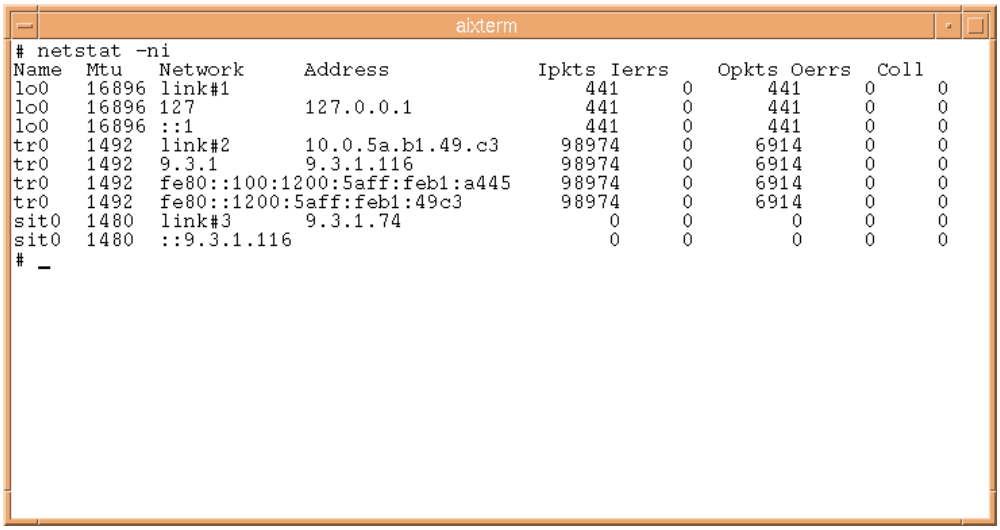


Figure 33. Typical Output from the netstat -ni Command

Typical output from the `netstat -rn` command is shown in Figure 34.

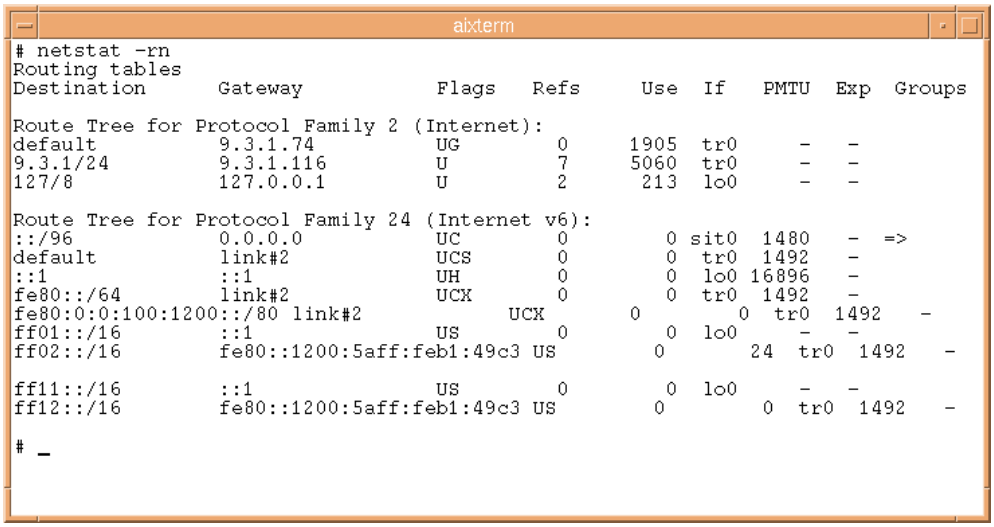


Figure 34. Routing Tables Shown by netstat -rn Command

Again, note the IPv6 style addresses and the use of the `::` (double colon).

For displaying IPv6 statistics, a new option has been added to `netstat`. By using the -p ipv6 option, `netstat` will display all the statistics associated with IPv6. Typical output from this command is shown in Figure 35.

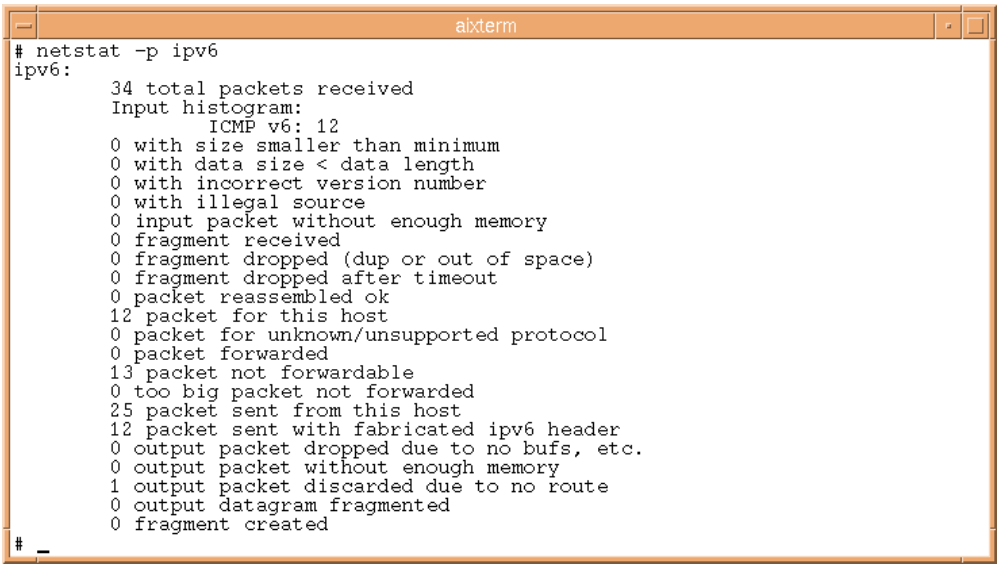


Figure 35. IPv6 Statistics from netstat -p ipv6 Command

7.1.8.2 ifconfig

`ifconfig` is used to assign an address to a network interface or configure network interface parameters. Address family `inet6` has been added to the list of `ifconfig` address family arguments. A new parameter called first can be set. This puts an IPv6 address at the first place on an interface to select it as the source for unbound sockets. Examples of the use of `ifconfig` are shown in Figure 36. This example shows the interface status, binding a site-local address to `tr0`, then making the site-local address the primary IPv6 address. Finally, the results of the actions are displayed.

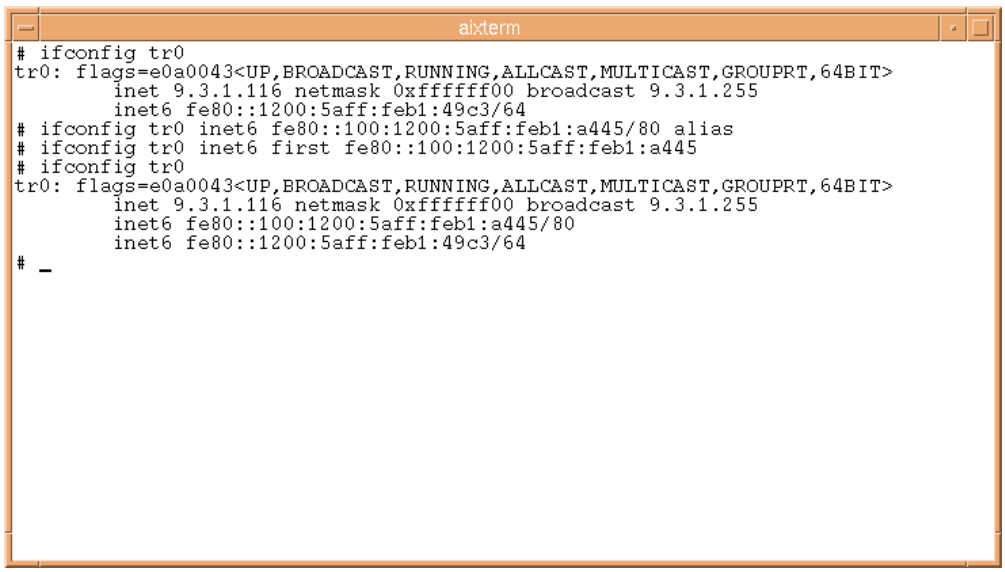


Figure 36. Example of ifconfig Command Usage

7.1.8.3 route

route was modified to process IPv6 addresses and structures. For the IPv6 address family, -inet6 must be used. The length of a destination prefix can be specified with the option -prefixlen or with the syntax prefix/length. There is also a new command to route called get that looks up and displays the route for a destination. See Figure 37 for route command usage examples.

Note: The get command works for all protocol families, not just IPv6.

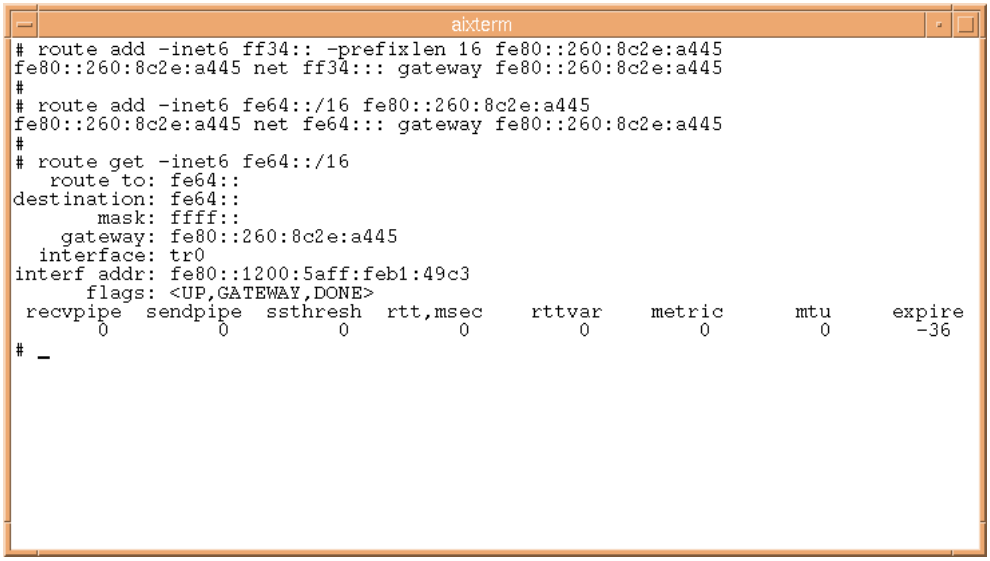


Figure 37. Example of route Command Usage

7.1.8.4 autoconf6 Command

autoconf6 is a new command with IPv6. The autoconf6 command is used to assign addresses to network interfaces and add some needed routes at boot time. It configures the link-local address, adds the appropriate interface route, and adds a compatibility address if an IPv4 address exists. autoconf6 should be used after configuration of IPv4 and before any IPv6 action. It can take, as an argument, the name of the main IEEE LAN interface.

Table 32. auotconf6 Options

Flag	Description
-v	Verbose output.
-6	Do not install the SIT tunnel interface.

The following shows an example of the typical output that is received when running the autoconf6 -v command:

```
got interface lo0
got interface en0
got interface sit0
default IEEE interface is en0
interface en0 is booting
setup IEEE interface en0
config en0 fe80::260:8c2e:a445/80
add route flags=801
        dest=ff02::
        gateway=fe80::260:8c2e:a445
        mask=ffff::
add route flags=801
```

```
        dest=ff12::
        gateway=fe80::260:8c2e:a445
        mask=ffff::
Sending ND sol for fe80::260:8c2e:a445 on en0
en0 setup good
interface en0 is booted
setup SIT interface sit0
config sit0 9.3.145.129
config sit0 ::903:9181/96
add route flags=901
        dest=::
        gateway=::903:9181
        mask=ffff:ffff:ffff:ffff:ffff:ffff::
setup loopback interface lo0
config lo0 ::1/128
add route flags=801
        dest=ff01::
        gateway=::1
        mask=ffff::
add route flags=801
        dest=ff11::
        gateway=::1
        mask=ffff::
add route flags=901
        dest=::
        gateway=fe80::260:8c2e:a445
        mask=::
setup2 IEEE interface en0
config en0 ::903:9181 alias
         with mask ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffc0
setup2 loopback interface lo0
config lo0 ::7f00:1/128 alias
```

7.1.8.5 TCP/IP Tracing Commands

The TCP/IP tracing commands `tcpdump`, `iptrace`, and `ipreport` have been modified to understand IPv6 addresses and data structures. Tracing IPv6 data flow is therefore as easy as IPv4.

7.1.8.6 traceroute Command

The `traceroute` command now takes advantage of the IPv6 protocol hop limit field and attempts to elicit an ICMPv6 TIME_EXCEEDED response from each gateway along the path to some host. The only mandatory parameter is the destination host name or IPv6 address. It infers the interface MTU at the beginning or uses the packet size argument. It also tries to get the path MTU using ICMPv6 PACKET_TOO_BIG messages. There are three new options with `traceroute`: -d, -f, and -g. These are described in Table 33.

Table 33. traceroute Options

Flag	Description
-d	Turn on socket-level debugging.
-f flow	Set the flow identifier in probe packets (default zero).
-g gateway_addr	Add gateway_addr to the list of addresses in the IPv6 Source Record Route header. If no gateways are specified, the SRR extra header is omitted.

7.1.8.7 ndp Command

The `ndp` command displays and modifies the IPv6-to-Ethernet address translation tables used by the IPv6 Neighbor Discovery Protocol. With no flags, the command displays the current NDP entry for hostname. The host may be

specified by name or by number using IPv6 textual notation. Table 34 shows these options.

Table 34. ndp Options

Flag	Description
-a	The program displays all of the current NDP entries.
-d	A superuser may delete an entry for the host called hostname with the -d flag.
-i interface_index	Specify the index of the interface to use when a NDP entry is added with the -s flag (useful with the local-link interface).
-n	Show network addresses as numbers (normally ndp attempts to display addresses symbolically).
-s <media_type> hostname hardware_addr	Hostname with the hardware address hardware_addr for the specified media type. The hardware address is given as six hex bytes separated by colons. The valid media types are: ether, 802.3, fddi, and 802.5. For 802.5 (token-ring), an optional src route can be appended to the hardware_addr. The src route is specified as a list of 16-bit hex numbers separated by colons. The entry is made permanent unless the word temp is given in the command.
-t <if type>	Invoke an interface-specific ndp command. The remaining syntax for the command is based on the interface-specific command.

Figure 38 shows use of the `ndp` command to first show all the current NDP entries and then delete the entry associated with `e-crankv6-11`.

Figure 38. Example of ndp Command Usage

7.1.8.8 ndpd-host Command

The `ndpd-host` command manages the NDP (Neighbor Discovery Protocol) for non-kernel activities, such as router discovery, prefix discovery, parameter discovery, and redirects. `ndpd-host` deals with the default route, including default

router, default interface, and default interface address. Table 35 provides the flags for the ndpd-host command.

Table 35. ndpd-host Options

Flag	Description
-d	Enables debugging (exceptional conditions and dump)
-v	Logs all interesting events (daemon.info and console)

7.1.8.9 inetd Daemon

The inetd daemon creates AF_INET sockets for services that are tcp4 and udp4 and AF_INET6 sockets for services that are tcp6 and udp6. If the -6 option is used, inetd creates AF_INET6 sockets for services that are TCP and UDP. Otherwise, it creates AF_INET sockets for these services. The new members of the servtab structure are se_family (address family) and se_ctrladdr_size (for the differences in size of sockaddr_in and sockaddr_in6). Also, the se_un member was modified to include the sockaddr_in6 structure. The config() function has an added AF_INET6 case.

7.1.9 IPv6 Socket Library Support

Socket library support has been modified to allow for application developers to begin porting to IPv6:

- IPv6 data structures (include files define these).
- socket(), bind() and connect() support the AF_INET6 address family and the sockaddr_in6 address structure.
- ASCII print function is provided with inet_pton() and inet_ntop().
- Address/name translations with IPv6 addresses.
- Interface/index information retrieval.
- Address manipulation/query (in6_ifXXX(), XXX_ADDR6()).
- IPv6-specific getsockopt() and setsockopt() calls.

7.1.10 System Management Changes and Additions

The recommended method of configuring IPv6 is through SMIT. New SMIT panels have been added and existing ones changed to achieve this. Existing system configuration commands, such as mkdev and chdev, can also be used but will not be described here.

The IPv6 configuration panels are reached through a new option added to the standard TCP/IP panel, as shown in Figure 39.

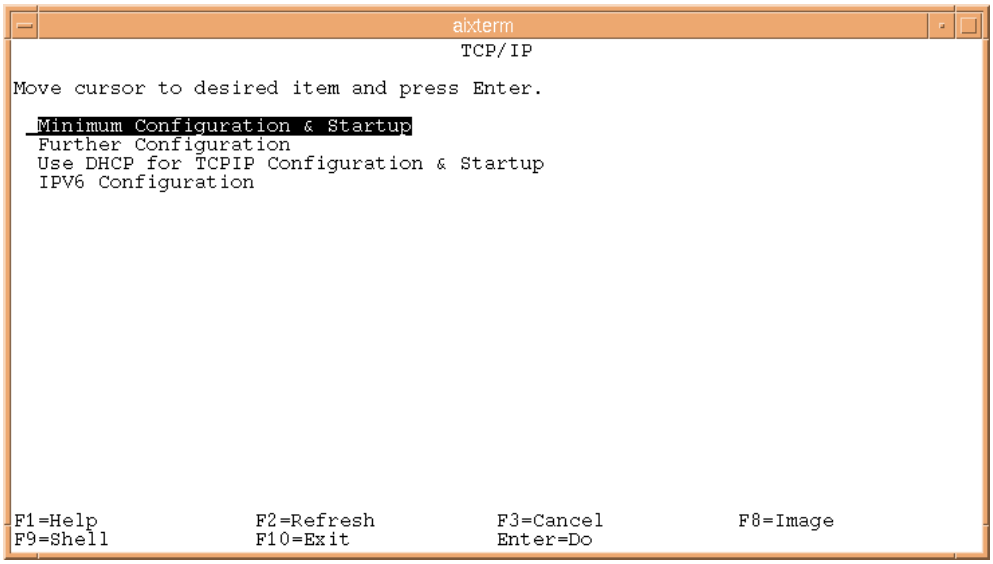


Figure 39. New SMIT TCP/IP Configuration Panel Entries

Many of the IPv6 SMIT panels are virtually identical, in both layout and function, to their IPv4 equivalents, so we will not give examples in this publication. One panel that does not exist in non-IPv6-enabled releases is the panel for configuration of tunnel interfaces. This is shown in Figure 40.

When adding an interface, both the IPv6 and IPv4 source and destination addresses must be specified.

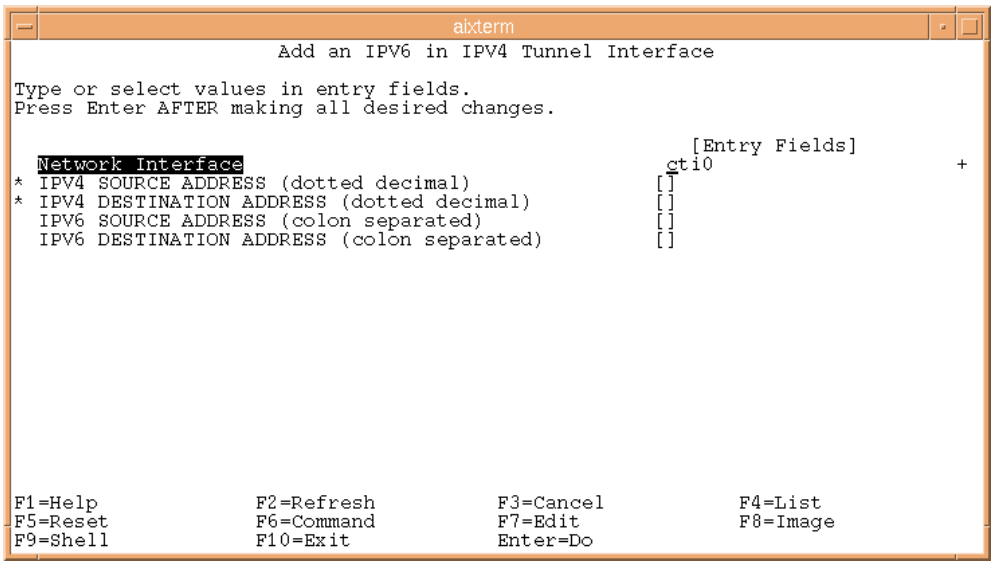


Figure 40. Configuring IPv6 Tunnel Interfaces with SMIT

This panel executes the chdev command that calls `ifconfig` and adds an entry to the ODM.

There is also a SMIT panel that allows configuration of the IPv6 daemons, shown in Figure 41. It is reached through the SMIT fastpath, `smit daemon6`.

This panel presents the user with options to configure the autoconf6 process, the ndpd-host subsystem, and also provides a link to the inetd subsystem configuration.

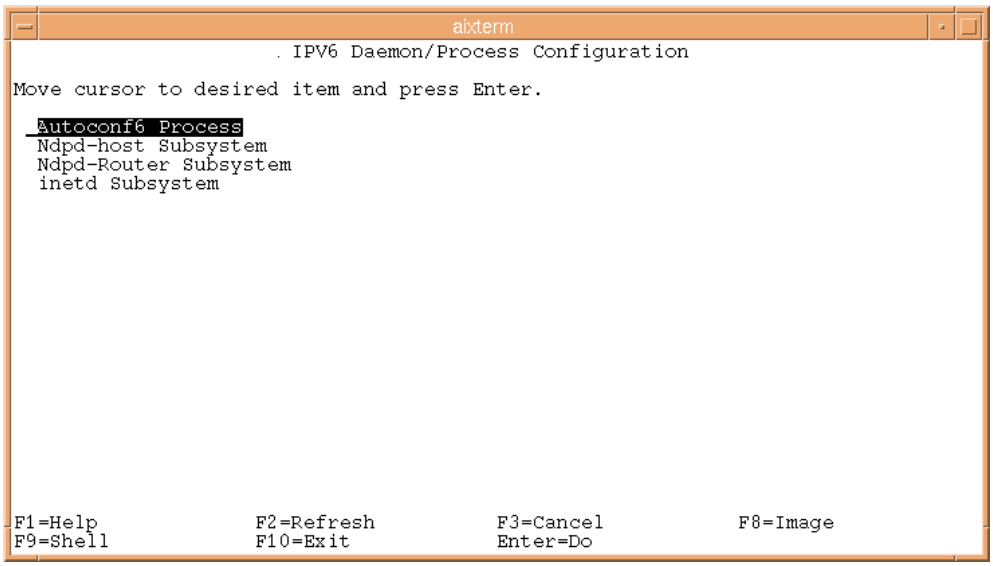


Figure 41. IPv6 Daemon Configuration SMIT Panel

If the **Change / Show** option is selected from the Autoconf6 SMIT configuration panel, the following panel is displayed. The VERBOSE parameter instructs the autoconf6 process to display what it is doing and also show any errors it is encountering. The SIT option allows the autoconf6 process to be started without the SIT interface and IPv4-compatible parameters being installed. Once you have made any changes you want to the default parameters and pressed the **Enter** key, SMIT uncomments the autoconf6 entry in the /etc/rc.tcpip file and starts the autoconf6 process.

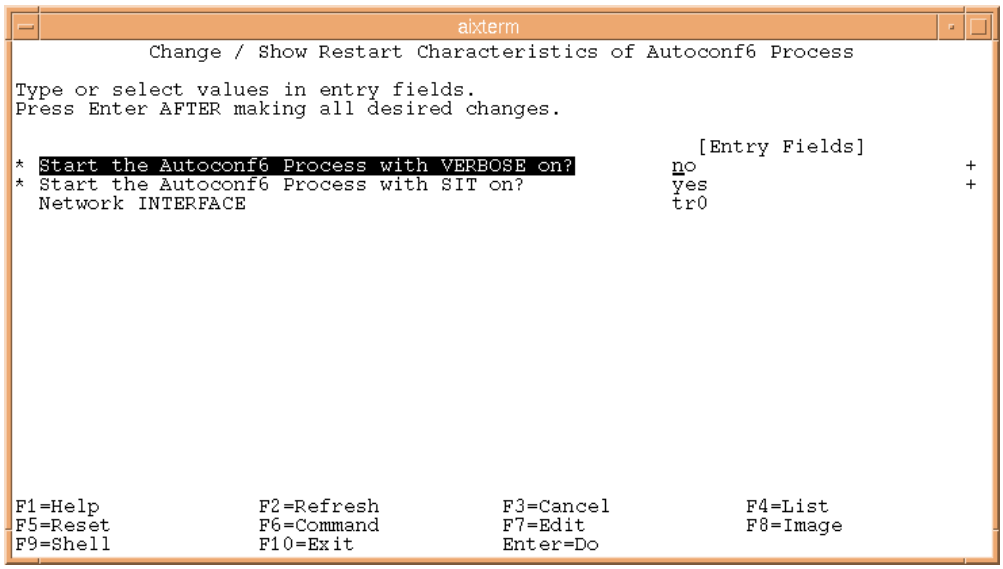


Figure 42. IPv6 autoconf6 SMIT Configuration Panel

7.1.11 IPv6 and IPSec-Related RFCs Implementation

Table 36 shows the RFCs that have been implemented in the IP Version 6 support included with AIX Version 4.3.0. Please consult these RFCs, or their successors, for the latest information on IP Version 6 protocols.

Table 36. RFCs Implemented in AIX Version 4.3.0

RFC number	RFC Title
1825	Security Architecture for the Internet Protocol
1826	IP Authentification Header (AH)
1827	IP Encapsulating Security Payload (ESP)
1828	IP Authentification Using Keyed MD5
1829	The ESP DES-CBC Transform
1883	Internet Protocol, Version 6 (IPv6) Specification
1884	IP Version 6 Addressing Architecture
1885	Internet Control Message Protocol (ICMPv6) for IPv6
1933	Transition Mechanisms for IPv6 Hosts and Routers
1970	IPv6 Stateless Address Autoconfiguration
1971	Neighbor Discovery for IP Version 6 (IPv6)

7.2 IP Security Enhancements (4.3.1)

AIX 4.3.1 has added new functions to IP Security:

- Triple-DES is an additional choice for U.S. and Canada customers.
- Performance improvements have been made to the CDMF and DES encryption.
- The filter table now supports unlimited number of rules.
- Any combination of installed authentication and encryption algorithms can be used for ESP with Authentication.

In addition, new parameters have been added to the following IPSec commands:

- `exptun`
- `imptun`
- `mktun`
- `rmfilt`

7.3 TCP/IP Command Security Enhancement (4.3.1)

AIX Version 4.3.1 offers secure remote TCP/IP commands: rsh, rcp, rlogin, telnet, and ftp. With this capability, Kerberos 5 authentication is used between these commands and server daemons, avoiding the need for user passwords to pass in the clear on the network. Instead, Kerberos 5 credentials are used to authenticate users. User credentials can be forwarded to the server.

7.4 Dynamic Host Configuration Protocol Enhancements (4.3.1)

A new command `dadmin` has been added to assist the DHCP administrator.

The `dadmin` command lets the DHCP administrator query and modify the state of his DHCP servers' databases. It gives the administrator the ability to locally or remotely query the DHCP server for the status of an IP address, query for a pool of IP addresses, query for a client, delete an IP address mapping, refresh the server, and change the server's tracing level. The `dadmin` command is backwards compatible with previous AIX release DHCP servers to list their IP address status and refresh.

When querying for IP address information, the `dadmin` command returns the IP address's status, and depending on this, may return the lease duration, start lease time, last leased time, whether the server supports DNS A record updates for this IP address, and the client identifier that is mapped to this IP address.

When querying for client information, the `dadmin` command returns the client's IP address and IP address status, whether the server supports DNS A record updates for this IP address, the last time the client was given any IP address, and the hostname and domain name used by the client.

When modifying the server tracing level, the `dadmin` command sets and returns the server tracing level in the form of a tracing mask. This mask represents a bit string, where each bit represents whether a specific log item is being traced by the server (see DHCP Server Configuration in the online documentation). From least significant to most significant order, these log items are LOG_NONE, LOG_SYSERR, LOG_OBJERR, LOG_PROTOCOL and LOG_PROTERR (same value), LOG_WARN and LOG_CONFIG (same value), LOG_EVENT and LOG_PARSEERR (same value), LOG_ACTION, LOG_INFM, LOG_ACNTING, LOG_STAT, LOG_TRACE, LOG_START, and LOG_RTRACE.

7.5 TFTP Block Size Option (4.3.1)

The implementation of TFTP in AIX Version 4.3.1 has been enhanced to include the Block Size Option proposed in RFC 1783. Older implementations of TFTP relied upon a fixed block size of 512 octets that although easy to code and implement in the limited ROM space available in some clients, proves to be very inefficient on LANs whose MTU size may be 1500 octets or more.

This implementation of TFTP has modified the TFTP Read Request and TFTP Write Request packets to include a block size option. When a client sends a read or write request, it can now include an option to request that the server uses a block size other than 512 octets. If the server is willing to accept the blocksize option, it responds with an Option Acknowledgment (OACK) containing the blocksize that must be equal to, or smaller than, that requested by the client. If the client is unable to accept the blocksize returned by the server, it must reply with an error packet and terminate the transfer.

Tests on the performance of TFTP with different blocksizes have revealed that file transfer time can be reduced by as much as 80 percent by using larger block sizes.

Benefits are only gained when a client is able to perform block size negotiation. At the time of writing, the IBM Network Station TFTP client has this function. This change does not affect the current AIX TFTP client application.

Note

If the blocksize exceeds the MTU of the path, then fragmenting will occur, and performance improvements will be reduced by the need to perform reassembly of packets.

7.6 IPv6 Routing Support (4.3.2)

The first release of IPv6 support in AIX 4.3 was a host implementation. It did not support routing between interfaces on the same server. Also, the routing applications were not well defined and not available in the INRIA code base to interpret IPv6 routing protocols.

AIX Version 4.3.2 supports IPv6 gateway capability in addition to IPv6 host supported in AIX Version 4.3. Version 6.0 of gated from INRIA is ported from INRIA to AIX 4.3.2 with IPV6 support so that gated and some of its routing protocols can manipulate IPv6 addresses. Some changes have been made to the AIX kernel, netinet kernel extension, and non-gated applications to support IPv6 routing.

7.6.1 Gated Version 6.0

The gated daemon provides gateway routing functions for the following protocols:

- Routing Information Protocol (RIP)
- Routing Information Protocol Next Generation (RIPng)
- Exterior Gateway Protocol (EGP)
- Border Gateway Protocol (BGP) and BGP4+
- Defense Communications Network Local-Network Protocol (HELLO)
- Open Shortest Path First (OSPF)
- Intermediate System to Intermediate System (IS-IS)
- Internet Control Message Protocol (ICMP) / Router Discovery routing protocols
- Simple Network Management Protocol (SNMP)

The gated process can be configured to perform all of these protocols or any combination of them.

Dynamic routing tables change according to the network conditions. They are built from information passed to gated by the routing protocols.

Each protocol performs the same basic functions, in that they determine the best route for a destination and then distribute the routing information to other systems on the network. Each protocol performs these tasks based on their own particular algorithms.

Gated collects the routing information from each protocol and then selects the best route per destination among them. Each routing protocol has its own way of calculating the cost of a route, that is, the metric value of a route. For example, RIP uses distance (hop count) and HELLO uses time (delay in milliseconds) to determine the cost of a route. Gated uses its own preference values to determine who has the best route. Preference values range from 0-255 with lower numbers indicating more preferred routes. Once gated has selected its best route to a destination, it puts that route in its routing table and later installs it into the kernel's routing table.

There are three new commands delivered along with the new gated supporting IPv6 routing. They are `gdc`, `ospf_monitor`, and `ripquery`.

Following are some brief descriptions of these new introduced commands:

7.6.1.1 gdc Command

The `gdc` command provides an operational user interface for gated. The interface is user-oriented for the operation of the gated routing daemon. It provides supports for:

- Starting and stopping gated daemon
- Delivering signals to manipulate the gated daemon
- Maintenance and syntax checking of configuration files
- Removal of state dumps and core dumps

The `gdc` command can reliably determine the running state of gated and produces a reliable exit status when errors occur, making it advantageous for use in shell scripts that manipulate gated.

7.6.1.2 ospf_monitor Command

The `ospf_monitor` command queries OSPF routers to provide the detailed statistics. It monitors the OSPF gateways.

7.6.1.3 ripquery Command

The `ripquery` command sends a RIP request or POLL command, to a RIP gateway to request all the routes known by the gateway. It queries the RIP gateways. This command is used as a tool for debugging gateways.

Gated has the following enhancements:

- SRC support is added to `gated`. You can start gated using SRC or `gdc`.
- Message catalog is added for `gated`, `ripquery`, `ospf_monitor`, and `gdc`.

Following IPv6 protocols are added in gated:

- IPv6 Routing Information Protocol (RIPNG)
- IPv6 Internet Control Message Protocol (ICMPv6)
- IPv6 Border Gateway Protocol (BGP4+)

7.6.2 IPv6 Routing Functions

The changes for IPv6 routing in Institut National de Recherche en Informatique et en Automatique (INRIA) are merged into the AIX netinet and kernel. Also, some

of the routing applications are ported to AIX, specifically, ndpd-router and updates to ndpd-host and ndp.

AIX 4.3.2 adds the following IPv6 functions:

- IPv6 unicast routing support
- IPv6 multicast routing support - no mrouted6 though
- IPv6 anycast address support - mostly receive processing
- IPv6 multi-homed link local and site local support. The reason for adding multi-homed host support is because most routers are multi-homed and need the ability to reference different hosts on different links at the link local and site local levels.

7.6.2.1 IPv6 Unicast Routing

A packet sent to a unicast address is delivered to the interface identified by that address.

There are two main aspects added in AIX 4.3.2: `ndpd-router` and the some netinet changes.

The first major function is `ndpd-router`. The `ndpd-router` command provides a protocol engine for NDP and RIPng. It provides a simple method to distribute routes between gateways. This function is redundantly provided by `gated`. The routing specific mechanisms, like router advertisements and answers to router solicitations are handled by ndpd-router. RIPng is the follow-on for RIP with regards to IPv6.

The other major function is to provide extensions to the icmpv6 support so that all the NDP messages are supported. The new messages are for handling redirects.

The modifications to the IP input path allows the forwarding of IPv6 packets between interfaces on the same machine. This function is identical to the IPv4 low-level routing function. The ip6forwarding function handles redirecting incoming packets.

7.6.2.2 IPv6 Multicast Routing

The use of Internet Protocol (IP) multicasting enables a message to be transmitted to a group of hosts, instead of having to address and send the message to each group member individually. The Class D Internet addressing is used for multicasting.

In general, there are two pieces to multicast routing. One piece is a method to determine where the multicast message should be sent, and the other is a method to dynamically change the place the packet goes based on network conditions. In IPv4, the `mrouted` daemon handles both functions with the help of IGMP. In IPv6, multicasting, and its related membership function, is all handled in icmpv6. Usually, this is done by maintaining a table similar to a routing table that is used to direct messages. In IPv4, this table is maintained in netinet and updated by `mrouted`. In IPv6, INRIA adds it as an additional routing table. This way the route command could be used to manipulate static multicast routes.

7.6.2.3 IPv6 Anycast Address Support

In AIX 4.3, the anycast is similar to INRIA. This method involved creating anycast addresses and associating them with a specific interface. This is acceptable because, on incoming packet reception, AIX would talk to all the addresses on the address lists. But to modify these addresses, a user would have to remember the associated interface. INRIA has moved anycast addresses on to a private list. The management is separate from the normal addresses. In fact, a command called any6 is added to manipulate these addresses.

7.6.2.4 IPv6 Multi-Homed Support

AIX 4.3.0 is supported for IPv6 and multi-homed global addresses. This falls out naturally from the IPv6 routing table. The problem is with site local and link local addresses. The problem with site local addresses is the definition of network boundaries. This is not handled by the kernel. The problem with link local addresses is which interface to go out of for doing the initial NDP neighbor solicitations. The code defines a #define, MULTI_HOMED, that can have four values. Each value has different actions and operations.

The AIX IPv6 will support the host requirements from the following RFCs:

- RFC 1883 - Internet Protocol, Version 6 (IPv6) Specification
- RFC 1884 - IP Version 6 Addressing Architecture
- RFC 1885 - Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6)
- RFC 1886 - DNS Extensions to support IP Version 6
- RFC 1887 - An Architecture for IPv6 Unicast Address Allocation
- RFC 1970 - Neighbor Discovery for IP Version 6 (IPv6)
- RFC 1971 - IPv6 Stateless Address Autoconfiguration
- RFC 1972 - A Method for the Transmission of IPv6 Packets over Ethernet networks
- RFC 1981 - Path MTU Discovery for IP Version 6
- RFC 2019 - A Method for the Transmission of IPv6 Packets over FDDI networks

For a complete list of all RFCs and Internet drafts pertaining to IPv6, refer to `http://www.ietf.org/html.charters/ipngwg-charter.html`. This is an evolving list, and AIX conforms to a subset of these.

7.6.3 Commands Changed

The following AIX commands have been changed to support IPv6 routing:

- `ndpd-host`
- `ndp`
- `route`
- `autoconf6`
- `hostnew`
- `nslookup`
- `ifconfig`

There are new no options added:

- ip6forwarding

  This option works like the existing ipforwarding only for IPv6 packets.

  The IP packets forwarding function will be enabled by the following command:

  ```
  #no -o ip6forwarding=1
  ```

  The value of ipforwarding specifies whether the kernel should forward packets. The default value of 0 prevents forwarding of IP packets. A value of 1 enables forwarding.

- multi_homed
- main_if6
- main_site6
- site6_index

7.7 Enhancement for ifconfig Command (4.3.2)

The new flags introduced to `ifconfig` command are:

```
ifconfig [ -m ] Interface [ ProtocolFamily ] Interface ProtocolFamily
ifconfig -a [ -m ] [ -d ] [ -u ] [ ProtocolFamily ]
```

Table 37 provides the descriptions of the new ifconfig flags:

Table 37. ifconfig New Flags for Display Interface Information

Flags	Descriptions
-a	The -a flag can be used instead of an interface name. This flag instructs `ifconfig` to display information about all interfaces in the system.
-d	The -d flag displays interfaces that are down.
-m	If the -m flag is passed before an interface name, `ifconfig` will display all of the supported media for the specified interface.
-u	The -u flag displays interfaces that are up.

7.8 Latest BIND DNS (NameD) Support (4.3.2)

The Domain Naming System (DNS) is a method to distribute a large database of IP addresses, hostnames, and other record data across administrative areas. The end result is a distributed database maintained in sections by authorized administrators per domain.

AIX, similar to other UNIX platforms, offers the BIND DNS server. BIND, or the Berkeley Internet Name Daemon, is a DNS server implementation provided by the Internet Software Consortium. It has become the standard for DNS server implementations and a benchmark for DNS server intercompatibility.

AIX 4.3.2 incorporates IBM DNS value-added functions to the latest level of BIND, Version 8.1.2. This involves adding IBM secure dynamic DNS update protocol and incremental zone transfers to BIND 8.1.2, as well as extending this BIND's NOTIFY ability and parameter configuration.

The following are the new functions provided Bind Version 8.1.2:

- Secure dynamic DNS updates

  Currently, BIND offers only the unsecured RFC 2136 update protocol. This is an insufficient offering to customers desiring to implement a dynamic DNS environment in their networks. The secure update protocol is added as the solution for RFC 2136's insecure shortcomings and to provide backward compatibility with current AIX dynamic DNS customers.

- Incremental zone transfer

  Implement the RFC 1995 Incremental Zone Transfer protocol. This protocol defines a method through which secondary DNS servers can update their existing zone data to incorporate all the cumulative changes to the primary zone since the last transfer. This protocol supersedes the performance of ordinary zone transfers by limiting the amount of network traffic between primary and secondary DNS servers and the subsequent computation time in incorporating an entirely new zone. The protocol ensures that incremental zone transfers can be sent to indicate changes from both dynamic updates and from zones changed on disk (those reincorporated through a refresh signal or server restart).

- Notify

  Implementing the RFC 1996 Notify process. This is a method by which the primary DNS server can indicate to its secondary nameservers that zone data has been updated. This decreases the time periods in which a secondary DNS server will have data out of synchronization with its primary DNS server.

- File Conversion Utility

  Extending the configuration file conversion utility to support IBM functional additions to previous BIND releases. This involves mapping the dynamic keywords of previous named.boot files to a functional equivalent in the named.conf configuration file.

- Proprietary protocol for secure updates of dynamic notify dynamic zones.

  Bind 4.9.3 is available using named4. Bind 8.1.2 uses named8.

7.9 Web Server Performance Improved (4.3.2)

Web server workload has a large number of small file operations. In smaller file operations, the cost of processing connection set-up/tear-down far exceeds the data touching (such as copy or checksum) operations. By reducing the number of packets that are sent or received for connection set-up/tear down, performance is improved. IBM has developed a set of TCP changes to reduce the number of packets from 9 to 7 or 6.

For Web transaction oriented environments using HTTP protocols, AIX Version 4.3.2 reduces the number of network packets that are exchanged between the workstation and the server. This lowers the CPU packet processing overhead, increasing server performance and capacity.

7.9.1 Reducing the Number of TCP Packages

There are some modifications to the TCP protocol implementation in AIX 4.3.2 that improves the performances of HTTP-like transactions on Web servers and clients.

A performance improvement for Web servers and clients can be achieved by reducing the total number of TCP packets that are exchanged, without violating the TCP protocol. For Web servers and clients, this can be done by delaying certain ACK messages and piggybacking them with the next packet that will be sent and also by sending the FIN message along with the last data packet.

A typical HTTP transaction requires the exchange of nine TCP packets:

1. The client sends a SYN TCP packet to connect to the server.
2. The server acknowledges the client's SYN and sends a SYN to accept the connection.
3. The client acknowledges the server's SYN.
4. The client sends its HTTP request in a data packet.
5. The server sends a data packet containing the acknowledgment and the answer to the client's request.
6. The server shuts down its side of the connection by sending a FIN packet.
7. The client acknowledges the server's FIN
8. The client shuts down its side of the connection by sending a FIN packet.
9. The server acknowledges the client's FIN.

In AIX 4.3.2, the TCP exchange packets are reduced from nine to seven by using no command and to six by using send_file() system call with the SF_CLOSE flag.

7.9.2 Commands Affected

Reducing the TCP package number can be set by the user from the command line or by an application on a per process basis.

Two new options, delayack and delayackports, have been added into the no command.

delayack

The delayack options delays ACKs for certain TCP packets and attempts to piggyback them with the next packet sent instead. It enables the user to specify whether the delay will be performed for the SYN ACKs, the FIN ACKs, or both. This will only be performed for connections whose destination port is specified in the list of the delayackports attribute.

The delayack option may have one of the following four values:

0 No delay, This is the default value.

1 Delay the ACK for the SYN only

2 Delay the ACK for the FIN only

3 Delay the ACK for both.

Setting delayack to any other value will result in a failure with EINVAL error code.

Following are some examples of using delayack:

```
# no -o delayack=3
```

sets delay the ACK for both SYN and FIN.

```
# no -d delayack
```

sets delayack to the default value 0.

delayackports

The delayackports option allows a user to specify a list of up to ten ports for which the ACKs for SYNs and/or FINs defined by the delayack port option will be performed.

This allows a system administrator to configure this operation for specific applications that use a specific port. Although this will improve the performance for Web servers, this may not be good for other applications.

The delayackports option takes a list of ports numbers separated by commas and enclosed in curly braces.

For example: `no -o delayackports={80,1080,1180}`

An empty list, `no -o delayackports={}`, will clear the ports.

`# no -d delayackports`, sets delayackports to the default. "{}" means no ports.

The send_file() system call with the SF_CLOSE flag will send the FIN packet with the last data.

7.9.3 Reducing the Contention of INIFADDR and Route Lock

There are some changes in kernel to the route lock and inifaddr lock in order to improve performance. These changes decrease the processing time for each incoming and outgoing packet. These changes are internal and do not affect external user interfaces.

The route lock and inifaddr are used by the TCP/IP protocol packets:

INIFADDR lock examined by every incoming packet

Route lock examined by every outgoing packet

7.9.3.1 Reducing the Contention of INIFADDR Lock

Currently, for every incoming packet, the inifaddr list must be searched with the lock held. This causes a great deal of lock contention.

The INIFADDR lock contention is serious for SMP machines. This lock is used for all input packets, causing unacceptable lock contention on machines with multiple adapters. Therefore, it is a critical lock on 12-way RS/6000 S70 machine with multiple adapters.

Since this inifaddr list is changed very rarely, but searched so often, the lock contention is reduced by:

- Allowing concurrent searches

- New macros to allow for simple read-write locking, and the INIFADDR lock macros are changed to use them.

7.9.3.2 Reducing the Contention of Route Lock

Currently, all outgoing IP packets must take the single route lock. On an SMP machine with multiple adapters, a large number of packets must take this lock, so it is the hot (critical) lock on output. Reducing the contention on this lock can increase the throughput on SMP systems with multiple adapters.

There is a global route lock that must be taken any time any route is changed or used. Each connection keeps a pointer to a cached route, but it still must take the route lock to check the validity of its cached route. So, the route lock must be taken for every outgoing packet.

The new design eliminates the requirement to take the global route lock for outgoing packets in the case where the cached route is valid. This should improve performance for TCP, where the cached route will usually be valid so the global lock will not be needed on packet output. For UDP, the cached route often is not valid, and so the global lock will still be needed for output. But since the contention on this lock from other code will be reduced, there is also a performance improvement for UDP.

The global route lock that protects the structure of the routing table has also been changed to use the same read/write lock in places where the global lock is still needed.

7.10 TCP Checksum Offload on ATM 155 Mbps PCI Adapter (4.3.2)

The PCI ATM 155 adapter has a hardware TCP/UDP checksum capability. In AIX 4.3.2, the ATM network device driver is modified to use this feature.

In the new ATM 155 adapter device driver, the workload of TCP data checksum processing is offloaded from the AIX TCP/IP protocol stack to the adapter itself. A related enhancement automatically remembers the mapping of virtual addresses to physical addresses for the entire networking buffer pool to save address translation during networking I/O operations.

TCP checksum offload for ATM reduces the amount of time spent on the main CPU computing checksums, thereby reducing latency and allowing the system to handle more work, in particular, to handle more packets in the same amount of time. This results in performance improvements.

The SpecWeb96 benchmark is published on the 7017-S70 using ATM interfaces. It benefits from this checksum offload capability.

TCP and the other network layers continue to perform as they do today, particularly on adapters that do not support checksum offload. All TCP functions are the same.

In order to improve performance for TCP over ATM, small packets do not have their checksum processing offloaded since setting up the offload will probably use as much processor time as computing the checksum. There is no gain, and may actually be a loss, in offloading checksum processing for these small packets.

There is a packet size threshold where it is cheaper to do the checksum processing completely on the CPU. The driver only offloads checksums for packets larger than this threshold. This threshold is maintained internally as a variable or constant.

Currently, AIX checksums transmitted packets before copying them to the network adapter, and checksum received packets after they have been copied from the adapter to processor memory. Checksumming packets on the adapter increases the possibility of undetected corruption during the copy from or to the processor. This possibility is very small. The NDD provides a way for you to turn off the checksum offload capability for an adapter in cases where corruption is believed to be occurring.

7.10.1 Limitations

The feature of checksum offload only applies to TCP on ATM over IP Version 4. It means:

- This driver does not offload checksum processing for IPv6 TCP connections,
- This driver does not offload checksum processing for UDP datagrams.
- The checksum processing is not allowed when IPSEC is running since IPSEC may encrypt data, and the checksum for the data should be done before it is encrypted. Offloading, in this case, will cause the checksum to be calculated after the data has been encrypted, which is wrong. When IPSEC is running, the checksum must be calculated before the data is encrypted.

7.10.2 Command Changes

The command `ifconfig` is changed to handle a new interface option, checksum_offload. Table 38 lists the new options.

Table 38. Ifconfig New Options for Checksum Offload

checksum_offload	Enables checksum offload if the adapter associated with the interface supports it. This is the default for adapters that support checksum offload.
-checksum_offload	Disables checksum offload. This is the default for adapters that do not support checksum offload.

7.11 Thread-Based Application Connection Enhancement (4.3.2)

The thundering herd problem is a well-known, industry-wide performance problem that effects some Web servers. When multiple threads call accept() function on the same socket, all of these threads are woken up for a single new connection. For example, if there are 256 threads sleeping in accept(), only one thread needs to be woken up to handle the new connection. However, currently all 256 threads are woken up. Since only one thread receives the connection, the other 255 immediately go back to sleep. This results in wasted CPU time due to the unnecessary scheduling, running, and sleeping of the 255 threads.

The cause for this problem is that the socket layer uses a hash table to keep chains of threads waiting for events on specific sockets. All threads waiting for an event on a specific socket will hash to the same slot, but other threads waiting for events on other sockets could also hash to the same slot. For this reason, all the threads in the slot must be woken up when the event occurs, so that the right

thread will have a chance to handle the event. The other threads will simply go back to sleep.

In AIX Version 4.3.2, only a single thread is awakened, thus reducing CPU overhead. And the system now is able to handle higher loads of socket connections. Industry multi-threaded network server application developers will appreciate that AIX Version 4.3.2 overcomes the thundering herd problem of wasted CPU cycles resulting from waking up multiple threads of a network server when new connections arrive.

7.12 IBM 10/100 Mbps PCI Ethernet Adapter Device Driver (4.3.2)

In previous AIX version, the Ethernet device driver copies data from mbufs to its own buffers. Recent prototypes have shown that transmitting data directly from mbufs would lead to better performance.

AIX 4.3.2 provides a new device driver for IBM 10/100 Mbps PCI Ethernet adapter. The new device driver reduces data copy operations from network buffers to its private buffers. The network performance in AIX 4.3.2 benefits from enhancements in this new PCI network device drivers. These enhancements reduce CPU demand, thus increasing overall server capacity.

The device driver supports the PCI 10/100 Ethernet adapter for AIX Versions 4.2.1 and 4.3.0. The new PCI 10/100 Ethernet is modified to support AUI and BNC ports starting at AIX 4.3.2 and is backwards compatible. The performance enhancements for reducing the data copies in AIX 4.3.2 is backwards compatible to AIX 4.2.

The device driver is conform to the Common Data Link Interface (CDLI) interface specifications.

This adapter now supports the twisted pair, AUI and BNC ports.

The speeds supported are:

- 10 (10 Mbps, half-duplex)
- 20 (10 Mbps, full-duplex)
- 100 (100Mbps, half-duplex)
- 200 (100Mbps, full-duplex)
- Auto-negotiate.

7.12.1 Packaging

The PCI 10/100 Ethernet adapter software package includes:

devices.pci.23100020.rte

devices.pci.23100020.diag

7.12.2 Configuration Parameters

The following parameters are user configurable:

Transmit Queue Size (tx_que_size)

The device driver supports a user-configurable transmit queue. This is the queue the adapter uses (not an extension of the adapter's queue). It is configurable among the values of 16, 32, 64, 128, and 256, with a default of 256.

Because of the configurable size of the adapter's hardware queue, the driver does not support a software queue.

Receive Queue Size (rx_que_size)

The device driver supports a user-configurable receive queue. This is the queue the adapter uses (not an extention of the adapter's queue). It is configurable among the values of 16, 32, 64, 128, and 256, with a default of 256.

Receive Buffer Pool Size (rxbuf_pool_size)

The device driver supports a user-configurable receive buffer pool size. The buffer is the number of pre-allocated mbufs for receiving packets. The minimum size of the buffer is the receive queue size, and the maximum is 2 KB, with a default value of 384.

Media Speed (media_speed)

The device driver supports speeds of 10 (10 Mbps, half-duplex), 20 (10 Mbps, full-duplex), 100 (100 Mbps, half-duplex), 200 (100 Mbps, full-duplex), and auto-negotiate on twisted pair. On the AUI port the device driver will support speeds of 10 (10 Mbps, half-duplex) and 20 (10 Mbps, full-duplex). The BNC port will only support 10 (10 Mbps, half-duplex). This attribute is user-configurable, with a default of auto-negotiate on twisted pair.

Enable Alternate Address (use_alt_addr)

The device driver supports a configuration option to turn on and off use of an alternate network address. The values are yes and no, with a default of no. When this value is set to yes, the alt_addr parameter will define the address.

Alternate Network Address (alt_addr)

For the network address, the device driver accepts the adapter's hardware address or a configured alternate network address. When the use_alt_addr configuration option is set to yes, this alternate address is used. Any valid Individual Address can be used, but a multicast address can not be defined as a network address.

Inter-Packet Gap (ip_gap)

The inter-packet gap bit rate setting controls the aggressiveness of the adapter on the network. A smaller number will increase the aggressiveness of the adapter, while a larger number will decrease the aggressiveness (and increase the fairness) of the adapter. If the adapter statistics show a large number of collisions and deferrals, this number should be increased. Valid values range from 96 to 252, in increments of 4. The default value of 96 results in IPG of 9.6 microseconds for 10 Mb and 0.96 microseconds for 100 Mb media speeds. Each

unit of bit rate introduces an IPG of 100 ns at 10 Mb and 10 ns at 100 Mb media speed.

7.12.3 Trace

The device driver sets up three trace hook IDs (defined in /usr/include/sys/cdli_entuser.phxent.h). The first hook traces the transmit path, the second trace the receive path, and the third will trace everything else.

The device driver has three trace points: transmit, receive, and other, and has a trace table of at least 1000 entries when complied normally. It traces entry/exit from transmit/receive routines at the interrupt and user layers, as well as on entry to the interrupt handler when compiled normally.

Table 39 lists these hook IDs.

Table 39. Hook IDs of 10/100 Ethernet PCI Adapter

Trace	Trace hook ID
Transmit	0x2E6
Receive	0x2E7
Other	0x2e8

7.12.4 Error Logging

The device driver logs errors in the error log whenever one of the following errors occur: hardware failures, DMA operations, and memory faults.

Following are the error log entries returned by the 10/100 Mbps PCI Ethernet device driver:

ERRID_PHXENT_ADAP_ERR

Indicates that the adapter is not responding to initialization commands. User-intervention is necessary to fix the problem.

ERRID_PHXENT_ERR_RCVRY

Indicates that the adapter hit a temporary error requiring that it enter network recovery mode. It has reset the adapter in an attempt to fix the problem.

ERRID_PHXENT_TX_ERR

Indicates that the device driver has detected a transmission error. User-intervention is not required unless the problem persists.

ERRID_PHXENT_PIO

Indicates the device driver has detected a program I/O error. The driver was unable to fix the problem. User-intervention is required.

ERRID_PHXENT_DOWN

Indicates that the device has been shutdown due to an irrecoverable error. The device is no longer functional due to the error. The irrecoverable error that

caused the device to be shutdown is logged immediately before this error log entry. User-intervention is required to bring the device back online.

ERRID_PHXENT_EEPROM_ERR

Indicates that the device driver has detected an error in the EEPROM of the device. The driver will not become available. Hardware support should be contacted.

7.13 SDLC/BSC Support for 4-Port PCI Adapter (4.3.2)

This enhancement enables the existing AIX SLDC and BSC (Binary Synchronous Communications) protocols to run on the 4-port PCI adapter IBM ARTIC960Hx. It provides a high performance multiprotocol WAN connectivity on PCI based machines. Both SDLC and Bisync can run on the same adapter on a per port basis.

BSC is required for banks that have Automated Teller Machines (ATM) and want to upgrade to latest RS/6000 models. Newer systems like the 7017-S70 no longer have ISA slots.

The 4-port PCI Adapter has up to four leased lines connectivity. Each port can communicate at 2.0 Mbps. It replaces the ISA adapter (#2701). The adapter is implemented for use in systems that support 64 bit wide PCI local bus operating at 33 or 66 MHz.

4-Port Multi-Interface PMC is the daughter card connected to a 4-port PCI adapter and provides the hardware to support up to four network links. For each port of the adapter, the physical interface of the attached cable is auto selected. There are four connection ports on the adapter. Each function as individual network devices simultaneously. The user can configure which physical interface (V.24, V.35, V.36 or X.21) that should be used per port or the adapter can autoselect the physical interface based on the cable connected.

Installation of AIX using this adapter is also be supported. The code to support this is part of the system firmware. The AIX product device driver will run on the adapter after control has been passed to AIX.

7.13.1 Packaging

SDLC and Bisync are both shipped with base AIX.

Following PTFs are needed for using the SDLC and Bisync protocol:

- AIX 4.3.2 with IX81860
- AIX 4.2.1 with IX81861

7.13.2 Trace

The device driver will issue trace points for the following conditions:

- All error conditions
- At the beginning and end of each main function in the main path
- At the beginning and end of each function that is tracking buffers outside of the main path

- Debugging purposes. These trace points are only enabled when the driver is compiled with -DDEBUG turned on, and therefore, the driver can contain as many of these trace points as desired.

The main goals for having trace points in a CDLI device driver are to be able to monitor drivers for errors and to track packets as they move through the driver. To this end, it is important that trace points be placed at the beginning and end of each main routine that does processing on packets. Also, it is important to try and trace the flow of buffers (for example, mbufs) as they flow through the system. There should also be trace points placed at each point where an error could occur.

The device driver also has trace points to support the netpmon program (component cmdperft). There are generally five trace points:

WQUE An output packet has been queued for transmission.

WEND The output of a packet is complete.

RDAT An input packet has been received by the device driver.

RNOT An input packet has been given to the demuxer.

REND The demuxer has returned.

All CDLI drivers must register for a trace hook IDs.

NOTE: CDLI device drivers may register for more than one trace hook ID. In the case of multiple trace hook IDs, one could be used for transmit, one for receive, and another for errors.

7.13.3 Error Logging

The device driver will do error logging when the following situations occur:

- An error on PIO operations
- A hardware failure on the device
- An error on DMA operations
- Network errors
- Other device specific errors

7.14 Open Network Computing (ONC+)

The ONC+ technology has been licensed from SunSoft and is being included within AIX to meet customer requirements. This technology contains many different functional components; the main ones being: NFS Version 3, NIS+, CacheFS, TIRPC, and AutoFS. Not all of these components are included with this release of AIX. NFS V3 was introduced in AIX Version 4.2.1. CacheFS was introduced in AIX Version 4.3.0, and AutoFS was included in AIX 4.3.1.

7.14.1 CacheFS

CacheFS is a local disk cache mechanism for NFS clients. It provides the ability for an NFS client to cache file system data on its local disk, thereby avoiding use of the network and NFS server when the data is accessed and is not in physical memory. This improves NFS server performance and scalability by reducing

server and network load. Designed as a layered file system, CacheFS provides the ability to cache one file system on another. In an NFS environment, CacheFS increases the clients per server ratio, reduces server and network loads, and improves performance for clients particularly on slow links.

CacheFS is contained in the bos.net.cachefs fileset, which is not automatically installed when installing AIX.

7.14.1.1 How CacheFS Works

After creating a CacheFS file system on a client system, the system administrator specifies which file systems are to be mounted in the cache. When a user on that client attempts to access files that are part of the back file system, those files are placed in the cache. Note that the cache does not get filled until a user requests access to a file or files. Therefore, the initial request to access a file will be at normal NFS speeds, but subsequent accesses to the same file will be at local JFS file system speeds. Refer to Figure 43 to see the relationship of the components in CacheFS.

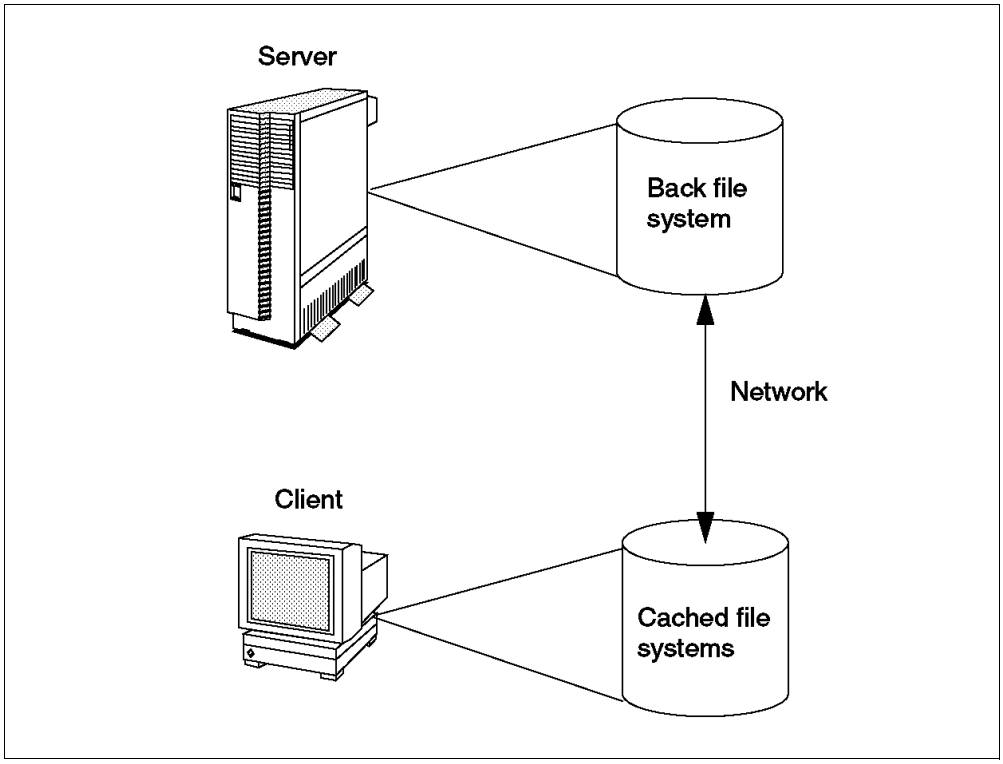


Figure 43. CacheFS Components

7.14.1.2 Configuring CacheFS

There are two steps involved in setting up a cached file system:

1. Create the local cache.
2. Specify and mount the file systems to cached.

Creating the Local Cache File System

A local cache file system is created by using the `cfsadmin` command with the `-c` flag, as shown in the following steps:

1. Login as root.
2. Create a cache using the `-c` flag of the `cfsadmin` command

   ```
   # cfsadmin -c -o <parameters> <cache-directory>
   ```

   where `parameters` specify resource parameters, and `cache-directory` is the name of the directory where the cache should be created. A list of configurable parameters is shown in Table 40 on page 185. Although the cache is referred to as a *cache filesystem*, it is not a filesystem in the true sense. It is, in fact, a cache directory, which resides on a normal JFS. For this reason, if you are creating a large cache filesystem, it is advisable to create a dedicated JFS to be used for this purpose. This is because the cache filesystem is created with parameters that indicate the percentage of the underlying filesystem it is allowed to use.
3. Mount the back file system in the cache

   ```
   # mount -V cachefs -o backfstype=nfs,cachedir=/<cache-directory> \
     remhost:/<remote-directory> <local-mount-point>
   ```

   where `remhost:/<remote-directory>` is the name of the remote host and file system where the data resides, and `<local-mount-point>` is the mount point on the client where the remote file system should be mounted.

CacheFS can also be administered using SMIT.

Cached File Systems Consistency Checking

To ensure that the cached directories and files are kept up to date, CacheFS periodically checks consistency of files stored in the cache. To check consistency, CacheFS compares the current modification time to the previous modification time. If the modification times are different, all data and attributes for the directory or file, are purged from the cache, and new data and attributes are retrieved from the back file system.

When a user requests an operation on a directory or file, CacheFS checks if it is time to verify consistency. If so, CacheFS obtains the modification time from the back file system and performs the comparison.

Note: It is important to remember that CacheFS is intended to be used as a mechanism for reducing network and server workload. Because the remote file system data is held locally on the client, and consistency is only checked at intervals, it means that the data on the server can change, and the client is unaware of this for a period of time. You should, therefore, only use it for read-only, or read-mostly, file systems where the file system contents do not change rapidly.

One example where CacheFS would be suitable is in a CAD environment where master-copies of drawing components can be held on the server and cached-copies kept on the client workstation when in use.

7.14.1.3 CacheFS Commands

New commands specific to CacheFS are:

- `cfsadmin`
- `fsck_cachefs`

cfsadmin

The `cfsadmin` command provides for the following CacheFS administration functions:

- Creating cached file systems
- Deleting cached file systems
- Listing cache contents and statistics
- Changing CacheFS resource parameters

The `cfsadmin` command syntax is:

```
cfsadmin -c [ -o cachefs-parameters ] cache_directory
cfsadmin -d [ cache_id | all ] cache_directory
cfsadmin -l cache_directory
cfsadmin -s [ mount point | all ] cache_directory
cfsadmin -u [ -o cachefs-parameters ]
```

Table 40 details the `cfsadmin` options.

Table 40. cfsadmin Options

Flag	Description
-c	Creates a cache under the directory specified by `cache_directory`. The directory must not exist prior to cache creation.
-d	Removes the file system specified by `cache_id` and releases its resources or removes all file systems if `all` is specified.
-l	Lists file systems stored in the specified cache with their statistics.
-s	Requests a consistency check on the specified file system, or all file systems, if `all` is specified.
-u	Updates resource parameters of the specified cache directory. Note that parameter values can only be increased, decreasing requires cache removal and recreation.
-o	CacheFS resource parameters see Table Table 41 for details.

Table 41 details CacheFS resource parameters.

Table 41. CacheFS Resource Parameters

Parameter	Description
maxblocks=n	Maximum amount of storage space that CacheFS can use, expressed as a percentage of the total number of blocks in the front file system. Default=90%
minblocks=n	Minimum amount of storage space that CacheFS is always allowed to use, expressed as a percentage of the total number of blocks in the front file system. Default=0%

Parameter	Description
threshblocks=n	A percentage of the total blocks in the front file system beyond which, CacheFS cannot claim resources once its block usage has reached the minblocks level. Default=85%
maxfiles=n	Maximum number of files that CacheFS can use, expressed as a percentage of the total number of inodes in the front file system. Default=90%
minfiles=n	Minimum number of files that CacheFS is always allowed to use, expressed as a percentage of the total number of inodes in the front file system. Default=0%
maxfilesize=n	Largest file size, expressed in megabytes, that CacheFS is allowed to cache. Default=3

fsck_cachefs

The `fsck_cachefs` command checks the integrity of cached file systems. By default it corrects any CacheFS problems it finds. Unlike the standard `fsck` command there is no interactive mode.

The `fsck_cachefs` command syntax is:

```
fsck -F cachefs [ -m | -o noclean ] cache_directory
```

Table 42 provides the available flags for this command.

Table 42. fsck_cachefs Options

Flag	Description
-m	Check, but do not repair.
-o noclean	Force a check on the cache even if there is no reason to suspect there is a problem.
cache-directory	The name of the directory where the cache resides.

The following example shows the actions required to allow an NFS client machine to mount the remote filesystem /images from the server aix4xdev and use the CacheFS mechanism to improve performance.

The /images filesystem on the server is a large filesystem, so a dedicated JFS will be used to store the local cache.

1. Use SMIT to create a JFS of the required size to act as the cache. For example, /cacheFS.
2. Mount the JFS to be used for the cache.

   ```
   mount /cacheFS
   ```

   Create an empty cache structure to be used as the cache. This is created using the cfsadmin command. The argument is the name of the cache directory object you want to create. The object should not exist.

   ```
   cfsadmin -c /cacheFS/cachedir
   ```

The command uses default values for the various resource thresholds of the cache object. If you want to alter these, you should create the cache object using SMIT with the cachefs_admin_create fastpath.

3. Mount the remote filesystem from the server as a cacheFS filesystem type, specifying the cache object to use for backing store and the name of the mount point that will be used to access the filesystem.

```
mount -V cachefs -o backfstype=nfs,cachedir=/cachefs/cachedir
aix4xdev:/images /images
```

The result of the action can be checked using the `mount` command with no arguments to list all mounted filesystems. Three new entries are present. The first entry is for the JFS used to store the cache object. The second shows the remote filesystem mounted onto the cache object as an NFS mount. The remote filesystem should not be accessed through this mount point. The third new filesystem mount shows /images as a CacheFS mount on the cache object.

Figure 44 shows the mount output of a CacheFS enabled system.

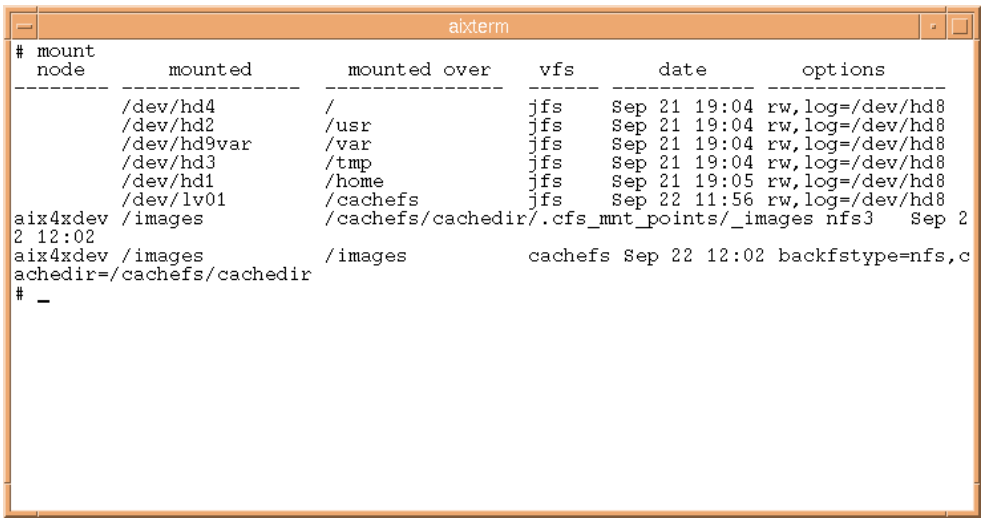


Figure 44. Output of mount Command Showing CacheFS

7.14.2 AutoFS (4.3.1)

AutoFS is the component of ONC+ that provides automatic mounting of NFS file systems. The automounting file system, autofs, mounts file systems when access is requested and unmounts the file system after a few minutes of inactivity, thus saving the network overhead traffic required to maintain the NFS connection. AutoFS allows you to break the connection when the file system is not being used and restart it again automatically when access is next desired.

7.14.2.1 How AutoFS Works

AutoFS is a client-side service. It is implemented using three components to accomplish automatic mounts. The components are:

- The `automount` command
- The autofs kernel extension
- The `automountd` daemon

The automount command is called at system startup time from /etc/rc.tcpip. It loads the autofs kernel extension if it is not already loaded and reads the master map information from the file /etc/auto_master. The automount command then passes the information it read from the master map to the autofs kernel extension. It then starts the automountd daemon and terminates.

The autofs kernel extension reads the information passed to it from the automount command and maintains an internal table of the autofs mounts. These autofs mounts are not automatically mounted at startup time. They are points under which file systems can be mounted in the future.

When a client attempts to access a file system that is not presently mounted, the autofs kernel extension intercepts the request and gets the automountd to mount the requested directory. The automountd daemon locates the directory, mounts it within autofs, and replies. On receiving the reply, autofs allows the waiting request to proceed. Subsequent references to the mount are redirected by the autofs. No further participation is required by automountd.

With this implementation of automatic mounting, the automountd daemon is completely independent from the automount command. Because of this separation, it is possible to add, delete, or change map information without first having to stop and start the automountd daemon process. Once the file system is mounted, further access does not require any action from automountd.

7.14.2.2 AutoFS Maps

AutoFS uses files referred to as maps for administration of network files. The map files contain information about filesystems and the names of the hosts on the network where the filesystems reside. Maps can be available locally or through a network name service like NIS. AutoFS uses three types of maps:

- Master maps
- Direct maps
- Indirect maps

7.14.2.3 Master Maps

The auto_master map associates a directory with a map. It is a master list specifying all the maps that autofs should know about.

Each line in the master map /etc/auto_master has the following syntax:

```
mount-point map-name [ mount-options ]
```

Mount-point

mount-point is the full (absolute) path name of a directory. If the directory does not exist, autofs creates it if possible. If the directory exists and is not empty, mounting on it hides its contents. In this case, autofs issues a warning message.

Map-name

map-name is the map autofs uses to find directions to locations, or mount information. If the name is preceded by a slash (/), autofs interprets the name as a local file. Otherwise, autofs searches for the mount information using the search specified in the name service switch configuration file.

Mount-options
mount-options is an optional, comma-separated list of options that apply to the mounting of the entries specified in map-name unless the entries in map-name list other options. The mount-options are the same as those for a standard NFS mount, except that bg (background) and fg (foreground) do not apply.

The auto_master map also has two special entries. They are:

```
+auto_master
/net            -hosts          -nosuid
```

The +auto_master entry is a reference to an external NIS master map. If one exists, then its entries are read as if they occurred in place of the +auto_master entry.

The /net entry is a mechanism that allows an NFS client to access all filesystems exported by a server. For example, accessing /net/fred will cause the autofs system to mount all of the filesystems exported by the machine fred under the root mount point /net/fred.

7.14.2.4 Direct Maps

A direct map is an automount point. With a direct map, there is a direct association between a mount point on the client and a directory on the server. Direct maps have a full path name and indicate the relationship explicitly. Lines in direct maps have the following syntax:

```
mount-point [ mount-options ] location
```

Key
The key is the path name of the mount point in a direct map.

Mount-options
The mount-options are the options you want to apply to this particular mount. They are required only if they differ from the map default.

Location
The location is the location of the file system. Specified as `server:pathname`.

An example of typical `/etc/auto_direct` map is:

```
/usr/local      -ro \
    /bin                    goanna:/export/local/bin \
    /share                  goanna:/export/local/share \
    /src                    goanna:/export/local/src
/usr/man        -ro         echidna:/usr/man \
                            platypus:/usr/man \
                            wombat:/usr/man \
/usr/games      -ro         koala:/usr/games
/usr/spool/news -ro         wallaby:/usr/spool/news \
                            wombat:/var/spool/news
```

7.14.2.5 Indirect Maps

An indirect map uses a substitution value of a key to establish the association between a mount point on the client and a directory on the server. Indirect maps are useful for accessing specific file systems like home directories. The auto_home map is an example of an indirect map. Lines in indirect maps have the following syntax:

```
key [ mount-options ] location
```

Key

key is a simple name (no slashes) in an indirect map.

Mount-options

The mount-options are the options you want to apply to this particular mount. They are required only if they differ from the map default.

Location

The `location` is the location of the file system, specified as server:pathname.

An example of a typical `auto_home` map is:

```
dale                        wombat:/export/home/dale
victor                      taipan:/export/home/victor
andrew                      emu:/export/home/andrew
swamy                       wallaby:/export/home/swamy
julia                       redback:/export/home/julia
tony                        goanna:/export/home/tony
george      -rw,nosuid      koala:/export/home/george
```

As an example, assume that the previous map is on host `echidna`. If user george has an entry in the password database specifying his home directory as /home/george, then whenever he logs into computer echidna, autofs mounts the directory/export/home/george residing on the computer koala.

7.14.3 NFS Server Performance Enhancement (4.3.2)

The NFS server performance of AIX 4.3.2 is enhanced with the implementation of a vnode cache in the JFS component of the kernel. The cache enables the NFS server code to translate an NFS file handle to a local vnode structure more efficiently than previous versions of AIX. As a result, the NFS server code spends less time holding a VFS lock word, which in turn, increases the available throughput of the NFS server.

Chapter 8. Graphical Environment Enhancements

Prior to the release of AIX Version 4.3, IBM shipped X11 Release 5 and additional backward-compatibility libraries plus Motif Version 1.2. AIX Version 4.3 provides graphics updates to two existing products. The first update, release 6 of the X-window system (X11R6), allows developers to use new extensions and enhancements to existing extensions. The second update, MotifNext or Motif 2.1, provides a range of new function including thread safe libraries and 64-bit enablement.

8.1 X-Windows Architecture Review

The X-windows architecture consists of three basic components: The client, protocol, and server. These components will be the subject of the following sections.

8.1.1 Client

The X Client is a software application that requests services from another application called the server, possibly across a network. A typical example would be a stock control application that used the services of an X Station to display output and accept user input from a keyboard or other input device. It is normally written in a high-level language, commonly C, and is linked with one or more of the X libraries. The instructions that the application developer uses to interact with the server are very low-level and allow them to open a connection with the server, create and manipulate windows, draw elementary shapes (lines, rectangles, circles, characters, and so on), and handle events, such as mouse movements or keystrokes. To increase productivity, toolkits have been developed that perform complex operations by calling several elementary subroutines. The most elaborate toolkits, such as Athena or Motif, also contain widgets. Widgets are program objects with a predefined appearance and behavior, such as a menu push button. The standard toolkits provided with AIX are:

Xt The basic X toolkit with hundreds of standard functions, also called intrinsics.

Xaw The Athena Widget set from the X Consortium.

Xmu Miscellaneous utilities.

Xext The extension library.

Xi The input extension library. It manages the peripherals, that is, keyboard and mouse.

Xm The Motif library of widgets with a 3D-look.

8.1.2 Protocol

The protocol is a formal description of the conversation that is carried out between the client and the server. The server and clients exchange information using messages. These messages are contained in packets that can be transported over a network if necessary. There are four packet types:

Request Used by the client to ask the server to perform an action.

Reply Used by the server to answer the client after a request has been received.

Event Used by the server to inform the client that something has happened that requires action by the client.

Error Used to indicate that something went wrong.

8.1.3 Server

The server is a program that runs on the workstation in which a graphical display is attached. The server program, called X, consists of two parts. One part is device-independent (dix) and interprets requests from Xlib, schedules client activity, manages the return of events and input to the Xlib library, and performs other generic actions. The second part is device-dependent (ddx) and renders the 2D graphic operations defined by the X-window system for the specific display adapter. The loadddx GAI (Graphic Adapter Interface) load module implements this interface. The server modules have strictly-defined function and peripheral support, and the only way to increase the capability of the X-server is through extensions. Extensions are new modules that can be loaded into the server and used to perform actions that the basic modules are incapable of.

For example, the core server (the core server is the name given to the server without any extensions) only supports the keyboard and mouse as peripherals. If you intend to use other peripherals such as a spaceball, dials, or scanner, you need to load an extension that is capable of handling these devices.

All of the extensions that are capable of being loaded will be discussed in the following section.

8.2 X-Windows System Release 6

The sixth release of the X-window system has been ported to AIX Version 4.3 from software provided by the X Consortium. The X Consortium is an independent, non-profit corporation, the successor to the MIT X Consortium, which was part of the MIT Laboratory for Computer Science. The actual source for the port was X11 Release 6.2, which is a proper subset of X11 R6.3 produced at the request of the OSF Common Desktop Environment (CDE) program. It was produced by the X Consortium and is being released by OSF simultaneously with CDE 2.1. Release 6.2 contains only the print extension and the Xlib implementation of vertical writing and user-defined character support. R6.3 is an update to R6.1. It is compatible with R6 and R6.1 at the source and protocol levels in all respects, and binaries are upward-compatible.

This section describes changes in X11 since Release 5. Release 6.2 contains new function in many areas. In addition, many errors have been corrected. Except where noted, all libraries, protocols, and servers are upward-compatible with Release 5. That is, R5 clients and applications should continue to work with R6 libraries and servers.

The following are new X Consortium standards in Release 6. Each is described in its own section below:

- X11 Security
- X Image Extension
- Inter-Client Communications Conventions Manual (update)
- Inter-Client Exchange Protocol

- Inter-Client Exchange Library
- X Session Management Protocol
- X Session Management Library
- Input Method Protocol
- X Logical Font Descriptions (update)
- SYNC Extension
- XTEST Extension
- BIG-REQUESTS Extension
- XC-MISC Extension
- X Keyboard Extension (XKB)
- RECORD Extension
- Double Buffer Extension (DBE)
- ICE X Rendezvous
- Print Extension
- Xlib Vertical Writing and User-Defined Characters

8.2.1 X11 Security

The X11R6 implementation provides five access control mechanisms, of which the two listed below are supported in AIX 4.3:

Host Access

Any client on a host in the host access control list is allowed access to the X-server. The access control list is changed with the `xhost` command.

MIT-MAGIC-COOKIE-1

A 128-bit plain-text cookie is provided by the client with the connection setup information. The xdm program automatically configures the X-server and client for each new login session.

8.2.2 X Image Extension

The sample implementation in Release 6 is a complete implementation of the full XIE 5.0 protocol except for the following techniques that are excluded from the sample implementation:

- ColorAlloc: Match, Requantize
- Convolve: Replicate
- Decode: JPEG (lossless)
- Encode: JPEG (lossless)
- Geometry: AntialiasByArea, AntialiasByLowpass

8.2.3 Inter-Client Communications Conventions Manual

X11 Release 6 includes Version 2.0 of the Inter-Client Communications Conventions Manual (ICCCM). This version contains a large number of changes and clarifications in the areas of window management, selections, session management, and resource sharing.

8.2.3.1 Window Management

The circumstances under which the window manager is required to send synthetic ConfigureNotify events have been clarified to ensure that any ConfigureWindow request issued by the client will result in a ConfigureNotify event, either from the server or from the window manager. It also added advice about how a client should inspect events to minimize the number of situations where it is necessary to use the TranslateCoordinates request.

The window_gravity field of WM_NORMAL_HINTS has a new value, StaticGravity, which specifies that the window manager should not shift the client windows location when re-parenting the window.

The base size in the WM_NORMAL_HINTS property is now to be included in the aspect-ratio calculation.

The WM_STATE property now has a formal definition (it was previously only suggested).

8.2.3.2 Selections

The CLIENT_WINDOW, LENGTH, and MULTIPLE targets have been clarified. A number of new targets for Encapsulated PostScript and the Apple Macintosh PICT-structured graphics format have been added. Also, a new selection property type C_STRING (a string of non-zero bytes) was defined. This is in contrast to the STRING type, which excludes many control characters. A selection requester can now pass parameters with the request. Another new facility is *manager selections*. This use of the selection mechanism is not to transfer data, but to allow clients known as managers to provide services to other clients. Version 2.0 also specifies that window managers should hold a manager selection. Currently, the only service defined for window managers is to report the ICCCM version number in which the window manager complies. Now that this facility is in place, additional services can be added in the future.

8.2.3.3 Resource Sharing

A prominent new addition in Version 2.0 is the ability of clients to take control of colormap installation under certain circumstances. Earlier versions of the ICCCM specified that the window manager had exclusive control over colormap installation. This proved to be inconvenient for certain situations, such as when a client has the server grabbed. Version 2.0 allows clients to install colormaps after having informed the window manager. Clients must hold a pointer grab for the entire time they are doing their own colormap installation.

Version 2.0 also clarifies a number of rules about how clients can exchange resources. These rules are important when a client places a resource ID into a hints property or passes a resource ID through the selection mechanism.

8.2.3.4 Session Management

Some of the properties in Section 5 of ICCCM 1.1 are now obsolete, and new properties for session management have been defined.

8.2.4 ICE (Inter-Client Exchange)

ICE provides a common framework in which to build protocols. It supplies authentication, byte order negotiation, version negotiation, and error reporting conventions. It supports multiplexing multiple protocols over a single transport

connection. ICElib provides a common interface to these mechanisms so that protocol implementors do not need to reinvent them. An `iceauth` program was written to manipulate an ICE authority file. It is very similar to the `xauth` program.

8.2.5 SM (Session Management)

The X Session Management Protocol (XSMP) provides a uniform mechanism for users to save and restore their sessions using the services of a network-based session manager. It is built on ICE. SMlib is the C interface to the protocol. There is also support for XSMP in Xt. A simple session manager, `xsm`, is included.

A new protocol, `rstart`, greatly simplifies the task of starting applications on remote machines. It is built upon already existing remote execution protocols such as `rsh`. The most important feature that it adds is the ability to pass environment variables and authentication data to the applications being started.

8.2.6 X Logical Font Description

The X Logical Font Description has been enhanced to include general 2D linear transformations, character set subsets, and support for polymorphic fonts.

8.2.7 SYNC Extension

The SYNC extension lets clients synchronize through the X- server. This eliminates the network delays and the differences in synchronization primitives between operating systems. The extension provides a general counter resource, allowing clients to alter the value of a counter and block their execution until a counter reaches a specific threshold. For example, two clients share a counter initialized to zero; one client can draw some graphics and then increment the counter. The other client can block until the counter reaches a value of one and then draw some additional graphics.

8.2.8 XC-MISC Extension

A new extension, XC-MISC, allows clients to retrieve ID ranges from the server. Xlib handles this automatically. This is useful for long-running applications that use many IDs over their lifetime.

8.2.9 BIG-REQUESTS Extension

The standard X protocol only allows requests up to 2^{18} bytes long. A new protocol extension, BIG-REQUESTS, has been added that allows a client to extend the length field in protocol requests to be a 32-bit value. This is useful for PEX and other extensions that transmit complex information to the server. The BIG-REQUESTS have already been implemented by IBM as an extension to X11 Release 5.

8.2.10 Double Buffer Extension (DBE)

The Double Buffer Extension (DBE) provides a standard way to utilize double-buffering, allowing flicker-free animation.

The older Multi-Buffering extension is not linked in to the X-server by default. It will move to unsupported status at the next release.

8.2.11 X Keyboard Extension

Prior to the introduction of the X Keyboard Extension (XKB) in X11R6, the core X protocol was used for keyboard interaction. The core X protocol has a number of limitations that make it difficult, or impossible, to properly support many common varieties of keyboard behavior. The X Keyboard Extension provides capabilities that are lacking, or cumbersome, in the core X protocol.

8.2.11.1 XKB Keyboard Extension Support for Keyboards

The X Keyboard Extension makes it possible to clearly and explicitly specify most aspects of keyboard behavior on a per-key basis. It adds the notion of a keyboard group to the global keyboard state and provides mechanisms to closely track the logical and physical state of the keyboard. For keyboard-control clients, XKB provides descriptions and symbolic names for many aspects of keyboard appearance and behavior.

In addition, the X Keyboard Extension includes additional keyboard controls designed to make keyboards more accessible to people with mobility impairments.

8.2.11.2 XKB Extension Components

The XKB extension is composed of two parts:

- A server extension.
- A client-side Xlibrary extension.

The server portion of the XKB extension encompasses a database of named keyboard components, in unspecified format, that may be used to configure a keyboard. Internally, the server maintains a *keyboard description* that includes the keyboard state and configuration (mapping). Keyboard is defined as the logical keyboard device, which includes not only the physical keys, but also potentially a set of up to 32 indicators (usually LEDs) and bells.

The *keyboard description* is a composite of several different data structures, each of which may be manipulated separately. The individual components are shown in Figure 45.

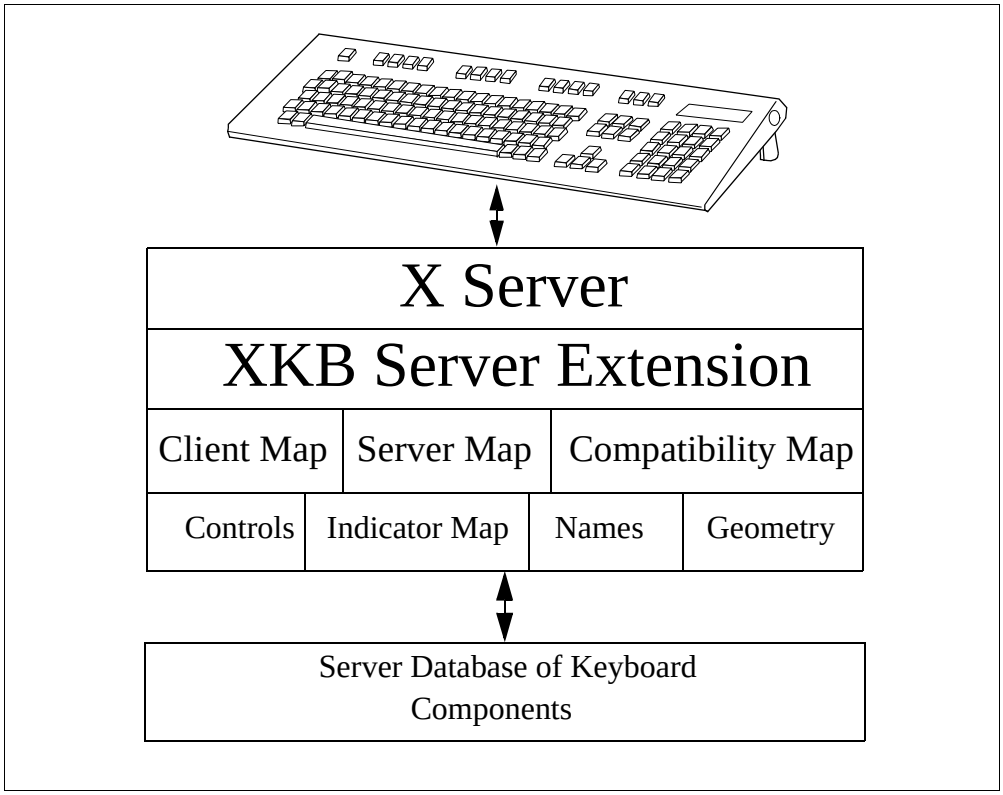


Figure 45. XKB Server Extension

Client Map The key mapping information needed to convert arbitrary keycodes to symbols.

Server Map The key mapping information categorizing keys by function (which keys are modifiers, how keys behave, and so on).

Controls Client configured quantities affecting how the keyboard behaves, such as repeat behavior and modifications for people with movement impairments.

Indicators The mapping of behavior to indicators.

Geometry A complete description of the physical keyboard layout sufficient to draw a representation of the keyboard.

Names A mapping of names to various aspects of the keyboard, such as individual virtual modifiers, indicators, and bells.

Compatibility Map The definition of how to map core protocol keyboard state to XKB keyboard state.

A client application interrogates and manipulates the keyboard by reading and writing portions of the server description for the keyboard. In a typical sequence, a client would fetch the current information it is interested in, modify it, and write it back. If a client wants to track some portion of the keyboard state, it typically maintains a local copy of the portion of the server keyboard description working with the items of interest and updates this local copy from events describing state transitions that are sent by the server. A client may request the server to reconfigure the keyboard either by sending explicit reconfiguration instructions or

by telling it to load a new configuration from its database of named components. Partial reconfiguration and incremental reconfiguration are both supported

8.2.11.3 Groups and Shift Levels

The graphic characters, or control functions, that can be accessed by one key are logically arranged in groups and levels. For example, the Radio Group is a set of keys whose behavior simulates a set of radio buttons. Once a key in a radio group is pressed, it stays logically depressed until another key in the group is pressed, at which point the previously-depressed key is logically released. Consequently, at most one key in a radio group can be logically depressed at one time. A radio group is defined by a radio group index, an optional name, and by assigning each key in the radio group XKBKB_RadioGroup behavior and the radio group index.

8.2.11.4 Client Types

The X11R6 specification differentiates between three different classes of client applications as shown in Figure 46.

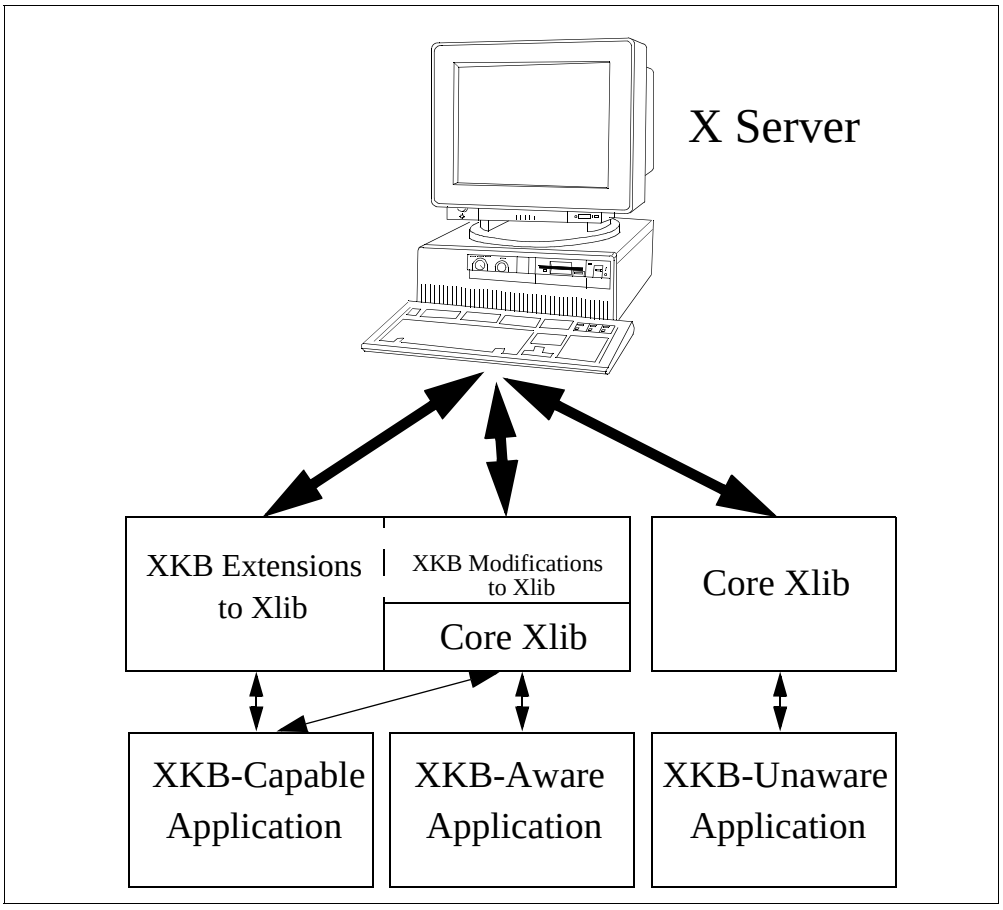


Figure 46. Types of XKB Clients

- XKB-aware applications. These applications make specific use of XKB function and Application Programming Interfaces (APIs) not present in the core protocol.

- XKB-capable applications. These applications do not use XKB extended function and APIs directly. However, they are linked with a version of Xlib that includes XKB and indirectly benefit from some of XKB's features.
- XKB-unaware applications. These applications do not make use of XKB extended function or APIs and require XKB's function to be mapped to core Xlib function to operate properly.

8.2.11.5 Protocol Errors

The XKB extension adds a single protocol error, BadKeyboard, to the core protocol errorset. Table 43 lists the protocol errors that can be generated and their causes.

Table 43. XKB Protocol Errors

Protocol error	Error cause
BadAccess	The XKB extension has not been properly initialized.
BadKeyboard	The device specified was not a valid core or input extension device.
BadImplementation	Incorrect reply from server.
BadAlloc	Unable to allocate storage.
BadMatch	A compatible version of XKB was not available in the server or an argument has correct type and range but is otherwise in error.
BadValue	An argument is out of range.
BadAtom	A name is neither a valid Atom nor None.
BadDevice	Device, Feedback Class, or Feedback ID is in error.

8.2.11.6 Extension Library Functions

The X Keyboard Extension replaces the core protocol definition of a keyboard with a more comprehensive one. The X Keyboard Extension library interfaces are included in Xlib.

Xlib detects the presence of the X Keyboard server extension and uses XKB protocol to replace some standard X library functions related to the keyboard. If an application uses only standard X library functions to examine the keyboard or process key events, it should not need to be modified when linked with an X library containing the X keyboard extension. All of the keyboard-related X library functions have been modified to automatically use XKB protocol when the server extension is present.

The XKB extension adds library interfaces to allow a client application to directly manipulate the new capabilities.

8.2.11.7 XKB Client Applications

With XKB, there are several new core clients:

- `xkbcomp`
- `xkbevd`
- `kbevd`
- `xkbprint`

8.2.12 X Record Extension

The purpose of this extension is to support the recording and reporting of all core X protocol and arbitrary X extension protocol.

The extension is used to record the core X protocol and arbitrary X extension protocol entirely within the X server itself. When the extension has been requested to record specific protocol by one or more recording clients, the protocol data is formatted and returned to the recording clients. The extension provides a mechanism for capturing all events, including input device events that do not go to any clients.

8.2.13 ICE X Rendezvous

The Inter-Client Exchange protocol (ICE) that became a standard in X11R6 specifies a generic communication framework for data exchange between arbitrary clients. The ICE protocol itself does not specify the manner in which two clients interested in communicating using ICE are made aware of each other's existence.

The ICE X Rendezvous protocol is one standard protocol by which two clients who have connections to a common X server can rendezvous. This new protocol is included in the ICE Protocol Specification document.

8.2.14 Print Extension

The print extension supports output to hardcopy devices using the core X drawing requests. The print extension adds requests for job and page control and defines how specific printer attributes are communicated between the server and printing clients. Printer attribute specifications are modeled after the ISO 10175 specification.

An X Client that wants to produce hardcopy output will typically open a second connection to an X print server, produce a print job, and then close the print server connection. The print server may be the same process as the display server (the term video server is sometimes used), although the implementation provided in R6.2 does not completely support video and print servers in the same binary.

8.2.14.1 Running an X Print Server

The print server is simply an X-server with the print extension and special DDX implementations. The X print server is started like any other X-server. The following command line is an example for use with a typical configuration:

```
# Xprt :1 -ac
```

The options used in the example are:

:1 On a host that is running a video display server you will need to specify a different display from the default.

-ac Disable access control since no simple mechanism for sharing keys is provided.

The X print server also supports the following additional options:

-XpFile

Points to the directory containing the print server configuration files.

XPCONFIGDIR

Environment variable specifying alternative location of the print server configuration files.

The print server is configured through a directory of configuration files that define printer model types and instances of printer models. An example configuration tree is provided.

By convention, clients locate the print server using the environment variable XPRINTER. The syntax of XPRINTER is an augmented DISPLAY:

```
printerName@host:display
```

where `printerName` is one of the printer instances listed in the print server configuration files. The use of XPRINTER, and its syntax, is an application convention only; there is nothing in the supplied libraries that uses (or parses) this environment variable.

8.2.15 Xlib Vertical Writing and User-Defined Characters

The Xlib output method implementation has been enhanced to support the XOM drawing direction XOMOrientation_TTB_RTL. Vertical writing information, and other locale specific information, is read from the file <XLocaleDir>/%L/XLC_LOCALE.

The X[mb|wc]TextEscapement functions now return the text escapement in pixels for the vertical or horizontal direction depending on the XNOrientation XOCValue.

The X[mb|wc]DrawString functions will now render a character string in the vertical or horizontal direction depending on the XNOrientation XOCValue.

The Xlib NLS database implementation has been enhanced to support extended segments used for interchanging non-standard code sets. Support has been added for control sequences and encoding names used in extended segments and conversion of glyph indexes when interchanging data in extended segments.

8.2.16 Xlib Library

Xlib now supports multi-threaded access to a single display connection. Xlib functions lock the display structure, causing other threads calling Xlib functions to be suspended until the first thread unlocks. Threads inside Xlib waiting to read to or write from the X-server do not keep the display locked, so, for example, a thread hanging on XNextEvent() will not prevent other threads from doing output to the server.

The display and GC structures have been made opaque to normal application code; references to private fields will get compiler errors. You can work around some of these by compiling with -DXLIB_ILLEGAL_ACCESS, but it is better to fix the offending code.

The Xlib implementation has been changed to support a form of asynchronous replies, meaning that a request can be sent to the server, and then other requests can be generated without waiting for the first reply to come back. This is used to an advantage in two new functions, XInternAtoms() and XGetAtomNames(), which reduce what would otherwise require multiple round trips to the server

down to a single round trip. It is also used in some existing functions, such as XGetWindowAttributes(), to reduce two round trips to just one.

Support for using poll(), rather than select(), is implemented, selected by the HasPoll configuration option.

Table 44 provides the Xlib functions that are new in Release 6.

Table 44. New X11R6 Xlib Functions

_XAllocTemp()	_XFreeTemp()
IsPrivateKeypadKey()	XAddConnectionWatch()
XAllocIDs()	XCloseOM()
XCreateOC()	XContextualDrawing()
XConvertCase()	XDestroyOC()
XDirectionalDependentDrawing()	XDisplayOfOM()
XESetBeforeFlush()	XExtendedMaxRequestSize()
XGetAtomNames()	XGetOCValues()
XGetOMValues()	XInitImage()
XInitThreads()	XInternalConnectionNumbers()
XInternAtoms()	XLocaleOfOM()
XLockDisplay()	XOMOfOC()
XOpenOM()	XProcessInternalConnection()
XReadBitmapFileData()	XRegisterIMInstantiateCallback()
XRemoveConnectionWatch()	XSetIMValues()
XSetOCValues()	XSetOMValues()
XUnlockDisplay()	XUnregisterIMInstantiateCallback()

8.2.17 Xt Toolkit

Support has been added for participation in session management, with call-backs to application function in response to messages from the session manager. In addition, the entire library is now thread-safe. This allows one thread at a time to enter the library and also protects global data from concurrent use. Support is also provided for registering event handlers for events generated by X protocol extensions and for dispatching those events to the appropriate widget.

A mechanism has also been added for dispatching events for non-widget drawings (such as pixmaps used within a widget) to a widget.

Two new widget methods, for instance, allocation and deallocation, allow widgets to be treated as C++ objects in a C++ environment.

A new interface allows bundled changes to the managed set of children of a *Composite*, reducing the visual disruption of multiple changes to geometry layout.

Several new resources have been added to shell widgets, making the library compliant with the Release 6 ICCCM. Parameterized targets of selections (new in Release 6) and the MULTIPLE target are supported with new APIs.

The client will be able to register callbacks on a per-display basis for notification of a large variety of operations in the X toolkit. This feature is useful to external agents, such as screen readers.

The file search path syntax has a new %D substitution that inserts the default search path, making it easy to prepend and append to the default search path.

The Xt implementation allows a configuration choice of poll() or select() for I/O multiplexing, selectable at compile time by the HasPoll configuration option.

The Release 6 Xt implementation requires Release 6 Xlib. Specifically, it uses the following new Xlib features: XInternAtoms() instead of multiple XInternAtom() calls where possible, input method support (Xlib internal connections), and tests for the XVisibleHint in the flags of XWMHints.

When linking with Xt, you now need to link with SMlib and ICElib as well. This is automatic if you use the XTOOLLIB make variable or XawClientLibs make variable in your Imakefiles.

This implementation no longer allows NULL to be passed as the value in the name/value pair in a request to XtGetValues(). The default behavior is to print the following error message and exit:

```
NULL ArgVal In XtGetValues
```

8.2.18 Xaw Toolkit

Some minor code corrections have been integrated. Text and Panner widget translations have been augmented to include keypad cursor keysyms in addition to the normal cursor keysyms.

The Clock, Logo, and Mailbox widgets have moved to their respective applications.

Internationalization support is now included. Xaw uses native widechar support when available; otherwise, it uses the Xlib widechar routines. Per-system specifics are set in XawI18n.h.

8.2.18.1 AsciiText

The name AsciiText is now a misnomer but has been retained for backward-compatibility. A new resource, XtNinternational, has been added. If the value of the XtNinternational resource is False (the default), AsciiSrc and AsciiSink source and sink widgets are created, and the widget behaves as it did for R5. If the value is True, MultiSrc and MultiSink source and sink widgets are created. The MultiSrc widget will connect to an input method server if one is available, or if one is not available, it will use an Xlib internal pseudo-input method that, at a minimum, does compose processing. Application programmers who want to use this feature will need to add a call to XtSetLanguageProc() to their programs.

The symbolic constant FMT8BIT has been changed to XawFmt8Bit to be consistent with the new symbolic constant XawFmtWide. FMT8BIT remains for

backwards compatibility; however, its use is discouraged as it will eventually be removed from the implementation.

Two new resources have been added, XtNinternational and XtNfontSet. These resources have been added to the following widgets:

- Command widget
- Label widget
- List widget
- MenuButton widget
- Repeater widget
- SmeBSB widget
- Toggle widget

If XtNinternational is set to True, the widget displays its text using the specified font set.

8.2.19 Header Files

Two new macros are defined in Xos.h, X_GETTIMEOFDAY and strerror.

X_GETTIMEOFDAY is like gettimeofday() but takes one argument on all systems. strerror is defined only on systems that do not already have it.

A new header file, Xthreads.h, provides a platform-independent interface to threads functions on various systems. When programming, include it instead of the system threads header file. Use the macros defined in it instead of the system threads functions.

Xalloca.h is solely responsible for defining ALLOCATE_LOCAL and DEALLOCATE_LOCAL. You should be able to add or update a platform's support for alloca() by editing this one file instead of finding and changing the multiple definitions that existed previously. Xpoll.h allows more portable, consistent select() and poll() use in the clients, including getting the fd_set properly defined. The servers still use select() on all systems, even those that have poll().

8.2.20 Fonts

There are three new Chinese bdf fonts: gb16fs.bdf, gb16st.bdf, and gb24st.bdf.

The Type 1 fonts contributed by Bitstream, IBM, and Adobe that shipped in /contrib in Release 5 have been moved into the core. Some of the *misc* fonts, mostly in the *Clean* family, have only the ASCII characters but were incorrectly labeled ISO8859-1. These fonts have been renamed to be ISO646.1991-IRV. Aliases have been provided for the Release 5 names.

The 9x15 font has new shapes for some characters. The 6x10 font has the entire ISO 8859-1 character set.

8.2.20.1 Font Library

The Type 1 rasterizer that shipped in /contrib in Release 5 is now part of the core.

There is an option to have the X server request glyphs only as it needs them. The X-server then caches the glyphs for future use.

Aliases in a fonts.alias file can allow one scalable alias name to match all instances of another font. The exclamation (!) character introduces a comment line in fonts.alias files.

A sample font authorization protocol, hp-hostname-1, has been added. It is based on host names and is non-authenticating. The client requesting a font from a font server provides (or passes through from its client) the host name of the ultimate client of the font.

Note: There is no check that this host name is accurate, as this is a sample protocol only.

The Speedo rasterizer can now read fonts with retail encryption. This means that fonts bought over-the-counter at a computer store can be used by the font server and X-server.

8.2.20.2 Font Server

The font server has been renamed from `fs` to `xfs` to avoid confusion with an AFS program. The `fsconf` utility is also renamed to `xfsconf`. The default port has changed from 7000 (used by AFS) to 7100 and has been registered with the Internet Assigned Numbers Authority. Until AIX 4.2.1, the default port was 7500 to avoid conflict with the IBM X Station at the time R5 was first shipped on AIX. This is changed to the new X11R6 default port.

You can now connect to `xfs` using the local/transport.

8.2.21 X Input Method

For each of the different locales, there is a single default input method associated with it. The AIX X-window system, prior to the release of X11R6, used the following default X Input Methods (XIM):

Local A local single byte input method

Thai A local type Thai input method

X11R6 XIM Protocol All of server type input methods

As well as these input methods, AIX provides a native input method, AIX IM, with the release of X11R6. The AIX implementation of X11R6 puts the AIX Input Method as the default input method.

8.2.21.1 XIM Module Loader

The introduction of the AIX Input Method as the default input method changes the loading priority of the different input method modules. The new order is show in the Table 45.

Table 45. XIM Module Loading Priorities

Priority	XIM Module	Locales	XMODIFIERS
1	AIX IM	AIX system	Depend on AIX IM
2	XC's Local	XLOCALEDIR	Local, none
3	XC's Thai	th	Don't care
4	XC's Proto	XIM server	Depend on IM servers

The following is a list of AIX XIM module features:

- AIX XIM module has the first loading priority.
- AIX XIM module tries to query specified language and IM modifier (XMODIFIERS) to the AIX Input Method.

8.2.21.2 AIX XIM Interface for Input Method Switching

X Consortium's sample code implements an input method switching mechanism at the top of XIM layer. All X input methods are registered at _XimImSportRec in the imImSw.c file. The _XimOpenIM() function invokes the checkprocessing method of the registered X input method to check that the X input method is supported by the specified locale and IM modifier. If the input method is supported, then the _XimOpenIM() function calls im_open method of the XIM. If it fails to open IM, then the im_free method is called to close it down.

The following section describes the methods provided by AIX's XIM module:

(*checkprocessing)() - _XimCheckIfAIXProcessing
This routine is called from the _XimOpenIM() function to check that a XIM module supports specified IM information (locale and modifiers).

This function returns True with the following conditions:

- If the XMODIFIERS is not specified, and the specified locale is supported by the AIX Input Method.
- If the specified XMODIFIERS and locale combination are supported by the AIX Input Method.

If the XMODIFIERS is specified, a combined AIX IM name is used to check if the specified locale and modifier combination are supported by the AIX IM. To make a combined AIX name, concatenate the IM modifier to the locale name. The AIX IM function accepts a language name that contains the IM modifier @im=.

For example:

```
$LANG = xx_XX

$XMODIFIERS = @im=foo

Combined Name = xx_XX@im=foo

AIX IM module = xx_XX@foo
```

(*im_open)() - _XimAIXOpenIM
This routine is called from the _XimOpenIM() function when an application calls XOpenIM().

(*im_free)() - _XimAIXIMFree
This routine is called from the im_free() function and is used to close down the input method.

8.2.22 Input Method Protocol

Some languages need complex pre-editing input methods. Such an input method may be implemented separately from applications in a process called an Input Method (IM) Server. The IM Server handles the display of pre-edit text and the user's input operation. The Input Method (IM) Protocol standardizes the

communication between the IM Server and the IM library linked with the application.

The IM Protocol is a completely new protocol based on experience with R5's sample implementations. The following new features are added beyond the mechanisms in the R5 sample implementations:

- The IM Server can support any of several transports for connection with the IM library.
- Both the IM Server and clients can authenticate each other for security.
- A client can connect to an IM Server without restarting, even if it starts up before the IM Server.
- A client can initiate string conversion to the IM Server for reconversion of text.
- A client can specify some keys as hot keys, which can be used to escape from the normal input method processing regardless of the input method state.

The R6 sample implementation for the internationalization support in Xlib has a new plugin framework with the capability of loading and switching locale object modules dynamically. For backward compatibility, the R6 sample implementation can support the R5 protocols by switching to IM modules supporting those protocols. In addition, the framework provides the following new functions and mechanisms:

X Locale database format

An X locale database format is defined, and the subset of a user's environment dependent on language is provided as a plain ASCII text file. You can customize the behavior of Xlib without changing Xlib itself.

ANSI C and non-ANSI C bindings

The common set of methods and structures are defined that bind the X locale to the system locales within libc, and a framework for implementing this common set under non-ANSI C base system is provided.

Converters

The sample implementation has a mechanism to support various encodings by plugin converters and provides the following converters:

- Light-weight converter for C and ISO 8859
- Generic converter
- High-performance converter for Shift-JIS and EUC
- Converter for UCS-2 defined in ISO/IEC 10646-1

Locale modules

The library is implemented so that input methods and output methods are separated and are independent of each other. Therefore, an output-only client does not link with the IM code, and an input-only client does not link with the OM code. Locale modules can be loaded on demand if the platform supports dynamic loading.

Transport Layer

There are several kinds of transports for connection between the IM library and the IM Server. The IM Protocol is independent of a specific transport layer protocol, and the sample implementation has a

mechanism to permit an IM Server to define the transports that the IM Server is willing to use. The sample implementation supports transport over the X protocol, TCP/IP, and DECnet.

There are IM Servers for Japanese, Korean, and internationalized clients using IM services.

8.2.23 New Functions

The following sections discuss the newest AIX functions.

8.2.23.1 Input Method Values

Table 46 provides the input method values.

Table 46. New Input Method Values

XIM Value	Supported as XIM Value	Optional in R6
XNQueryInputStyle	Yes	
XNResourceName	Yes	
XNResourceClass	Yes	
XNDestroyCallback	Yes	
XNQueryIMValuesList	Yes	
XNQueryICValuesList	Yes	
XNVisiblePostion	No	Optional
XNR6PreeditCallbackBehavior	No	Optional

8.2.23.2 Input Context Values

Table 47 provides the input context values.

Table 47. New Input Context Values

XIC Value	Supported as XIC Value	Optional in R6
XNArea	Yes	
XNAreaNeeded	Yes	
XNBackground	Yes	
XNBackgroundPixmap	Yes	
XNClientWindow	Yes	
XNColormap	Yes	
XNCursor	Yes	
XNDestroyCallback	Yes	
XNFilterEvents	Yes	
XNFocus Window	Yes	
XNFontSet	Yes	
XNForeground	Yes	
XNGeometryCallback	Yes	

XIC Value	Supported as XIC Value	Optional in R6
XNHotKey	Yes	Optional
XNHotKeyState	Yes	
XNInputStyle	Yes	
XNLineSpacing	Yes	
XNPreeditCallbacks	Yes	
XNPreeditState	Yes	Optional
XNPreeditStateNotifyCallback	Yes	
XNResetState	Yes	Optional
XNResourceClass	Yes	
XNResourceName	Yes	
XNStatusCallbacks	Yes	
XNStringConversion	No	Optional
XNStringConversionCallback	No	Optional

8.2.24 X Output Method

Locale-dependent text may include one or more text components, each of which may require different fonts and character set encodings. In some languages, each component might have a different drawing direction, and some components might contain context-dependent characters that change shape based on relationships with neighboring characters.

When drawing such locale-dependent text, some locale-specific knowledge is required; for example, what fonts are required to draw the text, how the text can be separated into components, and which fonts are selected to draw each component. Furthermore, when bidirectional text must be drawn, the internal representation order of the text must be changed into the visual representation order to be drawn. An X Output Method provides a functional interface so that clients do not have to be aware of locale-dependent details.

Two different abstractions are used in the representation of the output method for clients:

- The abstraction used to communicate with an output method is an opaque data structure represented by the *XOM* datatype.
- The abstraction for representing the state of a particular output thread is called an output context. The Xlib representation of an output context is an *XOC*, which is compatible with XFontSet (see Section 8.2.25.1, “XLC_FONTSET Category” on page 210) in terms of its functional interface.

8.2.24.1 Output Method Functions

The following are the various output method functions:

- XOpenOM()
- XCloseOM()
- XSetOMValues()

- XGetOMValues()
- XDisplayOfOM()
- XLocaleOfOM()

8.2.24.2 Output Context Functions

An output context is an abstraction that contains both the data required by an output method and the information required to display that data. There can be multiple output contexts for one output method. The programming interfaces for creating, reading, or modifying an output context use a variable argument list. The name elements of the argument lists are referred to as XOCvalues. It is intended that output methods be controlled by these XOCvalues. As new XOCvalues are created, they should be registered with the X Consortium. An XOC can be used anywhere an XFontSet() can be used, and conversely.

The concepts of output methods and output contexts include broader, more generalized, abstraction than font set, supporting complex and more intelligent text display, and dealing not only with multiple fonts but also with context dependencies. However, XFontSet() is widely-used in several interfaces, so XOC is defined as an upward-compatible type of XFontSet().

8.2.25 X11R6 NLS Database

The locale sensitive functions in Xlib refer to the X NLS database or X Locale Database. The Locale Database is the internationalized portion of Xlib but is not actually part of Xlib itself. This allows easier customizing of the X Locale Database without affecting the X environment.

In X11R6 the constituent elements of an X NLS database are:

- locale_name/XLC_LOCALE
- locale_name/Compose
- locale.alias
- locale.dir
- locale.ldx
- compose.dir
- tbl_data/charset_table

In X11R6 the location of the NLS database has changed from that of X11R5. Currently, the X11R5 NLS database is found under /usr/lib/X11/nls, and the X11R6 NLS database is found under /usr/lib/X11/locale. This is of importance from a binary-compatibility standpoint with X11R5 libX11.a provided for backwards-compatibility.

Each locale database file (XLC_LOCALE) contains one or more category definitions. In X11R6, the category XLC_FONTSET defines the XFontSet-related information, and the category XLC_XLOCALE defines the character classification and encoding conversion information.

8.2.25.1 XLC_FONTSET Category

The XLC_FONTSET category defines the XFontSet relative information. It contains the CHARSET_REGISTRY_CHARSET_ENCODING name and

character mapping side (GL, GR, and so on), which is used in the X output method, see Section 8.2.24, “X Output Method” on page 209.

The XLC_FONTSET information is held in the class fsN (where N=0,1, 2,...). The fsN class includes encoding information for the Nth character set or charset. For example, if there are four charsets available in the current locale, four fontsets, fs0, fs1, fs2, and fs3, should be defined. This class has two subclasses:

charset Specifies encoding information to be used internally in Xlib for this fontset. An example of the charset format is ISO8859-1:GL

font Specifies a code set of the font to be used internally in Xlib for searching appropriate fonts for this fontset. The left-most entry in this field has the highest priority.

The XLC_FONTSET category also contains two classes not documented in the X11R6 "X Locale Database Definition". These are:

object_name
Allows selection of which X Output Method (XOM) will be used.

on_demand_loading
Delays loading the fonts (not the font information) until they are actually needed. In some cases the on_demand_loading feature has been used to improve font performance.

8.2.25.2 XLC_XLOCALE Category

The XLC_XLOCALE category defines character classification, conversion, and other character attributes.

Note: The X11R5's locale database is not compatible with X11R6's locale database.

8.2.25.3 locale_name/Compose

The locale_name/Compose file contains the compose sequence rule of the locale for the local input method. This file is mapped to the compose.dir file according to a locale full name (see Section 8.2.25.4, “Configuration Files” on page 211). This file is not used by the AIX specific input methods. AIX supplies compose tables as part of the base operating system.

8.2.25.4 Configuration Files

All configuration files of the X11R6 NLS database are under the /usr/lib/X11/locale directory.

locale.alias This file maps a simplified locale name, or alias, to a full locale name. For example, en_US to en_US.ISO8859-1. The locale.alias file is referenced first by Xlib to map the simplified platform-specific locale name to the X internal full locale name. Xlib uses the full locale name to lookup further information.

locale.dir This file maps a full locale name to the locale database file that should be read. The locale.dir file is referenced after the locale.alias file to determine the locale definition file corresponding to the X internal locale name. For example, en_US.ISO8859-1 to iso8859-1/XLC_LOCALE. Note that /usr/lib/X11/locale is assumed, so libX11 should read in the file

/usr/lib/X11/locale/iso88590-1/XLC_LOCALE when running in the en_US locale.

locale.ldx This file is read by the AIX dynamic locale loader and maps a locale name to an AIX-specific XLC. For every locale, one of the following is specified:

STATIC - Tells the AIX locale loader to use the statically linked AIX XLC object.

DYNAMIC - Specifies that an XLC should be dynamically loaded.

NONE - Tells the AIX locale loader to return without loading a locale.

8.2.25.5 tbl_data/charset_table

The tbl_data/charset_table file contains the charset conversion table for each charset. This file is used by the UTF locale in the sample implementation and is not used by AIX X11R6 XLC.

8.2.26 Command Line Interfaces

The following sections describe several command line interfaces.

8.2.26.1 xhost

Two new families have been registered: LocalHost, for connections over a non-network transport, and Krb5Principal, for Kerberos V5 principals.

To distinguish between different host families, a new xhost syntax, `family:name`, has been introduced. Names are as before, families are as follows:

```
inet:     Internet host
dnet:     DECnet host
nis:      Secure RPC network name
krb:      Kerberos V5 principal
local:   contains only one name, ""
```

The old-style syntax for names is still supported when the name does not contain a colon.

8.2.26.2 xrdb

Many new symbols are defined to tell you what extensions and visual classes are available.

8.2.26.3 twm

An interface for setting client priorities with the SYNC extension has been added.

8.2.26.4 xdm

There is a new resource, choiceTimeout, that controls how long to wait for a display to respond after the user has selected a host from the chooser.

Support has been added for a modular, dynamically-loaded, greeter library. This feature allows different dynamic libraries to by loaded by `xdm` at run time to provide different login window interfaces without access to the `xdm` sources. The name of the greeter library is controlled by another new resource, greeterLib.

When you log in through `xdm`, `xdm` will use your password to obtain the initial Kerberos tickets and store them in a local credentials cache file. The credentials cache is destroyed when the session ends.

8.2.26.5 xterm

The `xterm` terminal emulator has been minimally internationalized to use the Xlib built-in input method with 8-bit character sets.

There is now support for a few escape sequences from HP terminals, such as memory locking. The termcap and terminfo files have been updated to reflect this.

The logging feature of xterm has been removed. This change first appeared as a public patch to Release 5.

8.2.26.6 xset

The screen saver control option has two new suboptions to immediately activate, or deactivate, the screen saver:

```
xset s activate
xset s reset
```

8.2.26.7 imake

The command line option `-C filename` has been added to `imake`. This option specifies the name of the .c file that is constructed in the current directory. The default is Imakefile.c. Threads.tmpl will be added for multithreaded rules.

8.2.26.8 xsm

The `xsm` session manager has many enhancements. Advanced signal handling in `xsm` appears for the first time in release 6.1. The session manager `dtsession` from CDE is the default for AIX.

8.2.26.9 xmh

The `xmh` mail reader is now session-aware.

8.3 Motif Version 2.1

This chapter provides an overview of all the changes in Motif 2.1 with respect to Motif 1.2 (and Motif 2.0). Compatibility with CDE/Motif 1.2 was given great emphasis in this release, even at the expense of compatibility with OSF/Motif 2.0. Some OSF/Motif 2.0 applications may experience problems because of the changes.

The next section reviews the features most visible to a desktop end-user, the new widgets. The following sections concern the toolkit and detail the extensibility framework, new features in the toolkit, namely Uniform Transfer Model (UTM), reorganization of the menu system, as well as miscellaneous enhancements.

We also describe how the UIL technology has been enhanced to support the extensibility framework and how it has become architecture-neutral with regards to the underlying operating system and computer processor.

A load-only version of the Motif 1.2 libraries is shipped in the X11.motif.lib fileset. The Motif libraries shipping on AIX 4.3 contain the Motif 1.2 and 2.1 shared

objects, so the default installation of Motif provides versions of Motif for existing applications (Motif 1.2) and for development of 64-bit and threaded Motif applications (Motif 2.1). However, there is no binary compatibility path for Motif 1.2 to Motif 2.1 since the shr4.o shared object shipping in libXm.a is Motif 1.2, and not Motif 2.1.

8.3.1 New Widgets

To match the function of other GUIs, OSF issued a request for widgets in July 1992. The widgets that were selected through this process are a multiple-font text widget (from Digital), a Container and a Note Book (both from IBM), and a combo-box also called drop down list, from Lotus. Finally, OSF also included a thermometer-like scale and a spin box widget. A graphical example of all the new widgets is presented in Figure 47.

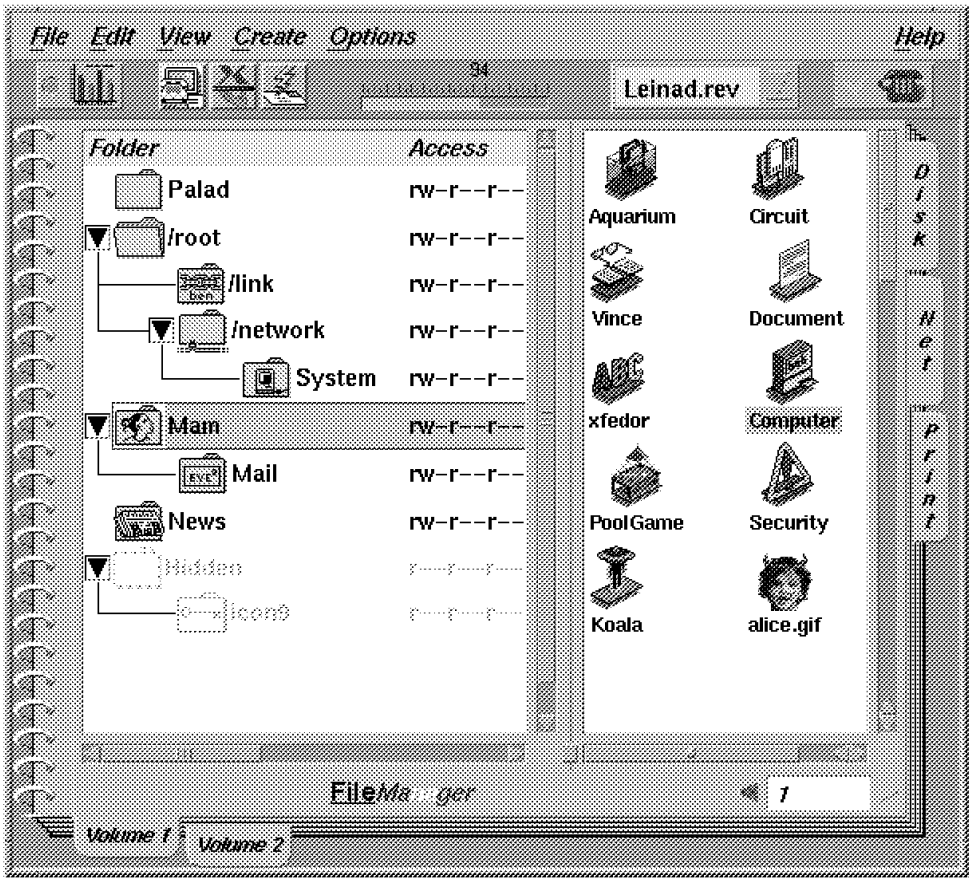


Figure 47. Example of the New Motif Widgets

8.3.1.1 Container Widget

The Container is a critical widget for the direct manipulation, object-oriented world toward which application software is moving. A container object offers views of the objects that the application can handle. Through direct manipulation, the user can select, move, copy, or delete these objects, drag them into other applications, or drop new objects into the container. The Container widget is a very powerful tool for application writers. Potentially, every application can be implemented with a Main Window and a Container. The menu bar of the Main Window specifies the generic operations on objects, and the container displays

the objects. A double click on an object activates it, and pop-up menus can be available for specific operations on the objects.

A Container can contain hierarchical objects that can be decomposed into further objects. The Container widget offers three kind of views:

- An icon view, in which every object is represented by an icon.
- A hierarchical view, also called outline or tree view, which displays the object tree.
- A detail view, a tabular view where every object appears as a line in a table and properties of the object are listed in the columns.

Icon View A container may display the objects in the typical large icon view. In this view, every object (a top level object of the tree) is displayed as an icon with a label. The Container widgets support different types of layout for those icons. The most popular are:

- Grid mode, where each icon fits into a grid cell.
- Free mode, where the user can freely move the icons around.

As an illustration, see the right side of Figure 47 with the Aquarium and the Circuit icons.

Tree View In the tree view, every object at the top level of the hierarchy is displayed with a small icon. If the object has sub-objects attached, the hierarchical organization is displayed with lines and indentation reflecting the tree structure. This can be seen on the left side of Figure 47 with the folder structuring.

For very large hierarchies that would take too much space, the Container widget offers a browsing facility. For each level of the hierarchy, the application can first decide whether or not the information below that level should be displayed. Secondly it can tell the container whether or not the user should be allowed to browse further.

If the user is not allowed to explore that branch of the tree, a mini-icon is displayed called the collapsed pixmap.

If the user is allowed to explore that branch of the tree, a mini-icon is displayed called the expand pixmap. Clicking on the expand pixmaps displays one more level in the hierarchy. As this new level is displayed, the mini-icon is turned into a collapse pixmap. When you click on the icon again, that tree branch is removed from the view.

Detail View The detailed view displays a table where each row represents an object, and each item in the row is a property of that object. The application is responsible for providing the detail data and the rendition attributes for it to be displayed.

8.3.1.2 Note Book

The Note Book is another powerful widget, simulating a real notebook that allows the application to display one page among a stack of pages, maintaining a constant page size. The Note Book widget includes pages, tabs, a status area and the page scroller. It stacks its page children so that all page children occupy

the same area just like real-world book pages. Tabs simulate notebook tabs. Major tabs are used for dividing pages into several sections, and minor tabs are used for subdividing the sections. The page scroller is used for switching the current page back and forth. The Note Book also provides tab scrollers for scrolling major and minor tabs when it cannot display all tabs. Tab scrollers are visible and enabled only when there is insufficient space to display all the major tabs or the minor tabs appropriate to the page. A complex implementation of a Note Book can be seen in Figure 47.

Major tabs displayed on the side of the Note Book allow the user to quickly access data. Once a primary tab is chosen (for example, a chapter), the minor tabs appear in the orthogonal direction (for example, sections) and allow further indexing inside the relevant parts of the document.

Pages, tabs, status areas, and the page scroller are created by the application as children of the Note Book widget. Any Motif widget may be a page of the Note Book. A major tab, or a minor tab, may be attached to a page by creating a tab child of the Note Book and setting a constraint to the page number of the targeted page. Note that the notebook widget makes heavy use of the traits of its child widgets to invoke their class methods. A tab, either a major tab or a minor tab, must be a Motif widget with a trait that can be activated (such as a push button). Tabs in a Note Book are associated with a page number not attached to any actual page widget. Therefore, it is possible to have a tab with an empty page. Destroying a page widget leaves an empty page. It does not tear the page out of the XmNotebook.

The page scroller child is not associated with a certain page. There is only one valid page scroller per Note Book, and it must carry the navigator trait. Since the application of the Note Book can provide page numbers, it is possible to have duplicate pages and empty pages. An empty page is a page slot where no page is inserted. This page displays just a blank background unless the application provides some visual information to this area while processing. Note that this feature is very useful for applications that have to display hundreds (or even thousands) of pages. It is not necessary to have hundreds of physical pages in memory. An application may actually use only one physical empty page, provide adequate tab indexing for the real number of pages it supports, and only update the contents of the (single) physical page with the information when required.

The Note Book widget is a versatile tool. Typical uses are:

- The display of on-line documentation, with quick access to information by using the tabs index.
- Reducing the real estate occupied by an application on the screen by making visible only the information required at that particular moment. A Note Book allows a developer to very quickly implement any kind of agenda or calendar application. It is the optimal base for a card filer application and provides good support for a hypertext applications.
- Dialog boxes that have variants. For example, a property sheet dialog box for a document editor might have a font part and a margin part as pages of the Note Book.

A Note Book allows a developer to very quickly prototype all kinds of applications that display organized information, such as a card filer, an agenda, or a calendar application.

8.3.1.3 Combo Box

As its name indicates, Combo Box combines the capabilities of a single line text widget and a list widget that allows users to type in information and also provides a list of choices to complete the text entry field. The list can either be displayed at all times or dropped down by the user. That is why Combo Box is sometimes referred to as *drop-down list*. An example of a Combo Box is shown on the top right of Figure 47 with the label Leinad.rev.

When the list portion of the Combo Box is hidden, users are given a visual cue, a downward-pointing arrow next to the text field. These drop down Combo Boxes are useful when presentation space is limited or when users will complete the text entry field more often by typing text than by choosing the entry field text from the list. The drop down Combo Box list should pop-up in the visible area of the screen but stay aligned with the text field (that is, at the bottom-edge of the screen, the list will be *up*).

The application programmer provides an array of strings that will fill the list. Each string becomes an item in the list, with the first string becoming the item in position one, the second string becoming the item in position two, and so on. The list can actually be manipulated with the regular API of the XmList widget.

Similarly, the text entry may be accessed by using XmTextField API. The size of the list is set by specifying the number of items that are visible in the list (XmNvisibleItemCount). If the number of items in the list exceeds the number of items that are visible, a vertical scroll bar will appear that allows the user to scroll easily through a large number of items.

A Combo Box allows a single item to be selected in two ways: by scrolling through the List and selecting the desired item, or by entering text into the text entry field. Selecting an item from the list causes that item to be displayed in the text portion of the Combo Box. The single-line text field in a Combo Box can be either editable or non-editable. If the text field is editable, the user can type directly into it to enter a new list item. If the application needs to validate the entered text, it can do so by installing a XmNmodifyVerifyCallbacks on the text field. If the text field is non-editable, typing text invokes a matching algorithm that attempts to match the entered text with items in the list.

8.3.1.4 Spin Box

Although more modest, the Spin Box widget offers a quick way of selecting, cycling, or setting a value within a range. It can be used by a very large range of applications. The Spin Box function can be implemented using the existing Motif Arrow Button widget and a label widget. However, the Spin Box widget makes it much easier for the developer and uses much less memory.

8.3.2 Motif Changes in Behavior

The following changes have been made to the behavior of Motif 2.1:

- Take focus without activating on Ctrl+Btn1 in List and Button widgets.
- Supports an indeterminate state and associated visual in the existing toggle button and toggle button gadget. The indeterminate state is useful in application property sheets.

- Snap-back scrolling, a feature that allows users to have the scroll bar slider snap-back to its starting position when the mouse is dragged beyond a certain distance from the scroll bar area.

8.3.3 The Motif Extensibility Framework

One of the problems Motif developers are facing is the complexity of developing new widgets for their applications. Subclassing Motif widgets without the source code of the parent class is a challenge with Motif 1.2. Even developing a new class from the superclass source code is not always easy. There are four sets of issues for the widget developer:

- Developing a new widget from scratch can take a significant amount of time. For its internal widget development, OSF built a library of internal functions that are commonly used in the widget set. For example, there are functions to draw 3D shadows. If those functions were available to widget writers, it would speed up the development process.
- The Motif widget class methods are currently not documented. Widget writers have to browse into the superclass source to figure out the class methods and what they do. Documenting the widget class methods of the most frequently subclassed widgets helps widget development.
- Sometimes, it is not possible to have new widgets behave as expected when they are managed by a standard parent. One of the key issues is that, in a number of cases, manager widgets perform tests about the nature of their children. These tests are typically done by testing the class pointers of the children. When applied to a custom widget, they fail, and the widget does not behave properly.
- Motif subclassing requires use of the Xt object framework, implemented in C. Most C++ application programmers, now a significant number, want to write subclasses of the Motif widgets directly in C++.

OSF has addressed these issues by developing an extensibility framework for Motif that will significantly help OSF/Motif widget developers and application programmers. This framework consists of four parts:

- A new mechanism has been developed on top of intrinsics, called *Traits*.
- A new set of APIs has been documented by OSF for widget writers.
- A new book hs been published by OSF with documentation on how to correctly subclass a Motif widget. This documentation is accompanied by a CD-ROM containing source code illustrations. The book also contains documentation for the existing class methods that can be re-used by developers.
- A set of C++ base classes has been developed that makes it possible to derive subclasses of the Motif Manager and Primitive widgets (the most frequently derived) directly in C++ while retaining the ability to use Motif functions on those widgets, such as keyboard traversal function XmProcessTraversal().

8.3.3.1 Traits

The trait abstraction was first introduced by Xerox. A trait is a characteristic of an object that can be expressed by a set of methods/data applied to, or carried by the object holding that trait. At a higher level, a trait implements a behavior that can be shared by different objects of a system without requiring any particular

hierarchical relationship between these objects or any particular implementation of the trait.

For example, consider the *I am a navigator* trait. This trait characterizes the fact that an object can set a value between a minimum and a maximum limit, regardless of the particular method used to set the value or the particular look and feel. A navigator can be implemented by a scroll bar, a thermometer-like control, or even a choice in the list of all possible values. The navigator provides the abstraction that guarantees that any object carrying this trait can be used to navigate arbitrarily between two values.

The I am a navigator trait implements methods that make it possible for clients of the object carrying the trait to set and get the min/max/current value independently of the implementation.

In Motif 2.1, both the XmScrollBar (an XmPrimitive subclass) widget and the XmSpinBox (an XmManager subclass) hold the I am a navigator trait, and they can both inherit, or specialize, the methods of that trait.

Therefore, the traits mechanism makes it possible to implement a form of multiple inheritance on top of the Xt object oriented framework, which only supports single inheritance. In the Motif 2.1 model, traits are seen as light abstract classes, sometimes called mix-ins.

Traits also make it possible to implement a form of polymorphism on the various widgets. For example, if a trait has a setvalue method, it can be used to set values of resources that actually have a different name. For example, a trait::setvalue() replaces XtSetValues() on multiple XmNresource names.

A side effect of traits is the potential reduction of code size in applications. In many cases, a Motif Manager widget would check the class of its children. Doing so requires access to the class pointer and at link time. Linking the class pointer global variable usually links chained modules in the application. Because those tests are replaced in the manager code with checking the children's traits, the spaghetti effect is eliminated.

Traits also augment code re-use inside Motif. A widget writer can simply decide to inherit a trait from another widget. It then inherits the class methods implemented by the OSF engineers at no cost.

8.3.3.2 Uniform Transfer Model (UTM)

OSF/Motif 1.2 provides the application programmer and widget writer with a number of different mechanisms to interchange data between applications. There are four forms of data transfer supported within Motif:

- Primary transfer. When the user clicks on pointer button BTransfer, the data currently selected (possibly in another application) is immediately transferred at the pointer location.
- The clipboard. This is probably the most widely-known mechanism. When the user activates the cut key (or the cut command into the Edit menu) the selected data is cut into the clipboard. Activating the paste key inserts the data at the current location of the destination cursor.
- Drag and drop. The visual metaphor for data exchange. The user can select data, drag it over the screen and drop it onto a recipient.

- Secondary transfer, also referred to as quick-copy, where the user first selects a destination and then uses Alt-BTransfer to select data that is moved to the destination by releasing the pointer button.

In each case, a data transfer can be characterized by the fact that:

- Some data is owned by an application, called the source.
- Another application, called the destination, wants to acquire the data. To acquire the data, the destination usually sends a request to the owner. The destination is also called the requestor.

Each particular transfer mechanism in Motif requires a new programming effort to accommodate, and thus, there is a burden on the programmer if all modes are to be supported, as recommended by the *Motif Style Guide*.

In Motif 2.1, OSF has implemented a new mechanism to unify access to the different mechanisms. Moreover, this mechanism allows the application programmer the ability to extend the function of toolkit-supplied widgets in the area of data interchange.

The model is very simple:

- On every widget that can act as source emit data, a convertCallback is available.
- On every widget that can act as a destination (receive data), a selectionCallback is available.

The convert callback is called each time the user has requested data through any mechanism. The programmer only needs to provide a buffer containing the data and the name of the data type (an atom). Motif automatically transfers the data and frees the memory after the data has been received. Similarly, the selection callback tells the programmer that data has arrived.

8.3.3.3 Menu System Improvements

Motif pop-up menus were never simple to program with Motif 1.2. When a pop-up menu was created, it did not pop-up automatically on the screen when a button was pressed, some programming was required to make it pop-up. It had the sense of a task left incomplete, particularly since the code would pop-up a menu when invoked from the Menu key of the keyboard.

In Motif 2.1, the following was implemented:

- Programmers can freely install pop-up menus on arbitrary widgets in the tree. For each menu, it is only necessary to specify whether the menu is only valid for the widget it is installed upon or for children of that widget as well.
- The menu system automatically pops-up the appropriate pop-up menu when the user of the application presses the appropriate button within the application.

For the most simple specification of an automatic pop-up, nothing else needs to be done by the application programmer other than creating the pop-up menu associated with its manager.

8.3.4 Miscellaneous Enhancements

The following sections describe various Motif enhancements:

8.3.4.1 Printing

Starting with Release 2.1, Motif includes support for printing using an X protocol-based print server. This print server produces output in three formats: PCL, PostScript, and Raster.

8.3.4.2 Thread-Safe Libraries

Xm and Mrm are thread-safe-enabled. This means that the libraries themselves are thread-safe, and a multithreaded application need not do explicit locking when accessing these libraries.

8.3.4.3 File Selection Box

The Motif File Selection Box offers a powerful, but complex, filtering mechanism for file selection. It offers all the power of the UNIX file system to users. The CDE User Model group, however, has come up with a design that seems better-suited for the average user.

The CDE File Selection Box replaces the regular filter control with two controls: the base directory entry and a wild-card entry for file name pattern. Only the subdirectories are shown in the directory list and the file names in the file list. Hence, scroll bars are not needed in this case.

OSF did not want to make obsolete the documentation for existing applications. They also wanted to maintain the old Motif 1.2 behavior. For these reasons, they decided to maintain both the old and the new styles. However, the new CDE style has become the default since it is more intuitive for most users.

8.3.4.4 String Manipulation

Motif offers string manipulation through an abstract object called XmString. The XmString abstraction associates a multi-lingual string with the encoding information attached to each locale and rendition to be used for a particular substring.

In Motif 1.2, the XmString abstraction supports the display with multiple fonts. However, the programmer has to separately create each character string that uses a different font, associate that string with a tag, and match that tag with a font. In Motif 2.1, the rendition is both simplified and extended. It is extended to support multiple fonts as well as multiple colors and TAB marks. It is simplified with the availability of a new API. This new API offers construction of the XmString.

8.3.4.5 Toggle Button Enhancements

When used as check boxes, toggle buttons (on/off buttons) are often not intuitive in Motif 1.x releases. Users are sometimes confused whether the etched-in appearance means on or off. The 3D effect used for the etched-in look is not always easily recognized in unusual light conditions (for example, industrial or military environments) or screens with limited color range. This has lead to enhancing the ToggleButton widget to support these new features:

- A check mark can be used inside the toggle square area.

- Since the check mark sign is particular to the US, for other cultures check boxes can be crossed instead.
- In addition to the regular button colors, two additional colors can be used to indicate On or Off.
- In radio boxes, a circular shape can be used instead of the diamond shape.

An example of the new toggle buttons are shown in Figure 47 in the upper-left and right corners.

8.3.4.6 Support for Right-to-Left Layout

Support has been added for languages written from right to left. Many of the Motif widgets now automatically reverse the geometry when the direction is right-to-left. For example, the Form widget automatically switches the left margin and the right margin, as well as the attachment constraints. The result is that developers can design applications using the internationalization facilities and specify a constrained layout that will work in both left-to-right and right-to-left environments.

8.3.4.7 Support for XPM Format

Motif 2.1 requires that applications are linked with the Xpm library. This library is freely available from the /contrib directory of the X Consortium and is also duplicated on the Motif tape. Xpm support makes it possible to support multi-color pixmap icons instead of two-color bitmaps.

8.3.4.8 Vertical Paned Window

The Motif 1.x Paned Window widget can only have horizontally-separated panes. Motif 2.1 adds a vertical mode.

8.3.4.9 Unit Conversion

New conversion methods have been added so that a user can specify a window size in inches, millimeters, or typographical points.

8.3.4.10 List Enhancements

When a selection is made from XmList, a new resource specifies whether XmList takes ownership of the primary selection. A keyboard navigation facility was added allowing the user to navigate directly to an item by typing the first character of that item. This is useful in lists organized in alphabetical order.

The Motif 1.2 List widget does not exhibit very good startup performance with large lists. An optimization was found in the algorithms and memory allocation strategy that reduces the startup time of a List widget by more than 30 percent on a typical workstation.

8.3.4.11 XmScreen Enhancements

Several resources have been added to the XmScreen widget to allow per-screen behavior.

A new resource is available to specify whether pixmaps or bitmaps should be used in an application. If bitmaps are used, the application is limited to two-color icons, or stipples, and it uses much less memory (typically eight times less on 8-bit color screens).

A set of resources was added to enable control of the color schemes by the application. The goal of these new resources makes it possible for an application to:

- Override the color calculation method used by Motif to generate 3D effects.
- Allocate the pixels in the colormap.

Color-intensive applications generally want to control the color schema and color allocation to optimize their own algorithms and use of the colormap. For example, on a system with 256 entries in the colormap, an application may want to reserve all 256 colors for its own usage. There are then no colors left for the Motif toolkit. With Motif 2.1, the application can plug-in its own allocation function. Then, each time Motif needs a new color, it calls that function. The application can then *loan* to Motif one of the colors it is already using.

A new screen resource allows an application to customize the bitmap that is used by Motif for the rendering of insensitive visuals. Support is also added for applications that want to display menus in overlay planes. Menus can now be displayed in a visual that is different than that of the application.

8.3.4.12 Virtual Bindings

As of Motif 1.2, users can customize the keyboard mapping by defining a keymap between virtual keys (for example, osfCancel) and real keys (for example, Cancel or Escape). They then run the `xmbind` command to bind the real keys to the virtual keys. However, a single real key can only be assigned to a single virtual key.

In Motif 2.1, each virtual key can be associated with multiple real keys, allowing users to use, for example, both the Escape key or the Cancel key to cancel an operation. Of course, errors can occur if a user incorrectly maps the same real key to multiple virtual keys.

8.3.4.13 Drag and Drop Enhancements

Several enhancements have been made to Drag and Drop. The following features are useful for sophisticated applications that are advanced in drag and drop technology.

A function has been added to query if a widget is a dropsite or not. A generic application-wide dragStartCallback has been added, so an application can be aware and take action for any drag occurring at any time in the application.

To provide better drag and drop feedback to the user, Motif 2.1 makes it possible for the drag icon to change in real time to show the effect that a drop would have if it occurred at the current location of the mouse cursor. The effect depends on the individual item, or items, that are being dragged.

Another enhancement that has no API, but is fairly visible to end users, is drag-scroll support. Users of small screens are often faced with the problem of not being able to drag data from one place and drop it in another because the target window is currently scrolled out of view. The user has to drop the icon on the desktop, scroll the window so that the destination is visible, and restart the drag. Drag scrolling prevents this, allowing the user to move the cursor over the scroll bar direction control and pause. After a customizable delay, the scrollable window automatically begins to scroll and makes the destination appear. As soon as the user moves the cursor, scrolling stops.

8.3.4.14 Scrolled Window and Scroll Bar Enhancements

The Scrollbar widget has been extended to support a look and feel like OpenStep, where both up and down controls are close together at one end of the scroll bar, as opposed to one control at each end. A new resource was added to give applications more control over the appearance of the slider.

It is practically impossible with Motif 1.2 for an application to implement a scrolled window that has a fixed title or a status bar inside the scrolled window area. Motif 2.1 makes it possible to have one or more arbitrary controls be displayed as non-scrollable in any direction.

Using this feature, users can simply implement a spreadsheet application with a column title that cannot be scrolled out vertically, a rows title that cannot be scrolled out horizontally, and a RowColumn widget for the cells. MainWindow, a subclass of ScrolledWindow, inherits this feature too.

8.3.4.15 Drawing Area

The drawing area is now traversable with the keyboard, making it possible for users to traverse to a drawing area and get its associated functions (for example, pop-up menus).

8.3.4.16 Performance Enhancements

A number of performance enhancements have been made and are described below.

XmString Performance

The memory used by an XmString that uses a single font and is smaller than 256 bytes (256 characters in Latin-based language environments) is less than it used to be. Note that this kind of string represents the majority of strings displayed in a typical user interface, for example, menu items, list items, and labels. The memory size of every XmString has also been reduced by four bytes (from Motif 1.2). Every string that consists of less than 256 characters uses a single encoding, single color, and single font.

Copying XmString is now managed by reference counting in 2.1, which makes XmString copy much faster and uses no memory. Since Motif does a lot of XmString copies internally, this should reduce the memory size used by many applications that do not release the memory of an XmString after they have passed it to a widget.

List Performance

The startup time for the List widget has been decreased compared to Motif 1.2. This is particularly noticeable for applications that handle large lists of several hundred items. See Section 8.3.4.10, “List Enhancements” on page 222.

X-Related Performance

In general, the amount of resources allocated in the X server, clients, and the number of round trips between client and server has been decreased.

Client-side memory has been decreased by using bitmaps instead of pixmaps whenever possible. In Motif 1.2, when a widget uses a background pixmap, a pixmap is allocated using the available depth of the screen. In Motif 2.1, a bitmap is used if only two colors are required.

Motif 2.1 has been optimized to generate fewer mouse events from the X server when using gadgets.

8.3.4.17 UIL Extensibility and Portability

Since OSF introduced an extensibility framework, it was necessary to reflect that in the UIL language. Once a developer has developed a new widget, that new widget should be usable from UIL as well. However, they should not have to generate a new compiler for every new widget supported. It was, therefore, necessary for OSF to develop a mechanism such that the UIL compiler could dynamically incorporate new widgets to be accepted in UIL source files and generate the appropriate binary UID code.

In the past, OSF received numerous enhancements requests because the UID files in the 1.x releases were not architecture neutral. You had to compile the same UIL source file for every target platform. Moreover, only 32-bit platforms were supported.

With OSF/Motif 2.1, OSF developed a new UIL technology that reaches those three objectives simultaneously:

- The UIL compiler can be told to dynamically accept new widgets.
- The UID files generated by the compiler are architecture neutral.

Note that the architecture neutral format is still dependent on 32-bit versus 64-bit architectures. A UIL file compiled on a 32-bit architecture can be read on any other 32-bit architecture but not on a 64-bit architecture. Similarly, a UIL file compiled on a 64-bit architecture can be read on any other 64-bit architecture but not on a 32-bit architecture. This design decision was made because the memory used for 64-bit architectures is about twice the size of that used on 32-bit, and 32-bit platforms should not have to bear a 100 percent penalty.

8.3.5 Compatibility with Motif 1.2 and 2.0

As stated at the beginning of this chapter, compatibility with CDE/Motif 1.2 was given greater emphasis than compatibility with OSF/Motif 2.0. The following issues should be noted concerning compatibility with Motif 1.2:

- If a customer application subclasses a widget in the group of DrawingArea, Label, List, Manager, or Primitive, there is a potential for binary incompatibility. The Class Records for these widgets changed from Motif 1.2 to Motif 2.0. The only resolution in this instance is to re-compile, unless the application writer used XmResolvePartOffsets (now known as XmeResolvePartOffsets in the *Motif 2.1 Widget Writer's Guide*).
- The XmString definition has changed in Motif 2.1. It is now a union instead of a typedef of char *. Existing Motif applications running on Motif 2.1 may have problems depending upon their usage of XmString.

Some OSF/Motif 2.0 applications may experience problems because of the following changes:

- The XmCSText widget has been withdrawn, as have those APIs added to OSF/Motif 2.0 solely to support it.
- Mrm support for word size-independent .uid files has been removed. Existing .uid files compiled with Motif 2.0 UIL may not be readable. As in OSF/Motif 1.2, .uid files are portable only between machines with the same word size.

- New XmComboBox XmNpositionMode and XmSpinBox XmNpositionType resources default to incompatible index values and should be forced by all applications using these widgets. XmONE _BASED is recommended for XmComboBox widgets because it lets applications distinguish between new values entered in the text field and the first item in the list.
- XmStringCreateLocalized now handles new lines and tabs.
- The _XmStrings array has, on some machines, been split into multiple subarrays with the same techniques used by libXt. This preserves compatibility with Motif 1.2 and permits future expansion.
- Labels for automatically-created subwidgets, like the buttons in a file selection box, are now unconditionally localized and may not be set or overridden by the user.
- The XmDisplay XmNenableThinThickness resource now has wider effect than it did in Motif 2.0.
- The XmDisplay XmNenableToggleVisual resource now changes the way XmNindicatorOn and XmNindicatorType values are rendered, instead of simply changing their default values. Motif 2.0 applications that called XtSetValues() for these resources may notice a change. New constants have been added to obtain the old behavior.
- In Motif 2.0, there were two distinct XmREPLACE constants with different values. The XmMergeMode constant has been renamed XmMERGE_REPLACE. This is a source compatibility issue; binary compatibility is unaffected.
- XmDisplay XmNdragReceiverProtocolStyle default value has been reverted to XmDRAG_PREFER_PREREGISTER. Users may find that XmDRAG_PREFER_DYNAMIC is more efficient.
- The XmNenableEtchedInMenu resource causes buttons and toggles in menus to be rendered with different colors than those in earlier releases.
- XmScrolledList and XmScrolledText scroll bar colors are computed differently. They are now derived from the scrolled window's background color, not the color of the XmList or XmText widget.
- To promote convergence with dtwm, mwm's panning, and virtual screen support has been removed, as has mwm's support for workspaces.
- The XmCxx library of C++ wrappers has been moved to the demos/lib directory.

8.4 X Virtual Frame Buffer (4.3.2)

The X Virtual Frame Buffer (XVFB) software technology was first introduced in AIX 4.3.1 and has been further enhanced in AIX 4.3.2 to support the RS/6000 SP.

The XVFB provides a virtual graphics adapter that allows the X Windows Server to start and operate on a server machine that has no physical graphics adapter. In addition, the OpenGL 3D Graphics rendering library can be used with the XVFB technology.

This capability was introduced primarily to provide improved support for 3D Rendering Server Applications. By removing the single shared resource (graphics adapter) from the system, and replacing it with a dynamic resource (system

memory), 3D Rendering Server Applications can scale in performance as processors are added to an SMP capable system. Other advantages of XVFB include:

- Less expensive 3D Rendering Server (since no graphics adapter or display is required)
- Better security (since the 3D Rendering Server does not have to be left logged in, XVFB runs in the background)
- Better scaling with SMP machines (since each process gets a private XVFB rendering area)

8.4.1 Direct Soft OpenGL

Direct Soft OpenGL (DSO) is a new IBM OpenGL rendering technology for AIX and RS/6000. Implemented to work with the XVFB, DSO was designed specifically to enhance the performance of 3D Rendering Server Applications by eliminating extraneous interprocess communication, eliminating process context switching overhead, and making rendering and image reading more direct and efficient.

DSO is a pure software implementation of OpenGL that runs as a *direct* OpenGL Context. Basically this means that all of the CPU intensive OpenGL work (3D rendering) is part of the application process (not part of the X Server process). By running *direct*, all of the interprocess communications with the X Server are eliminated, making 3D rendering much more efficient. In addition, the AIX operating system is not having to context switch between the X Server and the 3D rendering applications, making system utilization more efficient.

Using the XVFB and DSO software is as simple as installing the XVFB file sets (X11.vfb and OpenGL.OpenGL_X.dev.vfb) and starting the X server with the appropriate options (for example, X -vfb -x GLX -x abx -x dbe -force).

8.4.2 CATweb Navigator and XVFB/DSO

CATweb Navigator allows end users with Java Enabled Web Browsers to view and navigate product information created with CATIA Solutions. By using the intuitive set of Java Applets that make up the CATweb Navigator Client, users can connect to a CATweb Server machine, select models for viewing, and then view and navigate the models with a 3D viewer, 2D schematic viewer, or a report style viewer.

One CATweb Server machine can support multiple concurrently active CATweb Navigator Clients. For each CATweb Client that attaches to the CATweb Server, one or more CATweb processes will be started on the CATweb Server to handle the requests of that particular CATweb Client. One of these CATweb processes is a 3D rendering application. This application runs on the CATweb Server and is responsible for:

- Loading the requested CATIA model
- Rendering the 3D model on the fly as the CATweb Client requests new views
- Compressing the final rendered image
- Transferring the image to the Java CATweb Client

This application uses both the X Windows and OpenGL libraries to quickly and accurately render 3D images In addition, CATweb Navigator V2.0 and above has been enhanced to effectively exploit the capabilities of XVFB and DSO.

8.5 OpenGL Enhancements

The following section describes the enhancements to OpenGL.

8.5.1 OpenGL 64-bit Indirect Rendering (4.3.1)

In AIX 4.3.1, OpenGL has added 64-bit support for indirect rendering. To use this function, a new level of the xlc++ compiler is required. See /usr/lpp/OpenGL/README for more information.

8.5.2 OpenGL Performance Enhancements (4.3.2)

IBM OpenGL Versions 1.1 and 1.2 are enhanced to improve performance in several areas. Users of uniprocessor systems will note faster drawing of primitives under most conditions. All users should see improvements in:

- Throughput and cache utilization
- Latency when lighting is enabled
- Overall performance on any primitives using the new MultiDrawArray extension

8.5.3 OpenGL Version 1.2 and ZAPdb (4.3.2)

OpenGL Version 1.2, released on March 16, 1998, is the second revision since the original Version 1.0. OpenGL within AIX Version 4.3.2 is enhanced to support OpenGL Version 1.2. Several additions were made to the graphics library (GL), especially to the texture mapping capabilities and the pixel processing pipeline. Following are brief descriptions of each addition:

- Three-Dimensional-Texturing

  Three/dimensional textures can be defined and used. In-memory formats for the three-dimensional images, and pixel storage modes to support them, are also defined. One important application of three-dimensional texture is rendering volumes of image data.

- BGRA Pixel Formats

  BGRA extends the list of host-memory color formats. Specifically, it provides a component order matching file and framebuffer formats common on Windows platforms.

- Packet Pixel Formats

  Packed pixels in host memory are represented entirely by one unsigned byte, one unsigned short, or one unsigned integer. The fields with the packed pixels are not proper machine types, but the pixel as a whole is. Thus the pixel storage modes and their unpacking counterparts all work correctly with packed pixels.

- Normal Rescaling

  Normals may be rescaled by a constant factor derived from the modelview matrix. Rescaling can operate faster than renormalization in many cases, while resulting in the same unit normals.

- Separate Specular Color

  Lighting calculations are modified to produce a primary color consisting of emissive, ambient, and diffuse terms of the usual GL lighting equation, and a secondary color consisting of specular terms. Only the primary color is modified by the texture environment; the secondary color is added to the result of texturing to produce a single post-texturing color. This allows highlights whose color is based on the light source creating them, rather than the surface properties.

- Texture Coordinate Edge Clamping

  GL normally clamps such that the texture coordinates are limited to exactly the range [0,1]. When a texture coordinate is clamped using this algorithm, the texture sampling filter straddles the edge of the texture image, taking half its sample values from within the texture image, and the other half from the texture border. It is sometimes desirable to clamp the texture without requiring a border, and without using the constant border color.

  A new texture clamping algorithm, CLAMP_TO_EDGE, clamps texture coordinates at all mipmap levels such that the texture filter never samples a border texel. The color returned when clamping is derived only from the edge of the texture image.

- Texture Level of Detail Control

  Two constraints related to the texture level of detail parameter λ (lambda) are added. One constraint clamps λ (lambda) to a specified floating point range. The other limits the selection of mipmap image arrays to a subset of the arrays that would otherwise be considered.

  Together, these constraints allow a large texture to be loaded and used initially at low resolution and to have its resolution raised gradually as more resolution is desired or available. Image array specification is necessarily integral rather than continuous. By providing separate, continuous clamping of the λ (lambda) parameter, it is possible to avoid popping artifacts when higher resolution images are provided.

- Vertex Array Draw Element Range

  A new form of DrawElements that provides explicit information on the range of vertices referred to by the index set is added. Implementations can take advantage of this additional information to process vertex data without having to scan the index data to determine which vertices are referenced.

Because OpenGL 1.2 is a superset of OpenGL 1.1, all programs written for OpenGL 1.1 run on OpenGL 1.2 without modification, recompiling, or relinking. OpenGL 1.2 support is available in AIX 4.3.2 only on the GXT3000P PCI Graphics Accelerator. Users of other graphics adapters are limited to functions contained in OpenGL 1.1 and OpenGL 1.1 extensions. The optional Imaging Extension subset is not supported by the IBM OpenGL 1.2 implementation (at this time).

The ZAPdb interactive OpenGL debugger is enhanced to support all new features provided in OpenGL Version 1.2.

8.5.4 New OpenGL Extensions (4.3.2)

Three new extensions to OpenGL are available in AIX Version 4.3.2. For details on implementation, consult the user documentation. Brief descriptions of the new extensions follow as such:

- MultiDrawArray Extension

  Enables users to group together multiple primitives and send them to the adapter with one call. This is supported on all OpenGL-capable adapters except the GXT1000.

- Texture Mirrored Repeat Extension

  Gives users the capability of specifying texture maps without discontinuities at the edges. This is supported only on the GXT3000P.

- Color Blend Extension

  Gives users more options in creating blended and/or translucent colors without having to use an alpha buffer. This is supported only on the GXT3000P.

8.6 graPHIGS Enhancements (4.3.2)

IBM graPHIGS has been enhanced in AIX Version 4.3.2 to support the new high performance GXT3000P PCI Graphics Accelerator that attaches to the RS/6000 43P 7043 Models 150 and 260. Further performance improvements were made, and support for the new Euro symbol has been added.

8.6.1 Performance Enhancements

IBM graPHIGS within AIX Version 4.3.2 has been altered to improve the performance of polygon rendering under most conditions. Throughput to the graphics adapter has also been enhanced, often resulting in performance improvements for the rendering of all primitives.

8.6.2 Euro Symbol Support

The graPHIGS API offers several facilities for the display of text information:

- Geometric text and annotation text that allow an application to specify a character string that is to be displayed on a workstation.
- Echo of string device input that allows a user to see the input being entered on a string device (for example, a keyboard).

Your application can specify two attributes that affect the interpretation and display of text information: the character set identifier (CSID) and the font identifier. When the application specifies a character to be displayed, the CSID and font identifier together determine the symbol that is displayed to represent the character.

The application can specify a character string to be displayed in geometric text. The graPHIGS API, in processing that character string, accesses a character set file and a symbol file to get the information necessary to interpret and display the text:

- The character set file contains information about a particular character set. The contents of the character set file tell, amongst other things, the available

code points of the character set and the index that is used to locate the symbol in a symbol file. There is only one character set file for a particular character set identifier.

- The symbol file contains the drawing controls for each symbol that is displayed to represent the character. There is a symbol file for each available font within a particular character set.

IBM graPHIGS will use the same locales as AIX to support the Euro symbol. For details about the Euro symbol support through AIX locales refer to 10.5, “Euro Symbol Support for AIX (4.3.2)” on page 252. graPHIGS selects the appropriate code set and font support based on the value of the LANG environment variable:

- LANG=xx_XX

  The effective character set is given by CSID 10 - ISO 8859-1 (Latin 1) with the Euro in the x80 character position. The font related symbol file is afm0a01.sym, thus the graPHIGS default FONT 1 is used.

- LANG=xx_XX.IBM-1252

  graPHIGS does not refer to the explicitly stated code set in the name of the locale. That means that the code set designation IBM-1252 is ignored and the same conditions as LANG=xx_XX apply. The code set IBM-1252 is of particular interest for the Euro symbol support in conjunction with the single-byte migration option. This option provides customers with the opportunity to get Euro symbol support through a single-byte character set instead of using the Unicode locales. For further details, refer to section 10.5.6, “Euro SBCS Migration Option - IBM-1252 Locale” on page 270.

- LANG=XX_XX

  The effective character set is given by CSID 131 - Unicode with the Euro glyph in the standard U+20AC character position. The font related symbol file is afm8301.sym, thus the graPHIGS default FONT 1 is used.

- LANG=Xx_XX

  The effective character set is PC850, and no Euro symbol support is available.

Note: xx_XX, Xx_XX, and XX_XX represent the language and territory designation with respect to a given code set. For example, the national language support for German allows you to select between the de_DE, the De_DE, and the DE_DE locale.

The graPHIGS default font files for CSID 10 have added the Euro character at codepoint 0x80. If you have used these font files as a basis for your user defined font files, you will need to evaluate your use of the codepoint 0x80.

Chapter 9. Online Documentation

There have been several changes made to the way on-line assistance is provided in AIX Version 4.3. The main change is the move from InfoExplorer, that is essentially proprietary in nature, to a more generalized HTML-based system.

9.1 Documentation Search Service

The growing influence of the World Wide Web has made the Web browser a common element of users day-to-day work. For many, the Web browser is a core component of how they perform tasks. To meet this user need, InfoExplorer on AIX is replaced with the Documentation Search Service and HTML-based product documentation. InfoExplorer will no longer be part of the base AIX installation media; however, it will remain available as a separately orderable LPP to service the InfoExplorer libraries that still exist.

AIX 4.3 provides an optionally installable component called the Documentation Search Service. It allows you to search on-line HTML documents, such as the AIX product documentation. It provides a search form for use with a Web browser. When you enter key words into the search form, the Document Search Service searches for the words, and then presents a search results page that contains links that lead to the documents that contain the target words.

The Documentation Search Service contains all of the features that you would normally expect to see from InfoExplorer, such as:

- List of Books
- Commands Reference
- Programming Reference
- Search Facility

9.1.1 Installation of Documentation Search Service

You can set up one of your AIX systems in your organization to be the documentation server and all other systems as documentation clients. This will allow documentation to be installed only on one system and all other systems can access this system without needing the documentation installed locally.

You need the following products and components installed for a complete set of services:

- For the client:
  1. A Web browser
  2. The bos.docsearch.client.* filesets (for AIX integration)
- For the documentation server (which may also act as a client)
  1. The entire bos.docsearch package
  2. The documentation libraries
  3. A Web browser
  4. A Web server

The browser must be a forms-capable browser and the Web server must be CGI-compliant.

If you are planning on integrating your own documentation on the documentation server, you will also need to build the document's indexes.

Except for the section 9.3, "Invoking Documentation Search Service" on page 237, you need root authority to perform the installation and configuration tasks

There are a variety of ways to install the documentation, Web server, and Document Search Service. You can use the Configuration Assistant TaskGuide, Web-Based Systems Management, or SMIT.

The easiest way for a non-technical user to install and configure Documentation Search Services, is by using Configuration Assistant TaskGuide. To run the Configuration Assistant TaskGuide, use the `configassist` command. Then select the item titled Configure Online Documentation and Search.

If you would rather install Documentation Search Services manually, you can use SMIT.

9.1.1.1 Installing the Web Browser

Use `smit install_latest` to install Netscape supplied on the AIX 4.3 Bonus Pack CD. Use `smit list_installed` to check whether you have the following filesets installed, as shown in Figure 48:

```
                                COMMAND STATUS

Command: OK            stdout: yes           stderr: no

Before command completion, additional instructions may appear below.

[TOP]
  Fileset                      Level  State  Description
  ----------------------------------------------------------------------------
  Netscape.msg.en_US.nav.rte
                             4.0.6.0    C    Netscape Navigator Runtime
                                             Messages - U.S. English
  Netscape.nav.rte           4.0.6.0    C    Netscape Navigator Runtime
                                             Environment


State Codes:
 A -- Applied.
 B -- Broken.
[MORE...4]

F1=Help             F2=Refresh          F3=Cancel           F6=Command
F8=Image            F9=Shell            F10=Exit            /=Find
n=Find Next
```

Figure 48. Netscape Filesets

If you are installing the Netscape browser from other sources or you are installing other Web browsers, follow the installation instructions that come with the software. Note that there will not be any records in the ODM if your product source is not in installp format.

9.1.1.2 Installing the Web Server

You may install any CGI-compliant Web Server. The Lotus Domino Go Webserver is used here. It is supplied on one of the AIX 4.3 Bonus Pack CDs.

The Documentation Search Service uses its own search engine CGIs. Therefore, you do not need to install the NetQ fileset, the Webserver Search Engine. The following shows the filesets installed (Figure 49):

```
                                COMMAND STATUS

Command: OK            stdout: yes           stderr: no

Before command completion, additional instructions may appear below.

[TOP]
  Fileset                      Level  State  Description
  ----------------------------------------------------------------------------
  internet_server.base.admin
                             4.6.2.5    C    Lotus Domino Go Webserver
                                             Administration
  internet_server.base.doc   4.6.2.5    C    Lotus Domino Go Webserver
                                             Documentation
  internet_server.base.httpd
                             4.6.2.5    C    Lotus Domino Go Webserver
  internet_server.msg.en_US.httpd
                             4.6.2.5    C    Lotus Domino Go Webserver
                                             Messages - en_US
[MORE...9]

F1=Help             F2=Refresh          F3=Cancel           F6=Command
F8=Image            F9=Shell            F10=Exit            /=Find
n=Find Next
```

Figure 49. Domino Go Webserver Filesets

If you are installing the Domino Go Webserver from other sources or you are installing another Web server, follow the installation instructions that come with the software. Note that there will not be any records in the ODM if your product source is not in installp format.

9.1.1.3 Installing Documentation Search Service

The Documentation Search Service is (at the time of writing) on Volume 2 of the AIX 4.3 installation CDs. Install the client portions for a client AIX image, or install the entire bos.docsearch package for a documentation server. These filesets prereq other filesets during the install (such as IMNSearch).

- bos.docsearch.client.Dt
- bos.docsearch.client.com
- bos.docsearch.rte

For the documentation clients, you need only a Web browser. Installation of the bos.docsearch.client fileset will give you the CDE desktop icon and the `docsearch` command. Refer to 9.3, “Invoking Documentation Search Service” on page 237 for further details.

Use `smit list_installed` to check whether you have the following filesets installed as shown in Figure 50:

```
                                COMMAND STATUS

Command: OK            stdout: yes           stderr: no

Before command completion, additional instructions may appear below.

[MORE...1]
  ------------------------------------------------------------------------
  IMNSearch.bld.DBCS         1.2.0.4    C    NetQuestion DBCS Buildtime
                                               Modules
  IMNSearch.bld.SBCS         1.2.1.3    C    NetQuestion SBCS Buildtime
                                               Modules
  IMNSearch.rte.DBCS         1.2.0.4    C    NetQuestion DBCS Search Engine
  IMNSearch.rte.SBCS         1.2.1.3    C    NetQuestion SBCS Search Engine
  IMNSearch.rte.httpdlite    1.1.1.1    C    NetQuestion Local HTTP Daemon
  bos.docsearch.client.Dt    4.3.2.0    C    DocSearch Client CDE Application
                                               Integration
  bos.docsearch.client.com   4.3.2.0    C    DocSearch Client Common Files
  bos.docsearch.rte          4.3.2.0    C    DocSearch Runtime
[MORE...9]

F1=Help             F2=Refresh          F3=Cancel           F6=Command
F8=Image            F9=Shell            F10=Exit            /=Find
n=Find Next
```

Figure 50. Documentation Search Service Filesets

9.1.2 Configuring Documentation Search Service

Use either `wsm` or `smit` to configure the documentation search service. If you used the Configuration Assistant TaskGuide to install and configure the Documentation Search Service, you will not need to perform any further configuration.

For `wsm`, double-click on Internet Environment icon, or you can use `smit web_configure` to configure the following:

- Default browser

  Type into the field the command that launches the browser that you want to be the default browser for all users on this computer, for example, `/usr/prod/bin/netscape`. This will set the /etc/environment variable DEFAULT_BROWSER to the string you type in.

- Documentation and search server

  You can define the documentation search server location to be:

  - None - disabled
  - Remote computer

    Type the remote documentation server name. The default TCP/IP port address is 80. Change it to the port address used by the documentation server.

  - Local - this computer

    If you are using Lotus Domino Go Webserver or IBM Internet Connection Server in the default location, all the default settings of the cgi-bin directory and HTML directory will have been filled in for you. If you are using other Web servers, or you are not using the default location, you have to fill in your cgi-bin directory and HTML directory that the Web server requires. You may change the port address used by the server. If you change the

port address, you have to use the same address for all your documentation clients.

9.2 Installing Online Manuals

You can either install the documentation information onto the hard disk or mount the documentation CD in the CD-ROM drive. Mounting the CD will save some amount of hard disk space, but it requires the CD to be kept in the CD-ROM drive at all times. Also, searching the documentation from the CD-ROM drive can be significantly slower (in some cases up to 10 times slower) than searching the information if it is installed on a hard disk. In addition, there are two documentation CDs:

- The AIX Version 4.3 Base Documentation CD
- The AIX Version 4.3 Extended Documentation CD

Use `smit install_latest` to install the on-line manuals onto the hard disk. The fileset bos.docregister is a prerequisite for all on-line manuals. It will be automatically installed the first time you install any on-line manuals even if you have not selected this fileset.

Note

The installation images located on the AIX Version 4.3 Base Documentation and Extended Documentation CDs do not contain the HTML files. These files exist separately on the CD to allow access from non-AIX platforms. Installing the images from the CD will work correctly; however copying the installation images by themselves to another location is not enough for a proper install.

9.3 Invoking Documentation Search Service

You must logout and login again after the Documentation Search Service has been configured so that you will pick up the environment variables set up during the configuration.

If you are running the CDE desktop environment, double-click the Documentation Search Service icon in the Application Manager window.

Alternatively, you can use the command `docsearch` to invoke the documentation search service. Netscape will start and you should see the Documentation Search Service page.

You can invoke the Documentation Search Service without installing the docsearch client component. In fact, you do not even need to invoke the documentation search service from an AIX machine. You can do this by first invoking the browser and enter the following URL:

```
http://<server_name>[:<port_number>]/cgi-bin/ds_form
```

This URL points to a global search form on the document server where the name of the remote server given in `server_name`. The `port_number` only needs to be entered if the port is different from 80.

If you have not run Netscape previously, a lot of informational messages and windows will be shown while Netscape is setting up the environment in your home directory. This is standard behavior for the first execution of Netscape. The messages will not be shown again the next time you start Netscape.

The top part of the Documentation Search Service page allows you to specify your search criteria and the bottom part shows what on-line manuals have been installed. The following shows the documentation search service page with only the command reference manuals and the programming guide manuals installed (Figure 51):

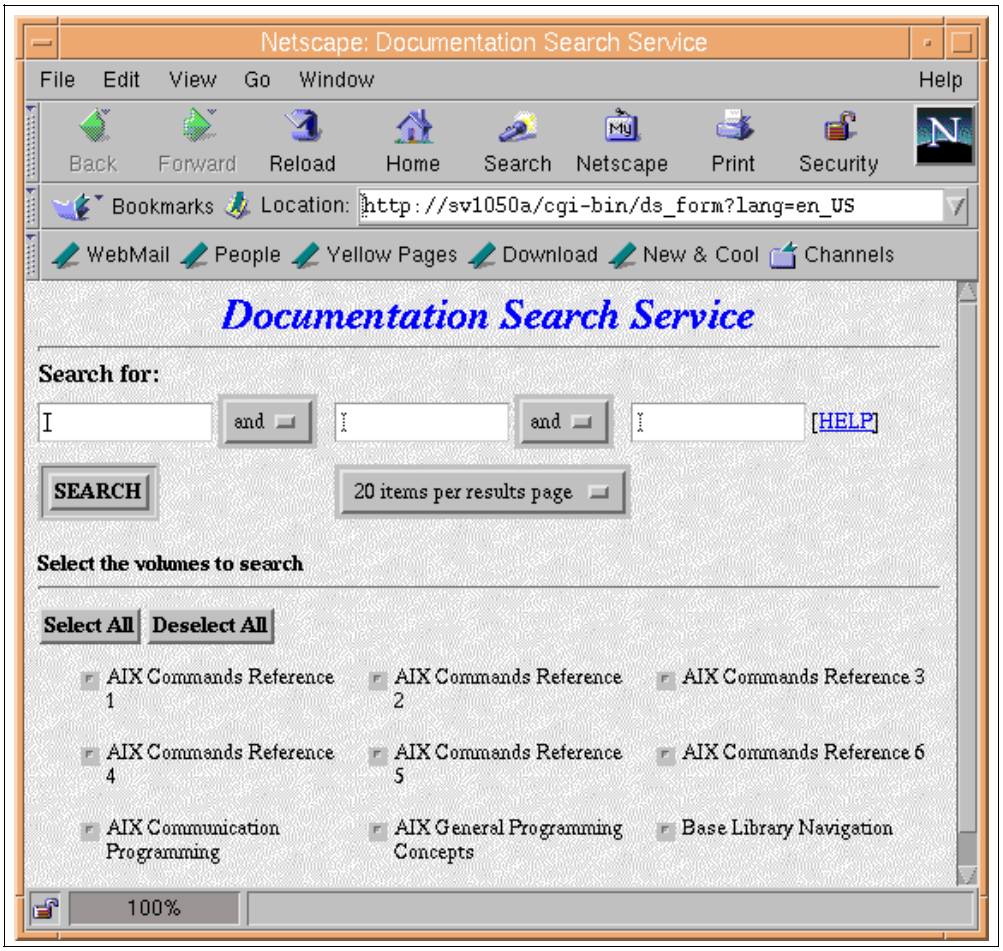


Figure 51. Documentation Search Service

If you have a problem starting the Documentation Search Service, check the following environment variables. These environment variables may be set, displayed, and changed using SMIT. Start SMIT, select **System Environments**, then select **Internet and Documentation Services**.

- On the client machine
  1. Invoke the Web browser manually and enter the URL `http://<server_name>[:<port_number>]/cgi-bin/ds_form` to ensure that the server is up and running.
  2. Ensure the DEFAULT_BROWSER variable is set to the command for starting your Web browser.

Use the command `echo $DEFAULT_BROWSER` to find out the command used in starting the browser. Test whether that command can bring up the browser by manually entering it on the command line.

3. Ensure the DOCUMENT_SERVER_MACHINE_NAME variable is set to the document server's hostname or ip address.
4. Ensure the DOCUMENT_SERVER_PORT variable is set to the port address used by the document server's port address.

- On the server machine

1. Ensure the DEFAULT_BROWSER variable is set to the command for starting your Web browser.

   Use the command `echo $DEFAULT_BROWSER` to find out the command used in starting the browser. Test whether that command can bring up the browser by manually entering it on the command line.
2. Ensure the DOCUMENT_SERVER_MACHINE_NAME variable is set to the local hostname.
3. Ensure the DOCUMENT_SERVER_PORT variable is set to the port address used by the local Web server.
4. Ensure that the CGI_DIRECTORY variable is set to the correct cgi-bin directory used by the local Web server.
5. Ensure that the DOCUMENT_DIRECTORY is set to the directory where the symbolic links `doc_link` and `ds_images` reside. If you have not changed the default, it should be in /usr/lpp/internet/server_root/pub for both IBM Internet Connection Server and Lotus Domino Go Web Server.
6. If you are not using the default directory, ensure that you have defined the necessary directory mapping in your Web server configuration file such that the directory can be resolved.

9.4 Internationalization

At the time of writing, the Documentation Search Service has language support for several languages and codesets. For a list of supported languages, codesets, and locales, see the language support table section in *AIX Version 4.3 General Programming Concepts: Writing and Debugging Programs*, SG23-2533.

Web browers also have a set of supported languages and codesets. You should be aware that differences between the client and server's language and codeset may cause some documents to become difficult to read.

For more information on the latest internationalization, see 10.7, "Documentation Search Service: DBCS HTML Search Engine (4.3.2)" on page 284.

9.5 Man Page Changes

Prior to AIX Version 4.3, the `man` command extracted its information from the InfoExplorer database. In AIX Version 4.3, the traditional InfoExplorer LPP is no longer part of the base. The replacement of InfoExplorer with HTML-based documentation has brought about a change in the way the `man` command now operates.

Previously, the man command had to search the InfoExplorer database to locate the information that it required. Now under AIX Version 4.3, the information is stored in HTML files. The HTML files are stored in the directory structure by COLLECTION_ID and BOOK_ID. The man command uses this structure to locate the required information. This new structure provides the user with performance benefits, because the search criteria is more narrowly defined.

In terms of overall function, the man command remains unchanged, performing the way it always has.

9.6 SMIT Documentation

With the introduction of HTML-based product libraries, the SMIT documentation and all of the SMIT helps have been converted to HTML. SMIT is updated to access the HTML versions of the help files. The naming convention of the filesets has also been changed to reflect the introduction of Web-Based Systems Management.

Chapter 10. National Language Support

The AIX operating system has no built-in dependencies on code set, character classification, character comparison rules, character collation order, monetary formatting, numeric punctuation, date and time formatting, or the text of messages. The national language support (NLS) environment is defined by a combination of language and geographic or cultural requirements. These conventions consist of four basic components:

- Translated language of the screens, panels, and messages
- Language convention of the geographical area and culture
- Language of the keyboard
- Language of the documentation

Customers are free to mix and match the above components for each user. The NLS environment allows AIX to be tailored to the individual user's language and cultural expectations.

To support this design, applications use standard APIs to display messages and handle characters in code set independent fashion. Additionally, libraries hide all code set-independent processing and do not alter the locale set by applications.

10.1 National Language Character Handling

The NLS feature of AIX Version 4.3 allows input and output of national language (NL) characters. NL-specific cultural conventions can be set by the user from the command line or by an application on a per-process basis. These cultural conventions include the territory-unique ways to represent date, time, monetary values, numbers, and collating sequences. By setting the appropriate environment variables, users can define their own NL behavior. Individual users may even operate using different locales, keyboards, and language text on the same system.

10.2 Levels of NLS Enablement

AIX provides the following levels of national language support enablement. To determine the level of NL support provided by any particular IBM licensed product, please consult the program product announcement material. To assist in classifying the extent to which a product provides national language support, the levels are listed following in descending order of national language capability:

Universal Language Support (ULS)
: Support for multiple languages based on universal character set ISO 10646. See Section 10.3, “Unicode” on page 242 for more information.

Full International Language Support (ILS)
: Support for all locales in the underlying operating system.

Multi-Byte Character Set Support (MBCS)
: Support for all locales based on multi-byte and single-byte code sets in the underlying operating system. Bidirectional code set support is limited.

Single-Byte Character Set Support (SBCS)
: Support is limited to single-byte locales in the underlying operating system. Bidirectional code set support is limited.

PASSTHRU
: Support is limited to the ability of the product to pass data through the program without processing. The information is handled in such a manner that all data, control, and graphics characters flow unaltered through the program directly to its output.

The following products and commands do not support full ILS:

- Adobe Acrobat Reader supports PASSTHRU only graphics, tplot, graph, and spline commands.
- Netscape Navigator supports PASSTHRU only.
- NFS supports PASSTHRU only.
- NCS supports PASSTHRU only.
- TCP/IP `telnet` command does not support NLS.

10.3 Unicode

The Universal Coded Character Set (UCS), or Unicode (UCS-2), is a character code designed to encode text for display and storage in computer-based files. The UCS global character encoding was developed jointly by the computer industry and the International Organization for Standardization (ISO) and defines the state of the art for character handling.

The design of the Unicode standard is based on the simplicity and consistency of today's prevalent character code set, ASCII (and Latin-1, an extended version of the ASCII code set), but goes beyond ASCII's limited ability to encode only the Latin alphabet. The Unicode encoding provides the capability to encode all of the characters used by all the principle written languages throughout the world.

To accommodate the many thousands of characters used in international text, the Universal Coded Character Set was developed and implemented in two variations:

UCS-2 A 16-bit code

UCS-4 A 32-bit code

However, UCS-2, the 16-bit code (Unicode), has already been established as the predominant code, while the 32-bit code is of limited practical interest. The Unicode implementation keeps character coding simple and efficient since the Unicode standard assigns each character a unique 16-bit value. It does not require complex modes or escape codes to specify modified characters or special cases.

The development of UCS on AIX Version 4.3 is based on Unicode code set Version 2.0, which is a widely accepted standard for encoding international character data. The AIX implementation of Unicode (UCS-2) will allow the user to process data from any of the supported Unicode scripts and to mix and match characters from differing language scripts within the same session.

Note: The terms UCS-2 and Unicode will be used interchangeably throughout this documentation.

The Unicode 2.0 standard (ISBN 0-201-48345-9) is the basis for the AIX implementation and is used as the primary reference.

Further information is also available at the Unicode URL: *http://www.unicode.org*

10.3.1 UTF-8

For file systems to be able to work with Unicode, a UCS Transformation Format was developed by the X/Open Internationalization (also known as I18N, because there are 18 letters between the I and the N) working group. This transformation, called UTF-8, provides a way of transforming Unicode into an ASCII representation.

The UTF-8 transformation is important when considering that the majority of file systems are ACSII-based. When using the UTF-8 Transformation Format, you are assured of a file system-safe way of using Unicode to store files.

The UTF-8 transformation format is a multi-byte code set capable of encoding the same set of characters of UCS-2 in 1 to 3 bytes or UCS-4 in 1 to 6 bytes. It has the following characteristics:

- ASCII (7-bit code) is a proper subset.
- It preserves the semantics of the portable character set.
- It preserves the semantics of a null octet for the C programming language.

Note: It is expected that UTF-8 will be a dominant transformation method where the UCS is not practical.

10.3.2 ULS

ULS, or Universal Language Support, refers to a toolkit of functions that enable applications to work with UCS. The ULS is designed to be operating system-independent, so the functions can be ported to any system.

The Universal Language Support specification describes a set of functions and utilities for addressing a wide range of national language support (NLS) problems. In summary, it provides:

- A universal coded character set capable of encoding most of today's languages and scripts.
- A set of universal language support functions for the processing of input and output of supported characters.
- A set of universal locale objects that describe processing of text on a global scope.
- A set of universal layout objects that describe processing of text on a global scope.
- A set of universal conversion objects for import/export of traditional data into the ULS environment.

This specification identifies the level of support expected for the Universal Coded Character Sets in the AIX operating system and all future releases of it. Without

abandoning its standards-based internationalization support, ULS will add support for a set of UCS locales as an extension to the existing internationalization language support.

10.3.3 Universal Locale

The Universal Locale is a method of using Unicode as a wchar_t or wide character encoding. As mentioned earlier, Unicode is based on a 16-bit encoding, whereas an ASCII character is based on an a 7-bit encoding.

For AIX to use Unicode across the entire system, it uses the UTF-8 encoding for files and uses Unicode as the process code. Process code is the actual code that is being executed in memory.

Note: All supported locales will function correctly in both their current code as well as UTF-8.

10.3.3.1 Locale Definitions

The following can be said about locale definitions:

- Unicode-based locales must be compiled based on a UTF-8 charmap containing all characters currently defined in Unicode 2.0. The names for such characters must be the exact character names as outlined in the standard, with any spaces imbedded in the character name changed to underscores. For example, the correct symbol name for the Unicode value U+00A1 would be <INVERTED_EXCLAMATION_MARK>.
- Character symbol names that are currently required to be defined by the `localedef` command must be defined with both the Unicode and POSIX symbol names in the charmap. For example, the value U+0041 would be defined as both <A> and <LATIN_CAPITAL_LETTER_A>.
- To ensure that multiple locales can be built from a given locale, all existing charmaps must be modified to use the Unicode names. Each charmap will define only those actually contained within the code set.

Many of the locale definitions have similar characteristics across multiple locales, especially in the LC_CTYPE section. From a maintenance perspective, it is highly desirable to be able to maintain a common source for these and allow them to have an include facility. The C preprocessor can be used for this purpose, provided that the localedef comment character and escape character (line continuation) be changed to values that do not conflict with the C preprocessor.

Currently, the default comment character for locales is "`#`", and the default continuation character is "`\`". By changing these to "`*`" and "`/`", respectively, you can use C preprocessor directives in the context of a localedef source file, such as `#include` and `#ifdef` statements to tailor the locale definition based on the characters available in the codeset. Thus, before running `localedef`, each locale source is run through the C preprocessor with the code set name to be used specified as an option to the C preprocessor through `-D` on the command line. For example, suppose your locale definition was to be compiled with four different code sets, ISO8859-1 (Latin), ISO8859-5 (Cyrillic), ISO8859-7 (Greek), and UTF-8 (Unicode). Moreover, suppose that in the LC_CTYPE section you wanted to declare the following three characters as uppercase:

```
<LATIN_CAPITAL_LETTER_A>
```

```
<GREEK_CAPITAL_LETTER_ALPHA>

<CYRILLIC_CAPITAL_LETTER_A>
```

This presents a bit of a problem since <LATIN_CAPITAL_LETTER_A> is defined in all four codesets, <GREEK_CAPITAL_LETTER_ALPHA> is only defined in ISO8859-7 and UTF-8, and <CYRILLIC_CAPITAL_LETTER_A> is defined only in ISO8859-5 and UTF-8. Such a situation could be coded as follows with the preprocessor directives:

```
upper /
<LATIN_CAPITAL_LETTER_A>/
#if defined(ISO88597) || defined(UTF8)
;<GREEK_CAPITAL_LETTER_ALPHA>/
#endif
#if defined(ISO88595) || defined(UTF8)
;<CYRILLIC_CAPITAL_LETTER_A>
#endif
```

Using such techniques, it is possible to create a locale definition that tailors itself to the code set being compiled against while not requiring you to define dummy codepoints in the charmap. This technique also fulfills the requirement that all locales for a given country be generated from the same locale source definition, so that consistency is achieved within a given language/territory combination.

To support Unicode symbol names in charmaps and localedef source files, the `localedef` command has been modified to accept symbol names of at least 85 characters instead of the current limit of 32. Currently, the longest symbol name from the Unicode standard is 83 characters long, not including the required < and > delimiters. This is a simple modification to symtab.h and has been done without any impact on performance, as the symbol names are only used during locale compilation, not at run time.

10.3.3.2 Locale Methods

The current set of locale methods, as provided in libc, was implemented based on the locales supported in AIX Version 3.2. Since that time, many new locales have been added to the system, mostly with corresponding locale-specific methods. It is a goal of this design to minimize the number of locale-specific method objects required and also to provide a framework under which future AIX locales can be easily added.

Locale-specific methods that are no longer used will remain in the libc.a library in order to insure binary compatibility. The following is the set of locale-specific methods that are provided with AIX Version 4.3. These locale methods have been placed into libi18n.a, instead of libc.a, to minimize incompatibilities with previous releases.

mblen()

Determine the length (in bytes) of a multi-byte character. This is the most difficult method to generalize. The following locale-specific variants are supported:

- __mblen_sb() - For single-byte code sets, always returns 1.
- __mblen_utf() - multi-byte determination rules for UTF-8-based locales.

- __mblen_std() - For all other code sets, use the __mbtowc_std() method to determine the number of bytes in the character by calling the appropriate iconv() converter and seeing how many bytes were able to be converted.

mbstopcs() Convert multi-byte string to process code string. This function is somewhat obsolete, as mbstowcs() is the preferred alternative. There is one standard locale method, __mbstopcs_std(), which loops through the multi-byte string calling mbtopc() to convert each multi-byte character to the appropriate process code.

mbstowcs() Convert multi-byte string to wide character string. There is one standard locale method, __mbstowcs_std(), which loops through the multi-byte string calling mbtowc() to convert each multi-byte character to the appropriate wide character.

mbtopc() Convert multi-byte character to process code. This function is somewhat obsolete, as its function is duplicated by the mbtowc() function. There is one standard locale method, __mbtopc_std(), which calls mbtowc() to perform the intended function of converting the multi-byte character to process code (wide character).

mbtowc() Convert multi-byte character to wide character. All of the provided mbtowc() methods convert the multi-byte character to a Unicode value. There are three different methods provided in order to optimize performance of this operation based on the nature of the multi-byte codeset:

- __mbtowc_iso1() - Since ISO8859-1 is a proper subset of Unicode. Its data can be converted to Unicode by simply casting its 8-bit value to a 16-bit value. This is the fastest method for converting ISO8859-1 to Unicode.
- __mbtowc_utf() - UTF-8-based data can be converted to Unicode by using a simple bit-shifting algorithm. This method should be used for all UTF-8 based locales since it is faster than the __mbtowc_std() method listed below.
- __mbtowc_std() - For all other codesets, the __mbtowc_std() method converts the multi-byte data to Unicode using the iconv() interface. The conversion descriptor is defined as a static pointer so that multiple calls to __mbtowc_std() will not incur the overhead of multiple iconv_open() calls. Because this locale method is dependent on libiconv, it has been placed into a common locale method object in /usr/lib/nls/loc/methods/stdmeth.o instead of in the libc library, to avoid creating an unnecessary dependency between libc and libiconv.

pcstombs() This method is obsolete since its function has been replaced by the method wcstombs(). Currently, this function just returns `-1`.

pctomb() This method is obsolete since its function has been replaced by the method wctomb(). Currently, this function just returns `-1`.

wcstombs() Convert wide character string to multi-byte string. There is one standard locale method __wcstombs_std(), which loops through the wide character string calling wctomb() to convert each wide character to the appropriate multi-byte string.

wctomb() Convert wide character to multi-byte character. All of the provided wctomb() methods convert a Unicode value to the appropriate multi-byte string. There are three different methods provided to optimize performance of this operation based on the nature of the multi-byte codeset:

- __wctomb_iso1() - Since ISO8859-1 is a proper subset of Unicode, its data can be converted from Unicode by simply casting its 16-bit value to an 8-bit value. This is the fastest method for converting Unicode to ISO8859-1.
- __wctomb_utf() - UTF-8-based data can be converted from Unicode by using a simple bit-shifting algorithm. This method should be used for all UTF-8-based locales since it is faster than the __wctomb_std() method listed below.
- __wctomb_std() - For all other codesets, the __wctomb_std() method converts the multi-byte data from Unicode using the iconv() interface. The conversion descriptor is defined as a static pointer so that multiple calls to __wctomb_std() will not occur. Because this locale method is dependent on libiconv, it has been placed into a common locale method object in /usr/lib/nls/loc/methods/stdmeth.o instead of the libc library to avoid creating an unnecessary dependency between libc and libiconv.

wcswidth() Determine the display width of a wide character string. There is one standard locale method, __wcswidth_std(), that loops through the wide character string calling wcwidth() to determine the display width of each character.

wcwidth() Determine the display width of a wide character. Since all wide characters can be assumed to have the Unicode encoding, each character's display width can be determined with a single locale method.

- __wcwidth_std() - Determine the display width of a character based on its Unicode value. Most characters that can be displayed have a display width of 1; CJK ideographs and Hangul syllables will have a display width of 2, and combining characters have a width of 0. The assumption that the wide character encoding is Unicode allows for a single wcwidth() method that is appropriate for all locales.

Localedef Command Impacts

The `localedef` command has a global method table that is defined internally to the command and is used if the `localedef` command user does not explicitly state the methods desired using the -m flag. These tables will need to be modified and expanded to reflect the newly-available methods.

Table 48. Internal Locale Methods Called for Each Locale

Locale method	Unicode locales	ISO8859-1-based locales	Single byte non-ISO1-based locales	Multi-byte locales
mblen()	__mblen_utf()	__mblen_sb()	__mblen_sb()	__mblen_std()
mbstopcs()	__mbstopcs_std()	__mbstopcs_std()	__mbstopcs_std()	__mbstopcs_std()
mbstowcs()	__mbstowcs_std()	__mbstowcs_std()	__mbstowcs_std()	__mbstowcs_std()

Locale method	Unicode locales	ISO8859-1-based locales	Single byte non-ISO1-based locales	Multi-byte locales
mbtopc()	__mbtopc_std()	__mbtopc_std()	__mbtopc_std()	__mbtopc_std()
mbtowc()	__mbtowc_utf()	__mbtowc_isol()	__mbtowc_std()	__mdtowc_std()
pcstombs()	__pcstombs_std()	__pcstombs_std()	__pcstombs_std()	__pcstombs_std()
pctomb()	__pctomb_std()	__pctomb_std()	__pctomb_std()	__pctomb_std()
wcstombs()	__wcstombs_std()	__wcstombs_std()	__wcstombs_std()	__wcstombs_std()
wcswidth()	__wcswidth_std()	__wcswidth_std()	__wcswidth_std()	__wcswidth_std()
wctomb()	__wctomb_utf()	__wctomb_isol()	__wctomb_std()	__wctomb_std()
wcwidth()	__wcwidth_std()	__wcwidth_std()	__wcwidth_std()	__wcwidth_std()

10.3.3.3 Input Methods

The universal input method that is currently used in the UNIVERSAL locale is the basis for input methods in Unicode-based locales. ISO/IEC DIS 14755 basic operation support has been added, which provides the ability to enter a Unicode character by holding down <Ctrl>,<Shift> and entering the appropriate hexadecimal representation. The character set lists align with the script designations in the Unicode 2.0 standard. Input method lists contain simply the xx_XX designation for the input method to be selected. These attributes are configurable through the .imcfg file for the universal locale.

The default input method selected should match that of the corresponding locale. For example, bringing up the JA_JP (Unicode) locale should default to the Japanese input method.

10.3.3.4 Fonts and X11 Locales

AIX currently provides bitmap fonts for UTF-8-based versions of the Baltic locales in the fileset X11.fnt.ucs.com. However, these fonts currently contain only those characters necessary for Baltic support. Similar types and sizes for these fonts already exist for a full complement of characters from ISO8859-1 to ISO8859-9, IBM-1046 and IBM-850. These existing bdf fonts have been combined to provide a nearly-complete set of single wide bitmap fonts.

In the same way, 19 point and 27 point CJK fonts can be generated to provide a complete set of UCS-based fonts in two sizes that cover the entire CJK Ideograph Range (4E00 - 9FFF) and the Hangul Syllables Range (AC00 - D7FF). These fonts are shipped with the fileset X11.fnt.ucs.cjk.

Dt font aliases have been created to provide the appropriate font sets, so that any Unicode-based locale should operate with a similar set of compatible fonts.

An alternate X11 locale has been made available for use in those installations where CJK font support is not desired and where loading of complete CJK fonts would be a severe hindrance to proper performance.

10.3.3.5 Layout Services

A single layout object, /usr/lib/nls/loc/methods/uni_layout.o, has been provided that will format Unicode characters according to the following sets of rules:

- For Hebrew and Arabic, characters are presented according to the bidirectional behavior rules as presented in the Unicode 2.0 standard, section 3.11.
- For Vietnamese, combining sequences that correspond to characters in the Unicode range U+1EA0 through U+1EF9 have been changed to their precombined forms for presentation purposes.
- Simple combining sequences for characters in the Basic Latin and Latin A extensions shall be honored and altered to the precomposed forms for presentation (that is, characters less than U+01FF).
- Where fonts exist to do so, combining sequences other than those mentioned above are presented as *show hidden* sequences.
- An optional layout engine that formats Korean according to the rules in Section 3.10 of the Unicode Standard.

10.3.4 Installation and Packaging

The following is true regarding installation and packaging:

- Unicode-based locales are selectable for install from SMIT and from the BOS install menus.
- Any locale support that is common to all UTF-8 Unicode based platforms is packaged and shipped in the fileset bos.loc.com.utf.
- Each Unicode based locale is packaged as bos.loc.XX_XX, where XX_XX is the language designation (that is, EN_US, FR_FR, and so on).
- If locale-specific X11 resources are necessary, they are packaged as X11.loc.XX_XX, as appropriate.
- Translated message filesets have been created for all Unicode-based locales that directly correspond with a translatable AIX language. For example, bos.msg.KO_KR was created for Korean messages in the Unicode-based locale.
- See 10.3.5, “List of Supported Unicode Locales” on page 249 for a list of supported Unicode locales.

10.3.5 List of Supported Unicode Locales

Table 49 provides the currently supported Unicode locales:

Table 49. Supported Unicode Locales

Unicode Locale	Language Designation
Albanian	SQ_AL
Arabic	AR_AA
Bulgarian	BG_BG
Catalan	CA_ES
Chinese (Simplified)	ZH_CN

Unicode Locale	Language Designation
Chinese (Traditional)	ZH_TW
Croatian	HR_HR
Czech	CS_CZ
Danish	DA_DK
Dutch	NL_NL
Dutch (Belgium)	NL_BE
English (Great Britain)	EN_GB
English (United States)	EN_US
Estonian	ET_EE
Finnish	FI_FI
French	FR_FR
French (Belgium)	FR_BE
French (Canada)	FR_CA
French (Switzerland)	FR_CH
German	DE_DE
German (Switzerland)	DE_CH
Greek	EL_GR
Hebrew	IW_IL
Hungarian	HU_HU
Icelandic	IS_IS
Italian	IT_IT
Japanese	JA_JP
Korean	KO_KR
Latvian	LV_LV
Lithuanian	LT_LT
Macedonia	MK_MK
Norwegian	NO_NO
Polish	PL_PL
Portuguese	PT_PT
Portuguese (Brazil)	PT_BR
Romanian	RO_RO
Russian	RU_RU
Serbian Cyrillic	SR_SP
Serbian Latin	SH_SP

Unicode Locale	Language Designation
Slovak	SK_SK
Slovene	SL_SI
Spanish	ES_ES
Swedish	SV_SE
Thai	TH_TH
Turkish	TR_TR
Vietnamese	VI_VN

10.4 Java NLS Support

Prior to AIX Version 4.3, Java Version 1.02 was shipped with AIX. In AIX Version 4.3, Java Version 1.11 is shipped as part of the base operating system.

The previous version of Java (1.02) had virtually no national language support (Java 1.02 only really has support for the English language). However, Java 1.1 is NLS enabled for the G7 countries: France, the United States, U.K., Germany, Japan, Italy and Canada. Currently, there is no support within Java 1.1 for bidirectional languages. All of the locale support within Java is based on Unicode.

Below is a summary of the Java 1.1 NLS support:

- Locale support
  - Character types based on Unicode 2.0
  - Date, time, number, and currency formatting
  - Collation
- External character sets
  - Font handling
  - Use OS-provided input methods
  - Message handling
  - Printing support
  - Text line-break

10.5 Euro Symbol Support for AIX (4.3.2)

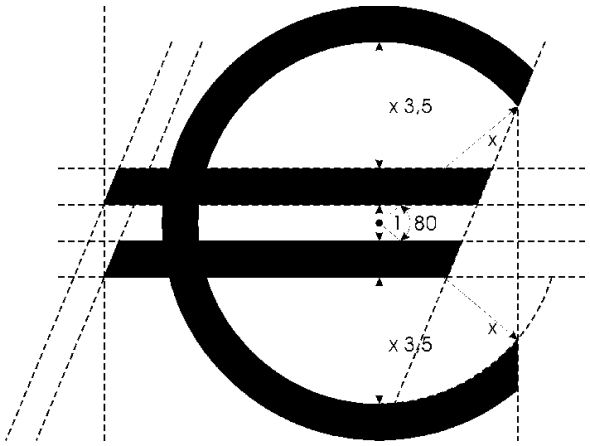


Figure 52. Euro Symbol (http://europa.eu.int/euro/html/entry.html)

This section will provide you with the necessary information required to introduce the Euro symbol as a valid graphical and printable character to your AIX system.

The primary means of code set support for the Euro symbol is achieved by use of the UTF-8 multi-byte encoded locales for each country. A detailed outline of the locale definitions for the UTF-8 code set, the keyboard definitions, the input methods, and the codeset conversion tables is given.

For those customers that have applications that will not support multi-byte encoding, such as UTF-8, a Euro single-byte migration option based on the IBM-1252 code set is provided. This topic is addressed in 10.5.6, “Euro SBCS Migration Option - IBM-1252 Locale” on page 270.

10.5.7, “Packaging” on page 271, and 10.5.8, “Installation of Euro Symbol Support” on page 272 cover the Euro symbol support in more practical terms. You may want to skip over to these sections first and install the Euro symbol related locales by following the documented step-by-step instructions, gain some experience in the new environment, then come back to the more theoretical oriented sections at a later time.

10.5.1 Overview

The Euro is being introduced by the European Monetary Union (EMU) as a common currency to be adopted by all EMU member countries. Initial use of the Euro in banking and industry is planned beginning in January of 1999. During the first three years after introduction, the Euro currency and the existing national currencies will both be used, and a fixed exchange rate will be established. The goal is to completely replace the existing national currencies by the year 2002.

The primary means of achieving code set support for the Euro symbol is to make use of the UTF-8 locales for each country. UTF-8 refers to the X/Open file system safe UCS transformation format (FSS-UTF). It is a multi-byte code set suitable to encode plain text on traditional byte-oriented systems, such as AIX, and is directly related to the universal code character set (UCS). The Unicode standard has adopted code point position U+20AC as <EURO_SIGN>. This is not to be confused with the old European currency symbol located at U+20A0.

The new locales must be able to effectively support the dual currency situation that will exist between the years 1999 - 2002. During this time period, the locale definition will still use the country's national currency definition as the default, but a mechanism is provided to switch the LC_MONETARY definition so that the Euro currency formatting is used instead of the country's national currency formatting rules.

For those customers that have applications that will not support multi-byte encodings such as UTF-8, a Euro single-byte migration option is provided as well. This option is based on the Windows 1252 placement of the Euro at 0x80 to provide the best compatibility with Windows 98/NT clients.

10.5.2 Local Definitions for the UTF-8 Code Set

A locale is made up of the language, territory, and code set combination used to identify a set of language conventions. The language specific information is accessed through the locale database that is compiled by the `localedef` command. The `localedef` command takes three different files as input:

- One file describes the local methods to be overridden in respect to the defaults when constructing a locale.
- The second file contains a mapping from the character symbols and collating element symbols to actual character encodings.
- Finally, the language conventions are grouped in six categories to include information about collation, case conversion, and character classification, the language of message catalogs, date-and-time representation, the monetary symbol, and numeric representation.

The local category source definitions are given in the third input file, the local definition source file.

The following categories can be defined for a given local:

LC_COLLATE	Determines character-collation or string-collation information.
LC_CTYPE	Determines character classification, case conversion, and other character attributes.
LC_MESSAGES	Determines the format for affirmative and negative responses.
LC_MONETARY	Determines rules and symbols for formatting monetary numeric information.
LC_NUMERIC	Determines rules and symbols for formatting and non-monetary numeric information.
LC_TIME	Determines a list of rules and symbols for formatting time and date information.

NLS uses the environment variables LC_COLLATE**,** LC_CTYPE**,** LC_MESSAGES**,** LC_MONETARY**,** LC_NUMERIC and LC_TIME to define the current values for their respective categories and to influence the selection of locales. In addition to the previously mentioned medium priority class environment variables, the LANG low priority class environment variable specifies the installation default locale. You can change the LANG variable at any time. Furthermore, the high priority class variable LC_ALL is provided that takes precedence above all the other NLS environment variables mentioned before.

Any UTF-8 based locale installed on an AIX system will provide the desired support for the Euro symbol. For those countries that may have to actively use the Euro (Table 51 on page 256), the locales will deliver the input methods and the keyboard maps required to enter the Euro symbol through the keyboard. For the same group of countries, an additional LC_MONETARY locale is available to enable the Euro currency formatting. This locale is identified by the suffix *@euro.* For example, if you are using the UTF-8 locale DE_DE, which specifies German language and territory, and your system is configured for Euro currency formatting, then the `locale` command will return the following output:

```
# locale
LANG=DE_DE
LC_COLLATE="DE_DE"
LC_CTYPE="DE_DE"
LC_MONETARY="DE_DE@euro"
LC_NUMERIC="DE_DE"
LC_TIME="DE_DE"
LC_MESSAGES="DE_DE"
LC_ALL=
```

The related locale definition source files and the locale databases can be found in the /usr/lib/nls/loc directory:

```
# cd /usr/lib/nls/loc
# ls -l DE_DE* | egrep  -v "\.[il]" | cut -c55-
DE_DE -> /usr/lib/nls/loc/DE_DE.UTF-8
DE_DE.UTF-8
DE_DE.UTF-8.src
DE_DE.UTF-8@euro
DE_DE.UTF-8@euro.src
DE_DE.UTF-8@euro__64
DE_DE.UTF-8__64
DE_DE@euro -> /usr/lib/nls/loc/DE_DE.UTF-8@euro
DE_DE@euro__64 -> /usr/lib/nls/loc/DE_DE.UTF-8@euro__64
DE_DE__64 -> /usr/lib/nls/loc/DE_DE.UTF-8__64
```

The locale database for the DE_DE locale is actually an alias for DE_DE.UTF-8, and likewise, the database for the DE_DE@euro locale is linked to DE_DE.UTF-8@euro. Note, you should not use the referenced locale databases DE_DE.UTF-8 and DE_DE.UTF-8@euro explicitly as parameter to the `chlang` command or as variable value for LANG. The names of the locales are aligned to the traditional AIX locale naming convention.

Since a locale is a loadable object module, a different object is required when running in the 64-bit environment. Consequently, you find one 64-bit enabled locale database for any 32-bit locale database. The 64-bit databases are readily identified by the *__64* suffix. In the 64-bit environment, the application will automatically append *__64* to the name of the locale when searching for the proper NLS.

The locale definition source files hold the suffix *.src* and belong to the separately installable bos.loc.adt.locale fileset.

10.5.2.1 Euro Sign Character Classification

For each UTF-8 based locale, the Euro sign was added as a valid graphical and printable character. The Euro sign is not classified as an alphabetic, upper or lower case letter, nor a numeric symbol.

For example, if you display the locale definition source file /usr/lib/nls/loc/DE_DE.UTF-8.src for the German UTF-8 locale, you will find one entry for each of the keywords `graph` and `print` in the LC_CTYPE category statement:

```
*************
LC_CTYPE
*************
...
graph /
...
          ;<EURO-CURRENCY_SIGN>/
...
          ;<EURO_SIGN>/
...
print /
...
          ;<EURO-CURRENCY_SIGN>/
...
          ;<EURO_SIGN>/
...
END LC_CTYPE
```

(<EURO-CURRENCY_SIGN> refers to the old European currency symbol.)

The locale definition source files belong to the separately installable bos.loc.adt.locale fileset. Note that the Euro sign is added for all UTF-8 based locales not only for the locales of the countries which are member of the European Monetary Union. Indeed, this is one of the advantages of the Universal Coded Character Set (UCS).

10.5.2.2 Euro Sign Encoding

For each UTF-8 based locale, the UTF-8 character set description source file /usr/lib/nls/charmap/UTF-8 maps the old European currency symbol to U+20A0 and the new Euro sign to U+20AC. Each character symbol definition consists of a symbol name and the character encoding. In our case, the code value is given as a set of three hexadecimal constants:

```
...
# Currency Symbols : U20A0 - U20CF
#

<EURO-CURRENCY_SIGN>                                \xe2\x82\xa0
...
<EURO_SIGN>                                         \xe2\x82\xac
...
```

Table 50 provides an overview of the different encoding for the <EURO-CURRENCY_SIGN> and the <EURO_SIGN>:

Table 50. Encoding for the European Currency Symbol and Euro Sign

Encoding	<EURO-CURRENCY_SIGN>	<EURO_SIGN>
UCS-2: Hexadecimal Representation	20A0	20AC

Encoding	<EURO-CURRENCY_SIGN>	<EURO_SIGN>
UTF-8 multi-byte: Byte Sequence in Hexadecimal Representation	e2 82 a0	e2 82 ac
UTF-8 multi-byte: Byte Sequence in Binary	11100010 10000010 10100000	11100010 10000010 10101100

For more details about character encoding refer to Code Set Overview in *AIX Version 4.3 General Programming Concepts: Writing and Debugging Programs*.

The character set description source files belong to the separately installable fileset bos.loc.adt.locale.

10.5.2.3 LC_MONETARY Formatting Information

For each Western European locale that may actively use the Euro symbol, an additional locale that contains the LC_MONETARY information for the Euro currency is provided. A list of all affected locales is given in Table 51:

Table 51. List of Locales for Euro-Specific LC_MONETARY Locale

Language/Territory	Identifier
Catalan	CA_ES
Dutch (Belgium)	NL_BE
Dutch	NL_NL
English (Great Britain)	EN_GB
Finnish	FI_FI
French (Belgium)	FR_BE
French (Switzerland)	FR_CH
French	FR_FR
German (Switzerland)	DE_CH
German	DE_DE
Italian	IT_IT
Portuguese	PT_PT
Spanish	ES_ES

To obtain the traditional local currency definition, the standard locale is used as before. Whenever the Euro currency formatting is desired, the LC_MONETARY category should be set by the application (using the `setlocale()` subroutine) or by the user (using the LC_MONETARY environment variable) to "*XX_XX@euro*", where *XX_XX* represents the language and territory designation for any of the locales listed in the previous table.

The LC_MONETARY information for each @euro variant is defined by the related locale definition source file. For example, you will find in the DE_DE@euro locale

definition source file /usr/lib/nls/loc/DE_DE.UTF-8@euro.src the following entries for the LC_MONETARY category:

```
*************
LC_MONETARY
*************

int_curr_symbol   "<E><U><R><SPACE>"
currency_symbol   "<EURO_SIGN>"
mon_decimal_point <COMMA>
mon_thousands_sep "<FULL_STOP>"
mon_grouping      3
positive_sign     ""
negative_sign     "<HYPHEN-MINUS>"
int_frac_digits   2
frac_digits       2
p_cs_precedes     0
p_sep_by_space    1
n_cs_precedes     0
n_sep_by_space    1
p_sign_posn       1
n_sign_posn       1

END LC_MONETARY
```

The XPG monetary formatting subroutine *strfmon()* uses the information provided to format monetary quantities according to the settings for the keywords, as provided in Table 52.

Table 52. LC_MONETARY Keywords for the Euro Locale

Keyword	Description
int_curr_symbol	Specifies the string used for the international currency symbol.
currency_symbol	Specifies the string used for the local currency symbol.
mon_decimal_point	Specifies the string used for the decimal delimiter.
mon_thousands_sep	Specifies the character separator used for grouping digits to the left of the decimal delimiter.
mon_grouping	Specifies a string that defines the size of each group of digits.
positive_sign negative_sign	Specifies the string used to indicate a nonnegative / negative-valued formatted monetary quantity.
int_frac_digits frac_digits	Specifies an integer value representing the number of fractional digits (those after the decimal delimiter) to be displayed in a formatted monetary quantity using the int_curr_symbol / currency_symbol value.
p_cs_precedes n_cs_precedes	Specifies an integer value indicating whether the int_curr_symbol or currency_symbol string, precedes (1) or follows (0) the value for a nonnegative / negative formatted monetary quantity
p_sep_by_space n_sep_by_space	Specifies an integer value indicating whether the int_curr_symbol or currency_symbol string is separated (1) or not separated (0) by a space from a nonnegative / negative formatted monetary quantity

Keyword	Description
p_sign_posn n_sign_posn	Specifies an integer value indicating the positioning of the positive_sign/negative_sign string for a nonnegative/negative formatted monetary quantity. 1 Indicates that the positive_sign/negative string precedes the quantity and the int_curr_symbol or currency_symbol string. 2 Indicates that the positive_sign/negative string follows the quantity and the int_curr_symbol or currency_symbol string. 4 Indicates that the positive_sign string immediately follows the int_curr_symbol or currency_symbol string.

For a detailed description of the LC_MONETARY category keywords refer to LC_MONETARY Category for the Locale Definition Source File Format in *AIX Version 4 Files Reference*. The locale definition source files are part of the separately installable bos.loc.adt.locale fileset.

Generally, the LC_MONETARY information for each @euro variant of the locale contains the same formatting information as the country's historical currency formatting with the following modifications, as necessary:

- The alphabetic international currency symbol as defined by the int_curr_symbol keyword is set to "<E><U><R><space>".
- The string to use for the locale currency symbol as defined by *currency_symbol* keyword is set to <EURO_SIGN> (the Euro symbol).
- The number of decimal places for Euro symbol formatting, as defined by the keywords *int_frac_digits* and *frac_digits* is set to two decimal places in all cases.
- Other formatting conventions, such as decimal point and thousands separator, can be uniquely specified by each country. The relevant keywords where minor deviations occur are *mon_decimal_point, mon_thousands_sep, p_cs_precedes, n_cs_precedes, p_sign_posn, n_sign_posn.* Table 53 summarizes the differences between the Euro LC_MONETARY locales

Table 53. Locale-Specific Deviations in the LC_MONETARY Category

Locale	mon_decimal_point	mon_thousands_sep	p_cs_precedes n_cs_precedes	p_sign_posn	n_sign_posn
CA_ES@euro	<COMMA>	<FULL_STOP>	1	1	1
DE_DE@euro	<COMMA>	<FULL_STOP>	0	1	1
ES_ES@euro	<COMMA>	<FULL_STOP>	1	1	1
FI_FI@euro	<COMMA>	<SPACE>	0	1	1
FR_BE@euro	<COMMA>	<FULL_STOP>	0	1	1
FR_FR@euro	<COMMA>	<SPACE>	0	1	1
IT_IT@euro	<COMMA>	<FULL_STOP>	1	4	4
NL_BE@euro	<COMMA>	<FULL_STOP>	0	1	1
NL_NL@euro	<COMMA>	<FULL_STOP>	1	1	2
PT_PT@euro	<DOLLAR_SIGN>	<FULL_STOP>	0	1	1

Note: the <FULL_STOP> character is similar in appearance to a period (.).

10.5.2.4 Collating Sequence for Euro Locales

For each UTF-8 based locale, the collation sequence for that locale was modified to contain the Euro symbol. The UTF-8 locales adhere to a multiple pass collation sequence, and the Euro sign will be collated in the fourth pass between the dollar sign and the sterling sign (British Pound). The appropriate entry for this behavior, in the associated locale definition source file, appear as follows:

```
...
LC_COLLATE

...
<DOLLAR_SIGN>       IGNORE;IGNORE;IGNORE;<DOLLAR_SIGN>
<EURO_SIGN>         IGNORE;IGNORE;IGNORE;<EURO_SIGN>
<POUND_SIGN>        IGNORE;IGNORE;IGNORE;<POUND_SIGN>
...

END LC_COLLATE
...
```

If the LC_MONETARY locale is set to activate the Euro symbol formatting, the Euro sign collates in the first pass between the dollar sign and the percent sign for all UTF-8 based locales listed in Table 53 on page 258. Any of the relevant locale definition source files /usr/lib/nls/loc/XX_XX.UTF-8@euro.src will have the following entries:

```
...
**************
LC_COLLATE
**************

oder_start

...
<DOLLAR_SIGN>
<EURO_SIGN>
<PERCENT_SIGN>
...

order_end

END LC_COLLATE
...
```

10.5.3 Keyboard Definitions

IBM follows the recommendation of the European Commission (EC) regarding placement of the Euro symbol on keyboards. This recommendation, along with other information from the EC, can be found at IT impact of the Euro*, (EC Information Society Project Office (ISPO))* `http://www.ispo.cec.be/y2keuro/euroit.htm`. The European Commission's recommendation is to place the Euro symbol at the position AltGr+e on all European keyboards, except on those keyboard layouts where the key combination AltGr+e is already assigned to produce a different character. In those cases, a combination of AltGr+4 or AltGr+5 will be assigned, depending on the particular keyboard layout. The AltGr (Alt Graphics) key allows you to enter additional characters through the keyboard and is located to the right of the space bar on keyboards with this feature.

Table 54 summarizes the AIX keyboards that will be modified to incorporate the Euro symbol and specifies the placement of the Euro symbol on each European keyboard. As of the writing of this document, these keyboard placements match the current recommendation of the EC regarding placement of the Euro symbol on keyboards.

Table 54. Keyboard Definitions to Incorporate the Euro Symbol

Language/Territory	AIX Keyboard Name	Keyboard ID	Euro Placement
Catalan/Spain	CA_ES	172	AltGr+e
Danish/Denmark	DA_DK	159	AltGr+5
Dutch/Belgium	NL_BE	120	AltGr+e
Dutch/Netherlands	NL_NL	143	AltGr+e
English/UK	EN_GB	166	AltGr+4
English/UK	EN_GB@alt	168	AltGr+e
Finnish/Finland	FI_FI	153	AltGr+5
Finnish/Finland	FI_FI@alt	285	AltGr+5
French/Belgium	FR_BE	120	AltGr+e
French/France	FR_FR	189	AltGr+e
French/France	FR_FR@alt	251	AltGr+e
French/Switzerland	FR_CH	150F	AltGr+e
German/Germany	DE_DE	129	AltGr+e
German/Switzerland	DE_CH	150G	AltGr+e
Icelandic/Iceland	IS_IS	197	AltGr+5
Italian/Italy	IT_IT	142	AltGr+5
Italian/Italy	IT_IT@alt	293	AltGr+5
Norwegian/Norway	NO_NO	155	AltGr+5
Portuguese/Portugal	PT_PT	163	AltGr+5
Spanish/Spain	ES_ES	172	AltGr+e
Swedish/Sweden	SV_SE	153	AltGr+5
Swedish/Sweden	SV_SE@alt	285	AltGr+5

AIX supports two different types of keyboards: low function terminal (LFT) and X server keyboards. Although these two keyboard maps appear to be the same, they are separate and distinct.

Low-function terminals (LFTs) support single-byte code-set languages using key maps. An LFT key map translates a key stroke into a character string in the code set of the given locale. LFT does not support languages that require multi-byte code sets. Hence, you will not have any Euro support on behalf of the lft0 pseudo device driver. However, if you configure your system to use UTF-8 locales to gain Euro symbol support, you will find by the use of the `lskbd` command, that the name of the LFT software keyboard map suggests a multi-byte keyboard map.

For example, in a German UTF-8 locale environment, you would get the following response by the `lskbd` command:

```
# lskbd
The current software keyboard map = /usr/lib/nls/loc/DE_DE.lftkeyboard
```

But a closer examination of the relevant keyboard map files reveals that they are symbolic links to the regular C locale LFT keyboard map: `/usr/lib/nls/loc/C.lftkeymap`.

For a full graphical Euro symbol enablement, you have to be in an X environment. The associated X server has an attached keyboard, and the server uses mapping tables to manage the mapping of keyboard events. The mapping of an X server keyboard can be changed by using the `xmodmap` command. This command converts the keyboard so that it returns the key symbol supported by the system.

At startup of the X server, a query to the ODM returns the locale that determined the keyboard map for the LFT pseudo device driver; that is, the `swkb_path` attribute of lft0 is examined for the currently used locale for the keyboard map. In the next step, the `xmodmap` command defines the proper keyboard mapping to the server according to the found locale.

When characters are typed in on the keyboard reach the server, the characters are in the form of key codes. These key codes are converted into keysyms (key symbols), or if applicable, into a pair of keysym/modifier as defined by the table provided in the client.(A modifier indicates the state of the keyboard as determined by the modifier keys: Shift, Lock, Ctrl, Alt and Alt Graphic. The keyboard table was defined to the sever by the `xmodmap` command and contains mappings for each of the keycodes into a predefined set of codes called keysyms or keysym/modifier.

The file containing the xmodmap command expressions to be run by the `xmodmap` command is in the directory /usr/lpp/X11/defaults/xmodmap/XX_XX, where *XX_XX* represents the language/territory designation for the UTF-8 locale. The following example shows the Euro specific entry for the German UTF-8 locale in /usr/lpp/X11/defaults/xmodmap/DE_DE/keyboard:

```
...
! Row 2         Base            Shift           Alt-Gr (Mod2)
! -----         ----            -----           -------------
!
!keycode 24 =   Tab             NoSymbol        NoSymbol
 keycode 25 =   q               Q               at
 keycode 26 =   w               W               NoSymbol
 keycode 27 =   e               E               EuroSign
 keycode 28 =   r               R               NoSymbol
...
```

If you press the AltGr+e key combination on a German keyboard, the first step of the input processing is completed when the keycode to keysym/modifier conversion is done. In this specific case, the modifier is masked. For example, the AltGr modifier and the character "e" are combined to map to the EuroSign key symbol. But the keysym still has to be mapped to the related character string in the code set specified by your locale before an application can process the input. This conversion is governed by the input method.

10.5.4 Input Methods for the Euro Symbol

For an application to run in the international environment, for which National Language Support provides a base, input methods are needed. An input method is a set of functions that translates key strokes, or more precisely, key symbol/modifier pairs into character strings in the code set specified by your locale. Each type of input method has the following features:

Keymaps Set of input method keymaps (imkeymaps) that work with the input method and determine the supported locales.

Keysyms Set of key symbols (keysyms) that the input method can handle.

Modifiers Set of modifiers, or states, each having a mask value, that the input method supports.

Your locale determines which input method should be loaded, how the input method runs, and which devices are used. The /usr/lib/nls/loc directory contains the input methods installed on your system. Input method file names have the format *XX_XX.im* where *XX_XX* represents the language and territory designation for any of the locales. For a given input method, you may find an associated configuration file identified by the suffix *.imcfg* .The input method provides support for user-defined imkeymaps, allowing you to customize input method mapping. The input methods support imkeymaps for each locale. The file name for imkeymaps is similar to that of input methods, except that the suffix for imkeymap files is *.imkeymap* instead of *.im*. The imkeymaps are generated by the keycomp command. The keycomp command compiles a keyboard mapping file into an input method keymap file. The locale specific keyboard mapping files are recognized by the suffix *.imkeymap.src* .

For a system set up for German UTF-8 Euro symbol support (DE_DE), you would find all the files mentioned above in the /usr/lib/nls/loc directory:

```
# ch /usr/lib/nls/loc
# ls -l DE_DE*im* | cut -c55-
DE_DE.UTF-8.im -> /usr/lib/nls/loc/sbcs.im
DE_DE.UTF-8.imcfg -> /usr/lib/nls/loc/UNIVERSAL.imcfg
DE_DE.UTF-8.imkeymap
DE_DE.UTF-8.imkeymap.src
DE_DE.im -> /usr/lib/nls/loc/UNIVERSAL.im
DE_DE.imcfg -> /usr/lib/nls/loc/UNIVERSAL.imcfg
DE_DE.imkeymap -> /usr/lib/nls/loc/DE_DE.UTF-8.imkeymap
```

Note that actually two input methods are present, and indeed, both are needed for your UTF-8 Euro environment. DE_DE.im is linked to the standard UNIVERSAL input method and DE_DE.UTF-8.im is an alias for the traditional single-byte input method commonly used in conjunction with the ISO8859 and the IBM-850 PC code sets. Both input methods are related by the use of the same UNIVERSAL input method configuration file UNIVERSAL.imcfg.

10.5.4.1 Single-Byte Character Set Input Method

The Single-Byte Character Set Input Method (SIM) is the standard that supports most of the locales. It is a mapping function that supports simple composition defined on workstation keyboards associated with single-byte locales.

SIM supports any keyboard, code set, and language that the keycomp command can describe. You can customize SIM using imkeymaps. The coded strings returned by the input method depend on the imkeymap. Due to this feature, you

can extend the capability of the SIM to support multi-byte UTF-8 locales. Adding just one new key combination (AltGr+e, AltGr+4 or AltGr+5) to make the input of the Euro sign from the keyboard feasible does not require an entire input method to be rewritten.

Similar to the German DE_DE alias for the DE_DE.UTF-8 locale contains the following entry in the keycomp source file /usr/lib/nls/loc/DE_DE.UTF-8.imkeymap.src:

```
...
XK_EuroSign                                          \
                    "\xe2\x82\xac"                   \
                    XK_EuroSign                      \
                    U                                \
                    U                                \
                    U                                \
                    U                                \
                    U
...
XK_e                                                 \
                    'e'                              \
                    XK_E                             \
                    XK_E                             \
                    XK_e                             \
                    '\x05'                           \
                    U                                \
                    XK_EuroSign
...
```

The XK_e maps to the XK_EuroSign keysym when the modifier key AltGr is used while entering the character e on the keyboard. The XK_EuroSign keysym, in turn, is mapped to the code point \xe2\x82\xac.

10.5.4.2 UNIVERSAL Input Method

If you configured your system to use a UTF-8 locale for Euro symbol support, you have access to the entire range of UTF-8 encoded characters. In order to allow for character input from the keyboard for several thousand different characters, the single-byte input method is not sufficient, and consequently, a UNIVERSAL input method is provided. Note, if your system is configured for a UTF-8 locale that does not support the character input of the Euro sign through a key combination, the UNIVERSAL input method has to be used.

The UNIVERSAL input method supports two general modes of character input:

- Switching between installed AIX locale input methods
- Selection of characters from character lists

The UNIVERSAL input method is configured by the default UNIVERSAL.imcfg file in the /usr/lib/nls directory. This file defines the set of available AIX locale input methods and the lists of characters for list-based input. It also defines the key combinations used to invoke the input method selection menus.

The default key combinations recognized by the UNIVERSAL input method are:

Ctrl+Alt+i	Invokes the input method selection menu
Ctrl+Alt+l	Invokes the menu of character lists
Ctrl+Alt+c	Invokes the current list of characters

A selection menu can be closed by pressing the Enter key.

After an input method is selected from the input method selection menu (see Figure 53 on page 264), it can be used in the same way as in its associated locale. The label at the left edge of the input method status bar describes the locale for the currently selected input method. For example, if the Simplified Chinese input method is selected from the selection menu, the CN_ZH label is displayed in the status area. Any status information specific to the current input method is displayed immediately to the right of this label. Of course, if you want to switch the input method, the related locale has to be installed on the system.

Most input methods require a locale-specific keyboard mapping to support certain input sequences. In the X environment, the keyboard can be remapped by invoking the `xmodmap` command with the keyboard files under the /usr/lpp/X11/defaults/xmodmap directory. For example, the following command remaps the keyboard so as to support the Chinese input method:

```
# xmodmap /usr/lpp/X11/defaults/xmodmap/CN_ZH/keyboard
```

Note that the keyboard mappings assume an associated physical keyboard.

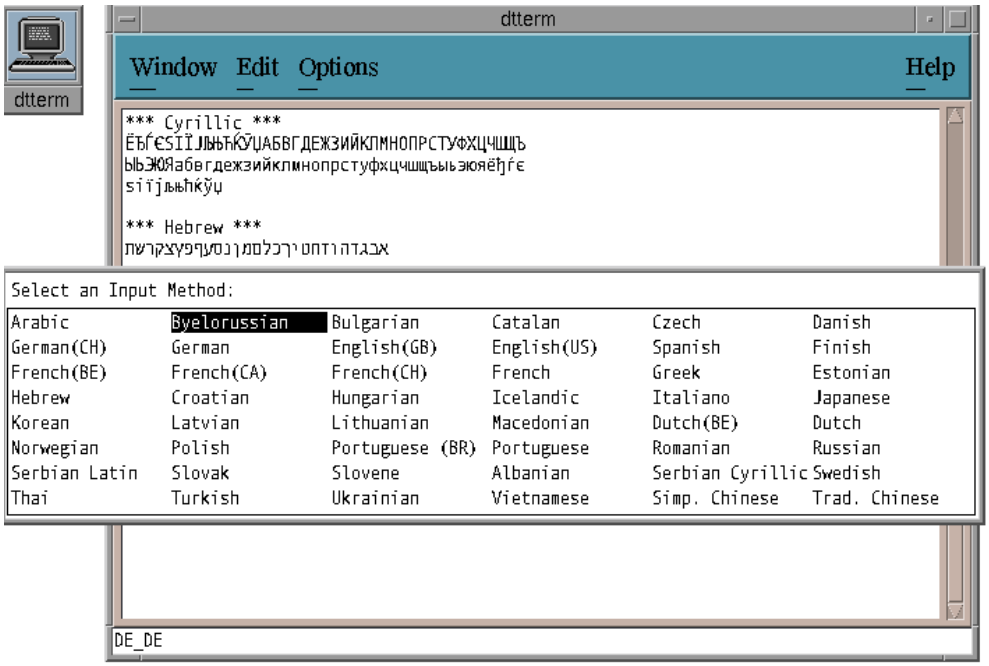


Figure 53. UNIVERSAL Input Method: Switching

By default, the Ctrl+Alt+I key combination invokes the menu of character lists (Figure 54 on page 265). After a list is selected from this menu, characters can be input by selecting them from the list with the mouse. The characters will initially be inserted in the input method pre-edit area. The characters can be *committed* by pressing the Enter key.

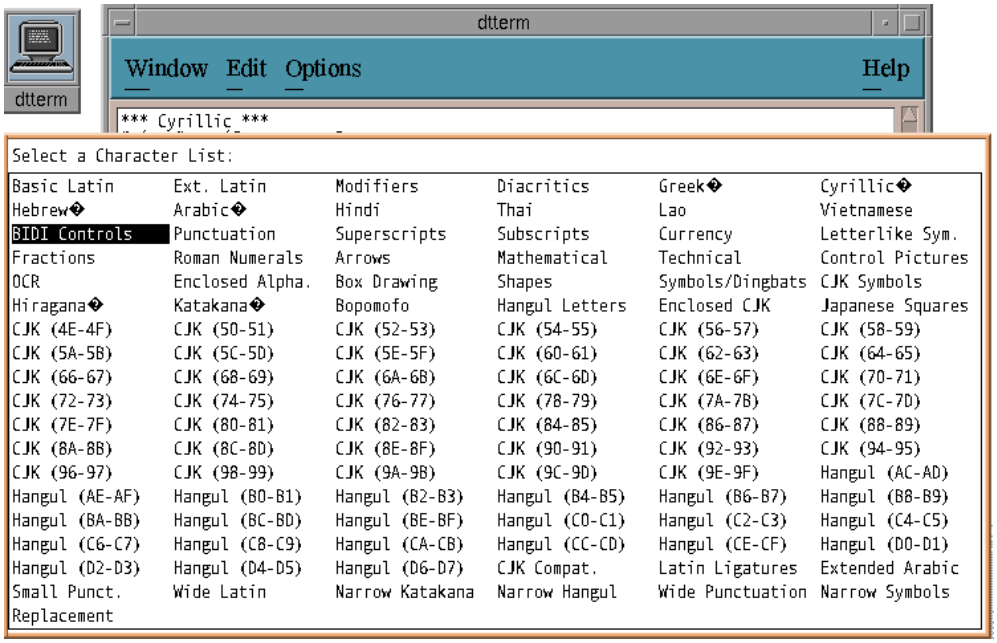


Figure 54. UNIVERSAL Input Method: Character List Selection

When a character list is selected from the character list menu, it becomes the current character list. It can then be invoked by simply entering the default Ctrl+Alt+c key combination (Figure 55 on page 266).

To enter the Euro sign, select the Currency character list and select the Euro sign from the list.

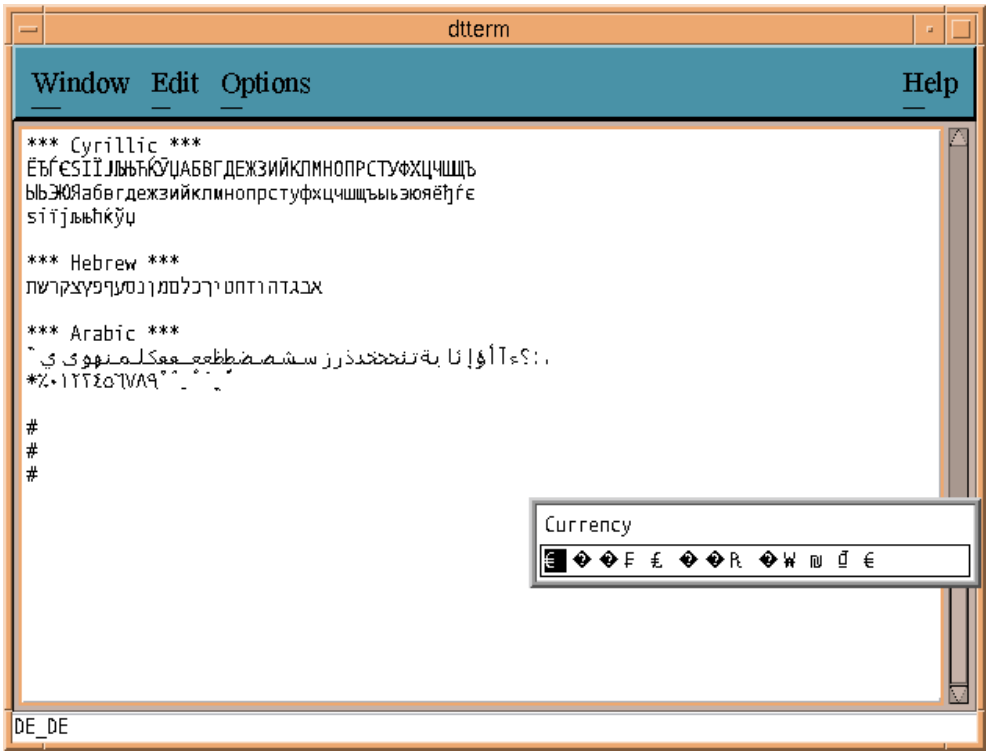


Figure 55. UNIVERSAL Input Method: Character List

10.5.5 Codeset Conversion Tables

As more code sets are supported, it becomes important not to clutter programs with the dependency of any particular code set. This is known as code set independence. To aid in code set independence, NLS supplies converters that translate character encoding values found in different code sets. Using these converters, a system can accurately process data generated in different code set environments

In addition to Unicode, a number of new 8-bit codesets have been proposed that will contain the Euro symbol. Although AIX will not have supported locales in many of these code sets, it will provide data conversion capabilities to, and from, these code sets. The new code sets that are being introduced are as follows:

- IBM-1252 (extended) - This code set will be used by the Euro single-byte (SBCS) migration option.
- IBM-858 - This code set is identical to IBM-850 except that the dotless letter i, which is used only in Turkish, is replaced with the Euro symbol. This code set will be used in OS/2.
- IBM-114x - Ten new code sets are being introduced that serve the same function as the existing EBCDIC code sets, except that one character is replaced with the Euro. The new code sets are summarized in Table 55.

Table 55. Existing EBCIDIC Code Sets

Existing EBCDIC Code Set	New Euro Code Set	Countries
IBM-037	IBM-1140	US
IBM-273	IBM-1141	Germany

Existing EBCDIC Code Set	New Euro Code Set	Countries
IBM-277	IBM-1142	Denmark, Norway
IBM-278	IBM-1143	Finland, Sweden
IBM-280	IBM-1144	Italy
IBM-284	IBM-1145	Spain
IBM-285	IBM-1146	Great Britain
IBM-297	IBM-1147	France
IBM-500	IBM-1148	Belgium, Canada, Switzerland
IBM-871	IBM-1149	Iceland

The new iconv converters use either the UCSTBL or the Universal_UCS_Conv conversion method.

The UCSTBL method is located in the /usr/lib/nls/loc/uconv directory and loads UCS-2 (Unicode) conversion tables created by the `uconvdef` command. In order to compile the UCS-2 (Unicode) conversion table, the `uconvdef` command reads a source file that defines a mapping between the UCS-2 and a particular multi-byte code set. The source files are in the /usr/lib/nls/uconv table directory with names that are composed of the code set name appended by the suffix .ucmap. The UCSTBL method uses the table to support UCS-2 conversions in both directions. The setup of a converter is complete if the proper symbolic links for conversions in each direction are created in the /usr/lib/nls/iconv directory. For example, the links for the conversion of the IBM-850 code set to UCS-2, and vice versa, are defined as shown below:

```
# cd /usr/lib/nls/loc/iconv
# ln -s /usr/lib/nls/loc/uconv/UCSTBL IBM-850_UCS-2
# ln -s /usr/lib/nls/loc/uconv/UCSTBL UCS-2_IBM-850
```

The Universal_UCS_Conv method is located in the /usr/lib/nls/loc/iconv directory and can be used to convert between any two code sets whose conversions to, and from, UCS-2 are defined. The conversion is instantiated by setting the proper links. Assuming that someone wants to define the conversion of the IBM-850 to, and from, the UTF-8 code set, you would have to enter the following sequence of commands:

```
# cd /usr/lib/nls/loc/iconv
# ln -s /usr/lib/nls/loc/uconv/Universal_UCS_Conv IBM-850_UTF-8
# ln -s /usr/lib/nls/loc/uconv/UCSTBL IBM-850_UCS-2
# ln-s /usr/lib/nls/loc/uconv/UCSTBL UCS-2_UTF-8
# ln -s /usr/lib/nls/loc/uconv/UCSTBL IBM-850
# ln -s /usr/lib/nls/loc/uconv/UCSTBL UTF-8
```

As you can see from the example above, UTF-8 converters are usually done by using the Universal_UCS_Conv and the /usr/lib/nls/loc/uconv/UTF-8 conversion.

It should be noted that, for some conversion pairs, a substitution character may be inserted when converting a Euro symbol from a codeset that contains the Euro to a codeset that does not (For example: Converting IBM-114x to ISO8859-1).

Table 56 summarizes the new iconv converters that are provided for Euro support.

Table 56. Converters for Euro Symbol Support

Converter	Type	Fileset
IBM-1140_IBM-858	Universal	bos.iconv.com
IBM-1140_ISO8859-1	Universal	bos.iconv.com
IBM-1140_UCS-2	UCSTBL	bos.iconv.ucs.ebcdic
IBM-1140_UTF-8	Universal	bos.iconv.ucs.ebcdic
IBM-1141_IBM-858	Universal	bos.iconv.de_DE
IBM-1141_ISO8859-1	Universal	bos.iconv.de_DE
IBM-1141_UCS-2	UCSTBL	bos.iconv.ucs.ebcdic
IBM-1142_IBM-858	Universal	bos.iconv.da_DK
IBM-1142_ISO8859-1	Universal	bos.iconv.da_DK
IBM-1142_UCS-2	UCSTBL	bos.iconv.ucs.ebcdic
IBM-1142_UTF-8	Universal	bos.iconv.ucs.ebcdic
IBM-1143_IBM-858	Universal	bos.iconv.com
IBM-1143_ISO8859-1	Universal	bos.iconv.com
IBM-1143_UCS-2	UCSTBL	bos.iconv.ucs.ebcdic
IBM-1143_UTF-8	Universal	bos.iconv.ucs.ebcdic
IBM-1144_IBM-858	Universal	bos.iconv.it_IT
IBM-1144_ISO8859-1	Universal	bos.iconv.it_IT
IBM-1144_UCS-2	UCSTBL	bos.iconv.ucs.ebcdic
IBM-1144_UTF-8	Universal	bos.iconv.ucs.ebcdic
IBM-1145_IBM-858	Universal	bos.iconv.es_ES
IBM-1145_ISO8859-1	Universal	bos.iconv.es_ES
IBM-1145_UCS-2	UCSTBL	bos.iconv.ucs.ebcdic
IBM-1145_UTF-8	Universal	bos.iconv.ucs.ebcdic
IBM-1146_IBM-858	Universal	bos.iconv.en_GB
IBM-1146_ISO8859-1	Universal	bos.iconv.en_GB
IBM-1146_UCS-2	UCSTBL	bos.iconv.ucs.ebcdic
IBM-1146_UTF-8	Universal	bos.iconv.ucs.ebcdic
IBM-1147_IBM-858	Universal	bos.iconv.fr_FR
IBM-1147_ISO8859-1	Universal	bos.iconv.fr_FR
IBM-1147_UCS-2	UCSTBL	bos.iconv.ucs.ebcdic
IBM-1147_UTF-8	Universal	bos.iconv.ucs.ebcdic

Converter	Type	Fileset
IBM-1148_IBM-858	Universal	bos.iconv.com
IBM-1148_UCS-2	UCSTBL	bos.iconv.ucs.ebcdic
IBM-1148_UTF-8	Universal	bos.iconv.ucs.ebcdic
IBM-1149_IBM-858	Universal	bos.iconv.is_IS
IBM-1149_ISO8859-1	Universal	bos.iconv.is_IS
IBM-1149_UCS-2	UCSTBL	bos.iconv.ucs.ebcdic
IBM-1149_UTF-8	Universal	bos.iconv.ucs.ebcdic
IBM-850_IBM-858	Universal	bos.iconv.ucs.com
IBM-858_IBM-1140	Universal	bos.iconv.com
IBM-858_IBM-1141	Universal	bos.iconv.de_DE
IBM-858_IBM-1142	Universal	bos.iconv.da_DK
IBM-858_IBM-1143	Universal	bos.iconv.com
IBM-858_IBM-1144	Universal	bos.iconv.it_IT
IBM-858_IBM-1145	Universal	bos.iconv.es_ES
IBM-858_IBM-1146	Universal	bos.iconv.en_GB
IBM-858_IBM-1147	Universal	bos.iconv.fr_FR
IBM-858_IBM-1148	Universal	bos.iconv.com
IBM-858_IBM-1149	Universal	bos.iconv.is_IS
IBM-858_IBM-850	Universal	bos.iconv.ucs.com
IBM-858_ISO8859-1	Universal	bos.iconv.ucs.com
IBM-858_UCS-2	UCSTBL	bos.iconv.ucs.com
IBM-858_UTF-8	Universal	bos.iconv.ucs.com
IBM_1141_UTF-8	Universal	bos.iconv.ucs.ebcdic
IBM_1148_ISO8859-1	Universal	bos.iconv.com
ISO8859-1_IBM-1140	Universal	bos.iconv.com
ISO8859-1_IBM-1141	Universal	bos.iconv.de_DE
ISO8859-1_IBM-1142	Universal	bos.iconv.da_DK
ISO8859-1_IBM-1143	Universal	bos.iconv.com
ISO8859-1_IBM-1145	Universal	bos.iconv.es_ES
ISO8859-1_IBM-1146	Universal	bos.iconv.en_GB
ISO8859-1_IBM-1147	Universal	bos.iconv.fr_FR
ISO8859-1_IBM-1148	Universal	bos.iconv.com
ISO8859-1_IBM-1149	Universal	bos.iconv.is_IS
ISO8859-1_IBM-858	Universal	bos.iconv.ucs.com

Converter	Type	Fileset
UCS-2_IBM-1140	UCSTBL	bos.iconv.ucs.ebcdic
UCS-2_IBM-1141	UCSTBL	bos.iconv.ucs.ebcdic
UCS-2_IBM-1142	UCSTBL	bos.iconv.ucs.ebcdic
UCS-2_IBM-1143	UCSTBL	bos.iconv.ucs.ebcdic
UCS-2_IBM-1144	UCSTBL	bos.iconv.ucs.ebcdic
UCS-2_IBM-1145	UCSTBL	bos.iconv.ucs.ebcdic
UCS-2_IBM-1146	UCSTBL	bos.iconv.ucs.ebcdic
UCS-2_IBM-1147	UCSTBL	bos.iconv.ucs.ebcdic
UCS-2_IBM-1148	UCSTBL	bos.iconv.ucs.ebcdic
UCS-2_IBM-1149	UCSTBL	bos.iconv.ucs.ebcdic
UCS-2_IBM-858	UCSTBL	bos.iconv.ucs.com
UTF-8_IBM-1140	Universal	bos.iconv.ucs.ebcdic
UTF-8_IBM-1142	Universal	bos.iconv.ucs.ebcdic
UTF-8_IBM-1143	Universal	bos.iconv.ucs.ebcdic
UTF-8_IBM-1144	Universal	bos.iconv.ucs.ebcdic
UTF-8_IBM-1145	Universal	bos.iconv.ucs.ebcdic
UTF-8_IBM-1146	Universal	bos.iconv.ucs.ebcdic
UTF-8_IBM-1147	Universal	bos.iconv.ucs.ebcdic
UTF-8_IBM-1148	Universal	bos.iconv.ucs.ebcdic
UTF-8_IBM-1149	Universal	bos.iconv.ucs.ebcdic
UTF-8_IBM_1141	Universal	bos.iconv.ucs.ebcdic
UTF-8_IBM_858	Universal	bos.iconv.ucs.com

10.5.6 Euro SBCS Migration Option - IBM-1252 Locale

Use of a Unicode (UTF-8) locale provides the most standardized and widely accepted way of achieving Euro support on AIX. However, not all applications or customers, are ready to migrate to the Unicode solution at this time. The Euro Single-Byte Code Set (SBCS) migration option provides a temporary solution for those customers who need Euro support immediately but are not yet ready to migrate to Unicode.

The migration option consists of an additional set of locales bound to the IBM-1252 code set (same as Windows 1252 extended) for each language/territory listed in Table 51 on page 256. ISO8859-1 is a proper subset of IBM-1252, and as such, all data currently encoded in ISO8859-1 can be used in the IBM-1252 environment without any requirement for data conversion. The IBM-1252 code set differs in relation to the industry standard code set ISO8859-1 to the extend that additional graphic characters are added in the ISO control characters range from 0x80 through 0x9F. The Euro character is defined at code point 0x80 in the IBM-1252 environment, which is identical to the Windows NT

and Windows 98 value. This will provide some level of compatibility for Windows clients running AIX server applications.

While all code page 1252 graphics will be processed correctly, the AIX font set will only be extended to support the Euro symbol not the additional unique graphics in code page 1252. Customers selecting this approach are cautioned to review display requirements for 1252 graphics.

Furthermore, assigning characters to the ASCII control points in the code page 1252 can be a significant problem for customers who use ASCII terminals or have applications that use ASCII terminal emulators. Trying to use ASCII terminals with the Euro symbol will require additional customization such as fonts or INOUT transforms, or will be limited to the existing non-euro character sets. AIX provides terminal definition tables that may have to be modified depending on the usage. Since AIX locales can be changed based on an application basis, the only applications to be reviewed for potential customization are those applications that require ASCII terminal input and/or ASCII terminal emulation support and must handle the Euro at the same time.

Usage of the SBCS migration option will require the use of the fully qualified locale and code set name, as follows:

For example, in order to use the German locale with the Euro SBCS migration option, the user would set LANG=de_DE.IBM-1252. This would provide the proper code set environment for the Euro character but would still format monetary quantities in the traditional national currency format. In order to specify Euro currency formatting, the user would additionally set LC_MONETARY=de_DE.IBM-1252@euro.

Keyboard mappings, including the Euro and bound to IBM-1252, are provided as links to the related ISO keyboard maps.

Although SBCS migration option locales are bound to IBM-1252, full font support for the additional group of characters in IBM-1252 that is not currently defined in ISO8859-1 (other than the Euro) is not guaranteed, nor is it guaranteed that all existing ISO8859-1 fonts will be extended to include the Euro or additional characters. At a minimum, the set of fonts corresponding to the Common Desktop Environment (CDE) dt aliases for IBM-1252 (such as "-dt-interface system-*" and "-dt-interface user-*") are enhanced to include the Euro.

Documentation is provided that will instruct customers who are using IBM-850 based locales how they can achieve Euro support on AIX by creating their own customized locale using the localedef facility. Refer to AIX Version 4.3 Base Documentation and AIX Version 4.3 Extended Documentation.

10.5.7 Packaging

Each Unicode based locale listed in Table 51 on page 256, and commonly referred to by XX_XX in this paragraph, is separately installable and contains the complete set of function to provide both traditional monetary as well as Euro monetary formatting. All locale specific modules are separately installable and are grouped and distributed in two different entities:

- bos.loc.utf.XX_XX fileset
- X11.loc.XX_XX package comprised of the filesets:

- X11.loc.XX_XX.Dt.rte
- X11.loc.XX_XX.base.lib
- X11.loc.XX_XX.base.rte

The scope of the files in the bos.loc.utf.XX_XX fileset is limited to provide the locale support for the Base Operating System (BOS) and the X11.loc.XX_XX package will add the locale support for the X environment. Several filesets will be automatically installed if the installation process can not find them on the system:

- bos.loc.com.utf (coreq. to bos.loc.utf.XX_XX)
- X11.fnt.ucs.ttf (coreq. to X11.loc.base.rte)

For a complete list of dependencies, examine the output of the suitable `lslpp` command after you installed the filesets on your system. For example, let your UTF-8 locale identifier be DE_DE, so you will get for the bos.loc.utf.DE_DE fileset the following requisites:

```
# lslpp -p "bos.loc.utf.DE_DE"

  Fileset              Requisites
  ----------------------------------------------------------------------------
Path: /usr/lib/objrepos
  bos.loc.utf.DE_DE 4.3.2.0
                       *coreq bos.loc.com.utf 4.3.2.0
```

For the X11.loc.DE_DE package, the `lslpp -p` command will yield:

```
# lslpp -p "X11.loc.DE_DE*"
  Fileset              Requisites
  ----------------------------------------------------------------------------
Path: /usr/lib/objrepos
  X11.loc.DE_DE.Dt.rte 4.3.2.0
                       *instreq X11.Dt.rte 4.3.2.0
  X11.loc.DE_DE.base.lib 4.3.2.0
                       *instreq X11.base.common 4.3.2.0
  X11.loc.DE_DE.base.rte 4.3.2.0
                       *coreq X11.fnt.ucs.ttf 4.3.2.0
                       *instreq X11.base.rte 4.3.2.0

Path: /etc/objrepos
  X11.loc.DE_DE.Dt.rte 4.3.2.0
                       *instreq X11.Dt.rte 4.3.2.0
```

Locale definitions for the Euro single-byte migration option is packaged in the same filesets as their non-Euro Latin-1 counterparts; that is, bos.loc.iso.xx_XX where xx_XX represents the language and territory designation. The bos.iconv.xx_XX filesets are requisites for the bos.loc.iso.xx_XX filesets.

10.5.8 Installation of Euro Symbol Support

This section is intended to give a step-by-step instruction for the configuration changes a system administrator has to accomplish in order to make an AIX system ready to support the Euro symbol. The UTF-8 and the SBCS environment for the German language and territory designation DE_DE, or de_DE.IBM-1252, are covered respectively. All the following statements apply to any of the language/territory combinations that are listed in Table 51 on page 256, you just replace DE_DE or de_DE.IBM-1252 by the locale identifier of your choice.

An assumption is that the AIX system was installed using the following default settings for the new and complete override installation for the primary language environment (RS/6000 Hardware with German keyboard attached):

```
...
2 Primary Language Environment Settings (AFTER Install):
     Cultural Convention ..............  English (United States)
     Language .........................  English (United States)
     Keyboard .........................  German
     Keyboard Type ....................  Default
...
```

Note, the UTF-8 Unicode locales are not available as optional choices for the primary language environment settings at the time of installation.

After installation the output of the `locale` command would be:

```
# locale
LANG=en_US
LC_COLLATE="en_US"
LC_CTYPE="en_US"
LC_MONETARY="en_US"
LC_NUMERIC="en_US"
LC_TIME="en_US"
LC_MESSAGES="en_US"
LC_ALL=
```

The software keyboard for the lft0 pseudo device driver would be:

```
# lskbd
The current software keyboard map = /usr/lib/nls/loc/de_DE.lftkeymap
```

The X server, and hence the Common Desktop Environment, will use the /usr/lpp/X11/defaults/xmodmap/de_DE/keyboard file to configure the German keyboard map at startup.

There are two different ways to gain UTF-8 Euro symbol support. Either you decide to have your system default environment configured to be UTF-8 or you prefer to offer the locale just as an additional language environment on the system.

10.5.8.1 Euro UTF-8: Additional Language Environment

In order to activate Euro symbol support using the German UTF-8 locale as an additional language environment, insert one of the AIX BOS CDs in the CD-ROM drive, issue the `smitty mlang` command, select the **Add Additional Language Environments** item from the menu, and press Enter. In the Add Additional Language Environment menu, you enter, by the aide of the F4 function key, the `UTF-8 German[DE_DE]` identifier in the appropriate entry fields as shown in Figure 56 on page 274.

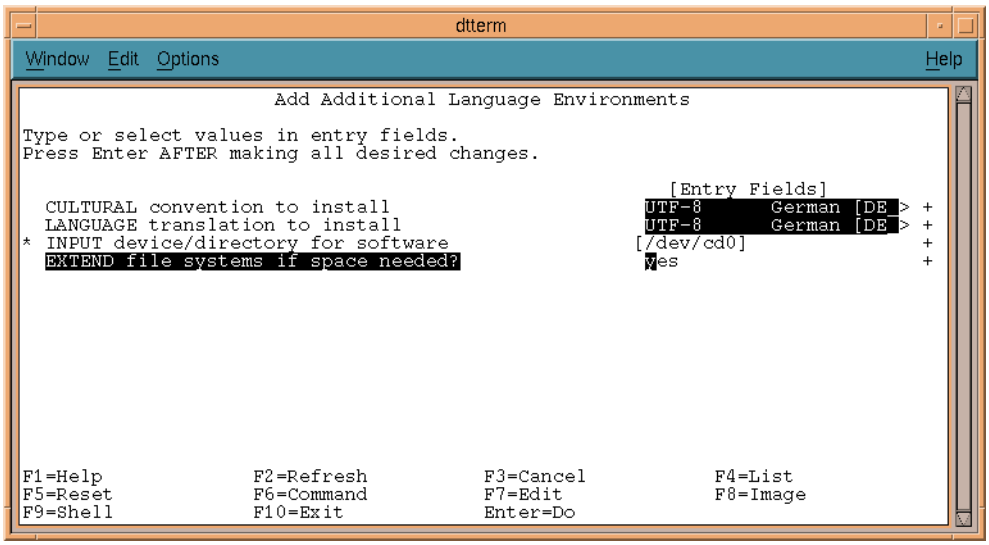


Figure 56. German UTF-8: Add Additional Language Environment

SMIT will install all necessary filesets for the locale support and you will receive the following output after SMIT has completed its task:

```
...

Installation Summary
--------------------
Name                        Level           Part        Event       Result
-------------------------------------------------------------------------------
bos.loc.com.utf             4.3.2.0         USR         APPLY       SUCCESS
X11.loc.DE_DE.base.rte      4.3.2.0         USR         APPLY       SUCCESS
X11.loc.DE_DE.base.lib      4.3.2.0         USR         APPLY       SUCCESS
X11.loc.DE_DE.Dt.rte        4.3.2.0         USR         APPLY       SUCCESS
X11.loc.DE_DE.Dt.rte        4.3.2.0         ROOT        APPLY       SUCCESS
X11.fnt.ucs.ttf             4.3.2.0         USR         APPLY       SUCCESS
bos.loc.utf.DE_DE           4.3.2.0         USR         APPLY       SUCCESS
bos.msg.DE_DE.net.tcp.cli   4.3.2.0         USR         APPLY       SUCCESS
bos.msg.DE_DE.rte           4.3.2.0         USR         APPLY       SUCCESS

---- end ----
```

The fileset bos.loc.com.utf is a requisite for the bos.loc.utf.DE_DE fileset and supplies the common locale support for UTF-8. Also, the fileset X11.fnt.ucs.ttf is pulled as a requisite by the installation process and gives you the indispensable `AIXwindows Unicode TrueType Font` support. All UTF fonts in AIX are TrueType fonts.

Now a user can begin executing in the DE_DE UTF-8 locale environment by setting the LANG environment variable to DE_DE:

```
export LANG=DE_DE
```

The `locale` command will return:

```
# locale
DE_DELANG=DE_DE
LC_COLLATE="DE_DE"
```

```
LC_CTYPE="DE_DE"
LC_MONETARY="DE_DE"
LC_NUMERIC="DE_DE"
LC_TIME="DE_DE"
LC_MESSAGES="DE_DE"
LC_ALL=
```

With this setting, all internationalized programs will execute in the German UTF-8 locale environment. If the Euro currency formatting is desired, you must change the LC_MONETARY variable in the following way:

```
export LC_MONETARY=DE_DE@euro
```

There is no need to complete the installation process and the system configuration by remapping the X server keyboard. The X server keyboard map /usr/lpp/X11/defaults/xmodmap/DE_DE/keyboard is implemented as a link to the German ISO keyboard map that is already set for your system.

Use the AltGr+e key combination to enter the Euro symbol through your keyboard or alternatively take advantage of the UNIVERSAL Input Method, invoke the menu of the character list by Ctrl+Alt+I, and enjoy the variety of different characters that are now available to you. For more information about the new input method, refer to the Chapter 10.5.4.2, “UNIVERSAL Input Method” on page 263.

10.5.8.2 Euro UTF-8: Primary Language Environment

In order to activate Euro symbol support using the German UTF-8 locale as primary language environment, insert one of the AIX BOS CDs in the CD-ROM drive and issue:

```
# smitty chlang
```

on the command line and enter by the aide of the appropriate function keys the `UTF-8 German[DE_DE]` identifier in the relevant entry fields as shown in Figure 57.

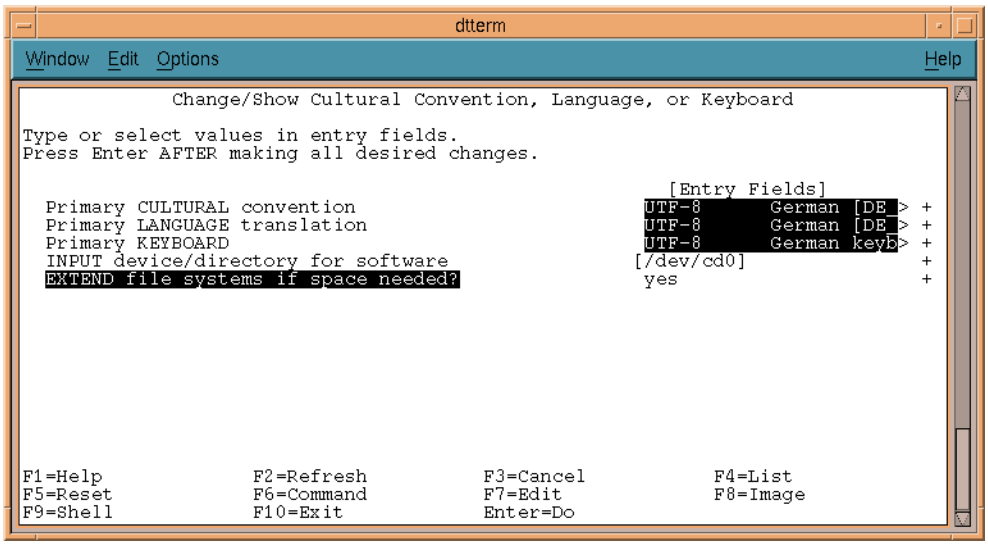


Figure 57. German UTF-8: Change/Show Cultural Convention, Lang., or Keyboard

SMIT uses the chlang command to set the environment variable LANG to DE_DE in the /etc/environment file and modifies the lft0 pseudo device driver's predefined ODM attribute swkb_path to match the UTF-8 keymap /usr/lib/nls/loc/DE_DE.lftkeymap. If necessary the chdev command changes the predefined primary and secondary font path of the lft0 pseudo device driver in the ODM to be /usr/lpp/fonts/Erg22.iso1.snf and /usr/lpp/fonts/Erg11.iso1.snf respectively. Furthermore, SMIT will install all necessary filesets for the BOS locale support and you will receive the following output after SMIT completed its task:

```
...

SUCCESSES
---------
  Filesets listed in this section passed pre-installation verification
  and will be installed.

  Selected Filesets
  -----------------
  bos.loc.utf.DE_DE 4.3.2.0                  # Base System Locale UTF Code...
  bos.msg.DE_DE.net.tcp.client
  bos.msg.DE_DE.rte

  Requisites
  ----------
  (being installed automatically;  required by filesets listed above)
  bos.loc.com.utf 4.3.2.0                    # Common Locale Support - UTF-8

  << End of Success Section >>



Installation Summary
--------------------
Name                        Level           Part        Event       Result
-------------------------------------------------------------------------------
bos.loc.com.utf             4.3.2.0         USR         APPLY       SUCCESS
bos.loc.utf.DE_DE           4.3.2.0         USR         APPLY       SUCCESS
bos.msg.DE_DE.net.tcp.cli   4.3.2.0         USR         APPLY       SUCCESS
bos.msg.DE_DE.rte           4.3.2.0         USR         APPLY       SUCCESS
lft0 changed

---- end ----
```

The lft0 pseudo device driver does not allow for a dynamic reconfiguration, and consequently, you must reboot the system to put the new software keyboard map, the primary, and the secondary font paths in effect. But prior to the reboot, it is absolutely essential to install the X11 locale support for the new UTF-8 locale of choice on your system. The command smit install_latest brings you to the proper SMIT menu. Fill in X11.loc.DE_DE in the SOFTWARE to install entry filed. SMIT displays the following:

```
...
Installation Summary
--------------------
Name                        Level           Part        Event       Result
-------------------------------------------------------------------------------
-
```

```
X11.loc.DE_DE.base.rte      4.3.2.0         USR         APPLY       SUCCESS
X11.loc.DE_DE.base.lib      4.3.2.0         USR         APPLY       SUCCESS
X11.loc.DE_DE.Dt.rte        4.3.2.0         USR         APPLY       SUCCESS
X11.loc.DE_DE.Dt.rte        4.3.2.0         ROOT        APPLY       SUCCESS
X11.fnt.ucs.ttf             4.3.2.0         USR         APPLY       SUCCESS

---- end ----
```

The fileset X11.fnt.ucs.ttf, pulled as a requisite by the installation process, gives you the AIXwindows Unicode TrueType Font support. All UTF fonts in AIX are TrueType fonts.

Optionally, you may modify the /etc/environment file to activate, on a system wide scope, the new Euro currency formatting. Add the following line:

```
LC_MONETARY=DE_DE@euro
```

Now you are ready to reboot your system. When the system comes up, and you are logged in again, you can enter the Euro symbol through the AltGr+e key combination, and by the aide of the UNIVERSAL input method, you will have access to several thousand different characters. Just use the Ctrl+Alt+I key combination, select the character list you want, and enter the characters out of the pop-up menu using a mouse click. If you are looking for a sample file that is actually encoded in UTF-8, install the Developers Toolkit for Unicode fileset bos.loc.adt.unicode and use the vi editor to examine the /usr/lib/nls/Unicode/samples/ut8.txt file.

10.5.8.3 IBM-1252 Code Set Euro Symbol Support

Within the framework of this test set up (LANG=en_US and German keyboard software support by swkd_path=/usr/lib/nls/loc/de_DE.lftkeymap), the IBM-1252 locale support for the BOS is already installed. The software keyboard support is contained in the same fileset as the IBM-1252 locale support: bos.loc.iso.de_DE.

You can verify that the IBM-1252 locale resides on the system. This means the locale database, the input method, the input method keymap, the LC_MONETARY locale database, and the 64-bit object modules are installed:

```
# ls -l /usr/lib/nls/loc/de_DE.IBM-1252*  | cut -c55-

/usr/lib/nls/loc/de_DE.IBM-1252
/usr/lib/nls/loc/de_DE.IBM-1252.im -> /usr/lib/nls/loc/sbcs.im
/usr/lib/nls/loc/de_DE.IBM-1252.imkeymap
/usr/lib/nls/loc/de_DE.IBM-1252@euro
/usr/lib/nls/loc/de_DE.IBM-1252@euro__64
/usr/lib/nls/loc/de_DE.IBM-1252__64
```

In case the German keyboard would not have been attached and configured you may first have to verify that the bos.loc.iso.xx_XX fileset of the desired locale is on the system. (xx_XX refers to one of the locale identifier as listed in Table 51 on page 256 modified to the extent that the first two letters are converted to lower case.) Use the `lslpp` command to accomplish this task. The lslpp command returns:

```
# lslpp -l "bos.loc.iso.*"
  Fileset                      Level  State      Description
  ----------------------------------------------------------------------------
Path: /usr/lib/objrepos
```

```
  bos.loc.iso.de_DE          4.3.2.0  COMMITTED  Base System Locale ISO Code
                                                 Set - German
  bos.loc.iso.en_US          4.3.2.0  COMMITTED  Base System Locale ISO Code
                                                 Set - U.S. English
```

The bos.loc.iso.de_DE is installed, and hence, no action is required.

If for some reason the bos.loc.iso.de_DE is missing on your system, you will have to install the fileset. The command `smit install_latest` brings you to the proper SMIT menu. Fill in bos.loc.iso.de_DE in the `SOFTWARE to install` entry field and press Enter. SMIT reports the progress.

Since you verified the existence of the BOS de_DE locale support, add the X11 locale support next. The package that must be installed is named X11.loc.de_DE, and it is recommend to use SMIT in conjunction with the fastpath `install_latest` to lay the foundation for the graphical locale support. Fill in X11.loc.de_DE in the `SOFTWARE to install` entry field, let SMIT do the work, and wait until you see the following lines at the end of the output:

```
...
Installation Summary
--------------------
Name                        Level           Part        Event       Result
-------------------------------------------------------------------------------
X11.loc.de_DE.base.rte      4.3.2.0         USR         APPLY       SUCCESS
X11.loc.de_DE.base.lib      4.3.2.0         USR         APPLY       SUCCESS
X11.loc.de_DE.Dt.rte        4.3.2.0         USR         APPLY       SUCCESS
X11.loc.de_DE.Dt.rte        4.3.2.0         ROOT        APPLY       SUCCESS

---- end ----
```

Set your LANG environment variable to de_DE.IBM-1252, export LANG and open a new dtterm that is aware of your new locale by the `/usr/dt/bin/dtterm` command. Note, that you do not have to introduce a new keymap to the X server since you are already using the German ISO keymap. Now you can enter the Euro symbol by the AltGr+e key combination. The Euro currency formatting will be to your service as soon as the LC_MONETARY environment variable is set and exported:

```
# export LC_MONETARY=de_DE.IBM-1252@euro
```

It should be mentioned that there is intentionally no SMIT support through the fast paths `chlang` and `mle_add_lang`. for the SBCS migration option. This means the smit menus **Change/Show Primary Language Environment** (`smit chlang`) and **Add Additional Language Environments** (smit mle_add_lang) will not allow you to establish the Euro symbol support through the IBM-1252 code set.

10.6 National Language Enhancements

The following enhancements have been made to national language support in AIX Version 4.3 and AIX Version 4.3.2.

10.6.1 Byelorussian and Ukrainian Localization

AIX Version 4.3 has introduced support for the Byelorussian and Ukrainian localization. Both languages use the existing ISO8859-5 codeset that contains all

of the characters needed to display the languages. The Localization provides the following items:

- A new input method for the input of characters to the Byelorussian and Ukrainian languages.
- Localized conventions are defined for Byelorussian and Ukrainian imkeymaps and lftkeymaps.
- Localized resource files for X clients.
- Additions to X locale database (I18N).
- Localized X keyboard mapping.

10.6.2 Thai Language Support

Previous to AIX 4.3, Thai language support was based on a solution from IBM Thailand. AIX Version 4.3 formally implements support for the Thai language.

The features implemented in AIX Version 4.3 include:

Locale The Thai locale is based on either TIS620-1 or Unicode Version 2. The wide character processing for UTF-8 Thai locales is based on Unicode.

Input Method
The Thai input method provides users with a facility for composing characters by combining several key strokes to produce a single Thai character.

cmdpios AIX Version 4.3 provides a printer filter for Thai/Vietnamese that supports downloadable UCS-2 fonts and can print data for both Thai and UTF-8 locales.

cmdiconv New iconv converters are present for the following Thai language codesets:

•TIS620-1

•Unicode

charmap New character map for Thai code set definition.

cmdtty Thai csmap is included in AIX Version 4.3.

xmodmap Localized X keysyms and server keyboard maps are present for Thai.

Aixterm Enhanced to support the Thai language.

imkeymap AIX Input Method keymaps (imkeymaps) for the Thai language have been added.

Motif Motif contains enhancements to support of Thai language.

Unicode Unicode support for the Thai language has been added.

10.6.2.1 Systems Management

The Thai locales are configurable just like any other locale in the system using the SMIT panel Manage Language Environment.

The user can install the following items:

- Thai Cultural Conventions (TIS620-1)

- Thai Cultural Conventions (Unicode)

10.6.2.2 Standards Compliance of Thai Language Support

The UTF-8 support for the Thai locale consists of the entire Base Multilingual Plane of ISO 10646. It complies with the following standards:

- X/Open Portability Guide, issue 3 and issue 4
- POSIX.2
- X Window System Version 11 Release 5
- Unicode Version 2.0 for display and fonts

Note: The localization components for this locale will only support the Thai and ISO8859-1 characters.

10.6.2.3 Application Binary Interface (ABI)

The Thai locale, input method, and conversions comply with the AIX NLS subsystem loadable interfaces.

10.6.2.4 Application Programming Interfaces (API)

AIX Input Method defines the API that is used by other programs to support multi-byte input. The Thai input method complies with the AIX Input Method APIs.

The Universal input method treats the Thai input method as another option in the list of input methods.

10.6.3 Vietnamese Language Support

Previous to AIX 4.3, Vietnamese language support was based on a solution from IBM Vietnam. AIX Version 4.3 formally implements support for the Vietnamese language.

The features implemented in AIX Version 4.3 include:

Locale The Vietnamese locale is based on either IBM-1129 or Unicode Version 2. The wide character processing for UTF-8 Vietnamese locales is based on Unicode.

Input method

In Vietnam, it is not possible to assign each character to a key on the keyboard. The input method provides a character composing mechanism for the user. In other words, several key combinations can be used to build or compose a character. The Vietnamese input methods are based on IBM 461 A/B on a 101 key US keyboard where some of the Vietnamese and French characters are mapped to the first and second rows.

cmdiconv

New iconv converters are present for the following Vietnamese Language code sets:

- IBM-1129
- Unicode
- EBCDIC Code (IBM-1130)
- PC Code (IBM-1258, which is the same as Windows Code page)

•TCVN2 (current standard code set used in Vietnam)

charmap New character map for Vietnamese code set definition.

cmdpios AIX Version 4.3 provides a printer filter for Vietnamese that supports downloadable UCS-2 fonts and can print data for both IBM-1129 and UTF-8 locales.

libmeth Locale-specific methods for wide character processing (based on Unicode).

cmdtty Vietnamese csmap.

i18n The Vietnamese language has been added to the X locale database.

xmodmap Localized X keysyms and server keyboard maps are present for Vietnamese.

CTL layout engine
Vietnamese makes use of combining zero width characters that require the use of layout services as defined by the CTL APIs.

10.6.3.1 Systems Management

The Vietnamese locales are configurable through the SMIT panel Manage Language Environment.

The user can install the following items:

- Vietnamese Cultural Conventions (IBM-1129)
- Vietnamese Cultural Conventions (Unicode)

10.6.3.2 Standards Compliance of Vietnamese Language Support

Vietnamese Language Support follows the standards that the AIX NLS Architecture follows, especially those standards listed below. The UTF-8 support for the Vietnamese locale consists of the portion of the Base Multilingual Plane of ISO 10646 that is supported by other locales on AIX. The relevant standards and resources are:

- X/Open Portability Guide, issue 3 and issue 4
- POSIX.2
- X Window System Version 11 Release 6
- IBM 1129 for internal Code Page
- IBM 461 A/B for Keyboard layout
- IBM 1130 for EBCDIC code Convert
- Unicode Version 2.0 for display and fonts

10.6.3.3 Migration

Current users of the country solution for Vietnam are expected to migrate to AIX Version 4.3 with the Vietnamese localization support. The new Vietnamese Language support is binary-compatible with the previous country solution for Vietnam.

10.6.4 Japanese Code Page 943 (AIX 4.3.2)

IBM-943 is a compatible code set for the Japanese Microsoft Windows environment. This code set is known as ’83 ordered Shift JIS.

In Japan, most of operating systems on PC clients support JIS 83 ordered Shift JIS code set (IBM-943) to implement Japanese. AIX has supported JIS 78 ordered Shift JIS code (IBM-932) since AIX Version 3. This was one of the advantages in enabling a UNIX-PC mixed environment. Recently, most Japanese PCs use JIS 83 ordered Shift JIS code set as the default, and this causes a data incompatibility between PC and AIX environments.

AIX 4.3.2 supports the IBM-943 code set that is needed for PC interoperability. The code page 943 supports interoperability with Microsoft Windows clients in Japan and makes AIX Version 4.3.2 the preferred release for Japan.

The difference between IBM-932 in previous AIX versions and IBM-943 is as follows:

- New JIS sequence ('83 ordered).
- NEC selected characters (83) are added.
- NEC's IBM selected characters (374) are added.
- Two new characters are added (defined by JIS90).

Ja_JP is used for a short locale name of the IBM-943 locale. The short locale name of IBM-932 is no longer supported.

If you want to use IBM-932 rather than IBM-943, you can use the IBM-932 by specifying the long locale name Ja_JP.IBM-932.

10.6.4.1 Installation and Packaging

Following list describes the installation and packaging of IBM-943 locale:

- The IBM-943 locale is selectable for installation from SMIT.
- Any locale support that is common to all Japanese locales are packaged and shipped in the existing fileset bos.loc.com.JP.
- The IBM-943 locale is packaged in bos.loc.pc.Ja_JP. This fileset contains both IBM-932 and IBM-943 locales.
- The IBM-943 locale-specific X11 resource is shared with the IBM-932 X11 resource.

10.6.5 Korean TrueType Font (AIX 4.3.2)

TrueType is the scalable font technology built into Windows and Macintosh workstations. AIX 4.3.2 now supports Korean TrueType fonts in the AIX X-environment.

A TrueType rasterizer is available in the AIXwindows environment in AIX 4.3. All scalable font systems need a rasterizer to convert their glyph outlines into bitmaps suitable for direct copying to the screen.

The TrueType rasterizer in AIX 4.3.2 supports TrueType 1.0 Font Files and most of the character sets currently defined in the Unicode 2.0 specification.

There are three kinds of extensions for the font file:

TTF The recommended file extension for TrueType font files.

TTC TrueType Collection file. A scheme where multiple TrueType fonts can be stored in a single file, typically used when only a subset of glyphs changes among different designs. They are used in Japanese fonts, where the Kana glyphs change but the Kanji remain the same.

OTF The recommended file extension for the Type 1 font (not for TrueType).

Changes to existing TTF rasterizer is described as follows:

- Korean support added
- Font types added: Batang, Dotum, and Sammul
- Font sizes supported: 9, 10, 12, 18, 24, 32, 40, 48, and 72
- Application support: X-window application, Netscape, and dtterm

AIX has two locales related with Korean: ko_KR (IBM-eucKR) and KO_KR (Unicode). In AIX 4.3.2, Korean TrueType font rasterizer works only on ko_KR locale based on ksc5601.

The TrueType rasterizer is available in the AIXwindows font library, which is shipped with both the AIXwindows Font Server and X server.

The Korean TrueType font file supplied in AIX 4.3.2 is a TrueType collection file that has two TrueType Korean font sets. One is proportional, and the other is monospaced. The font file name is hmfmm.ttc. This font file is packaged in the X11.fnt.ksc5601.ttf fileset. It is installed in the /usr/lib/X11/fonts/TrueType directory.

10.6.5.1 Standards

The AIXwindows Font server supports the X window System Font Protocol Version 2.0 and X Version 1.1 Release 6.1.

The TrueType rasterizer supports TrueType 1.0 font files.

The name of the AIX windows Font server in AIX 4.3 is changed from fs to xfs. Also, the font server's port number is changed from 7500 to 7100. These changes between X11R5 and X11R6 were made by the X consortium, suppliers of the X Window System technology.

10.6.5.2 Installation and Packaging

The TrueType rasterizer is automatically installed as part of the AIX window X server and font server.

The following is some information about Korean True Type font packaging:

X11.fnt.fontServer

The TrueType rasterizer is shipped in the AIXwindows font server, which can serve fonts to different X servers across the network.

X11.base.rte

This fileset ships the AIXwindows X server. The pre-req.s for this fileset are not be updated since the X server serves mostly local clients. If remote clients require additional fonts, then the X server should be configured to use an AIXwindow font server configured with the proper converters.

X11.samples.fnt.util

This fileset includes the ttfinfo sample utility, which displays the header information of a TrueType font.

10.7 Documentation Search Service: DBCS HTML Search Engine (4.3.2)

The AIX Documentation Search Service is extended to add the capability to search specified Double Byte Character Set (DBCS) codesets in Japanese, Korean, Simplified Chinese, and Traditional Chinese.

The doublebyte languages and code sets supported by Documentation Search Service (docsearch) in AIX 4.3.2 are listed in the Table 57. For a complete list of supported languages, codesets, and locales, see the language support table section in *AIX Version 4.3 General Programming Concepts: Writing and Debugging Programs*, SG23-2533.

Table 57. Additional Double-Byte Support in Docsearch

Language	CCSID	Codeset	Locale
Japanese	932 *	IBM-932 *	Ja_JP *
Korean	970	IBM-eucKR	ko_KR
Simplified Chinese	1383	IBM-eucCN	zh_CN
Traditional Chinese	950	big5	Zh_TW
* Note: The Ja_JP locale uses the 943 codeset and CCSID, however the Documentation Search Service currently supports 932.			

The Document Search Service is browser and Web-server based. It allows you to search registered HTML documents using a search form that appears in the Web browser. When you type words into the search form, the service searches for those words and then presents a search results page containing links that lead to the documents containing the target words.

The browser of a desired language code set is both used for display of text and the form based text entry fields.

The AIX search form allows you to search all documents that are registered on a host. Before any document can be searched using the documentation search service, it must have an index created, and the index must be registered with the search service. Some applications ship prebuilt document indexes inside their install package. When the application is installed, the indexes are automatically registered. The AIX Version 4.3 documentation and the Web-Based System Management application both ship prebuilt indexes for their documents. You can also create indexes for your own HTML documents and register them with the search engine so that they can be searched on line. For further information on

how to create indexes, see AIX Documentation Search Service in *AIX Version 4.3 General Programming Concepts: Writing and Debugging Programs,* SC23-2533.

If more than one language is installed, you can switch to other installed languages. The Documentation Search Service GUI is consistent across languages. Although there may be some locale differences such as the sort technique and display layout.

Functions available for nongraphical displays will depend on function provided by the browser used on the nongraphical display. For example, if an ASCII browser does not support the code sets used, the ASCII user of that browser will not be able to search the translated documents written in that code set.

The function for the double byte languages is implemented by the DBCS IMNSearch search engine. DBCS search duplicates the function provided for single byte languages. The functions of both the single and double byte CGIs in AIX 4.3.2 are the same as that of the AIX 4.3.0 single byte CGI with the following conditions noteworthy:

- Two languages cannot be displayed or searched at the same time. Documentation Search Service GUIs only display a single language at a time. Only the indexes of the current language are visible within the GUI for selection and searching. A selection menu button in the search form allows you to switch between languages.

  For example, if the GUI search from is being displayed in Spanish, then only Spanish books will show in the "select volumes to search" section at the bottom of the form. It is not possible to display indexes from different code sets on the same HTML page and have them display correctly.
- The results page adds a start-over button that allows you to jump back to the home search page from any results page.

Translation of the changes and additions to Documentation Search Service messages, GUI, and help search form are done by program integrated information (PII).

10.7.1 Documentation Libraries

The AIX 4.3.2 documentation library consists of two CDs. The first is the AIX Base Documentation in HTML format and the second is the AIX Extended Documentation in HTML and PDF format. As of this writing, the AIX Extended Documentation is only available in English.

The base documentation library contains most of the AIX user, system administrator, and application programmer guides. This library also contains basic reference documents such as the Commands Reference, Files Reference, and Technical Reference volumes intended for application programmers.

The extended documentation library contains books concerning adapters, books intended for system programmers, and technical specifications describing industry standards.

Most of the documentation in these libraries is in HTML format and must be viewed using an HTML version 3.2-compliant Web browser, such as the Netscape Navigator 4.0 browser. A few documents in these libraries are in PDF format and must be viewed using the Adobe Acrobat Reader, Version 3.0.

Documentation Search Service does not support PDF files. The Netscape Navigator browser and Acrobat Reader Version 3.0.1 are shipped with AIX 4.3.2 Bonus Pack.

The HTML documents can be searched using the Documentation Search Service, bos.docsearch, an optionally installable component of the AIX base operating system. It is highly recommended to install and configure the Documentation Search Services since this service may be used by other applications installed on the system. Chapter 9, “Online Documentation” on page 233 for installation information.

On-line documentation is also available at: *http://www.rs6000.ibm.com/aix/library*

10.7.2 Limitations

East European locales are not supported for searching, however product libraries may be available if they are translated.

For the supported locales, it is not necessary to do any conversions for the components to communicate with each other.

10.7.3 Invoking Documentation Search Service

The global search form can be accessed by:

- Type `docsearch` on the command line
- Click the Documentation Search Service icon in the CDE Desktop Help subpanel.

The Documentation Search Service desktop action and the `docsearch` command will both check for the existence of an environment variable DOC_LANG. This variable is used to define the language in which to display the search form GUI. It also specifies the language in which the search engine will conduct the search. If not defined, the language used will be that of the launching environment, if possible.

Set the DOC_LANG environment variable with the language locale you want to search. The search form will only display indexes of the currently selected language. The hit list will return hits from all books that were selected for search in the search form. The `chdoclang` command will set the DOC_LANG environment variable for you. See the AIX 4.3.2 Release Notes for more information on the `chdoclang` command.

The language selection menu in the search form allows you to select the language of the GUI. When a language is selected, the GUI is displayed in that language and displays only the indexes for that language. The selection menu is intelligent in that its contents are dynamically generated and it will display only those languages that are currently available on the system being used.

10.7.3.1 Japanese Documentation Search

A Japanese search can be done in a Japanese CDE environment. If you are in another language’s CDE environment, you need to set LANG to Ja_JP and map the keyboard using the following command:

```
#xmodmap /usr/lpp/X11/defaults/xmodmap/Ja_JP/keyboard
```

When you enter:

#docsearch

you get the screen such as shown in Figure 58 on page 287.

Figure 58. Japanese Search Form

After remapping the keyboard, using a US keyboard, you can input the words you want to search into the search field. Note that the entire product library is not installed for this example. Figure 59 shows an example of searching for "ls".

Figure 59. Searching Japanese Documentation

The resulting page is shown in Figure 59 on page 288.

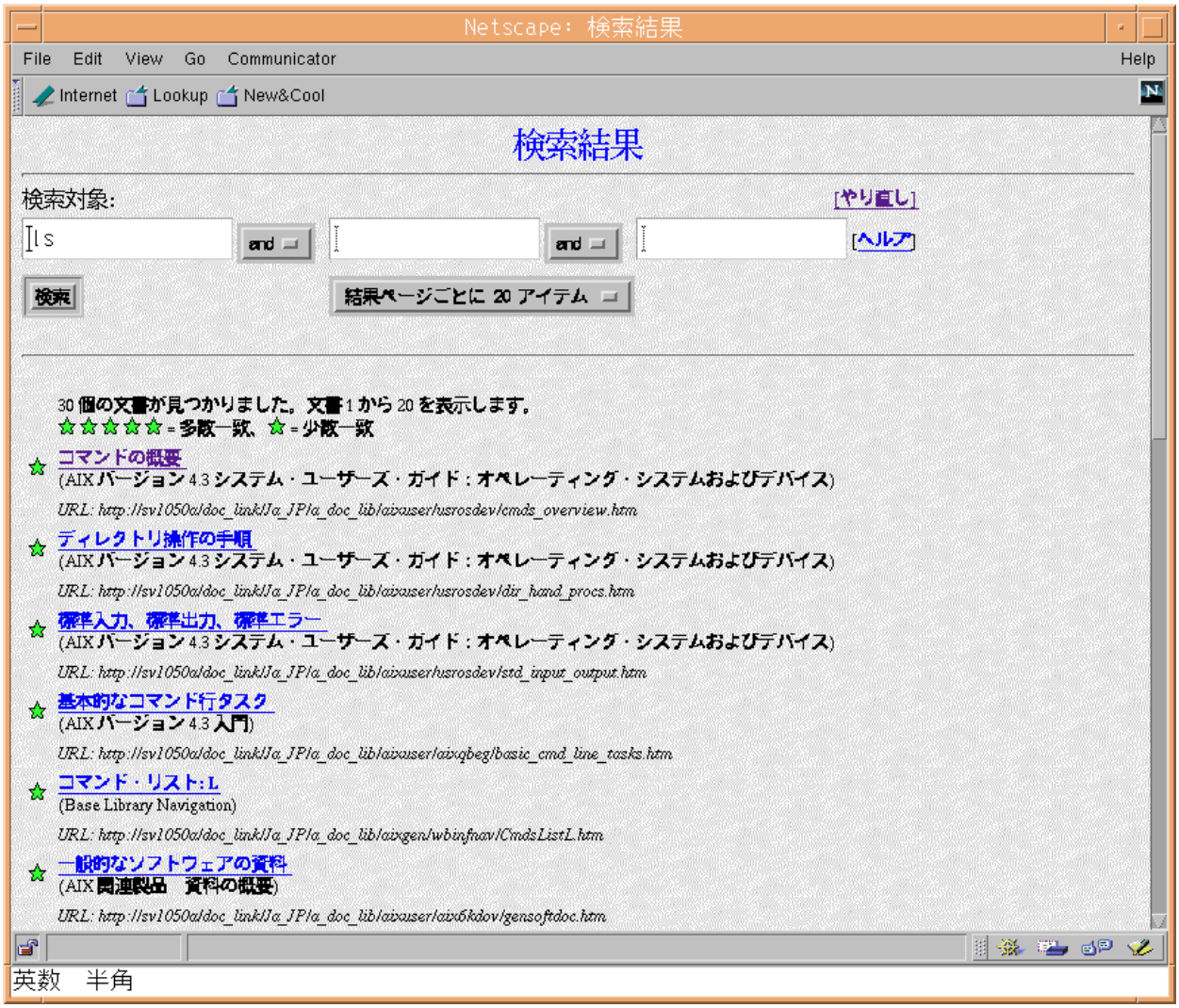


Figure 60. A Japanese Search Result

The results page lists documents that contain the term you searched for. To see one of the documents containing the term you searched for, click on the title of the document that you want to see. It is a hyperlink to that document. For example, Figure 61 on page 290 displays a page containing the term searched for in this example.

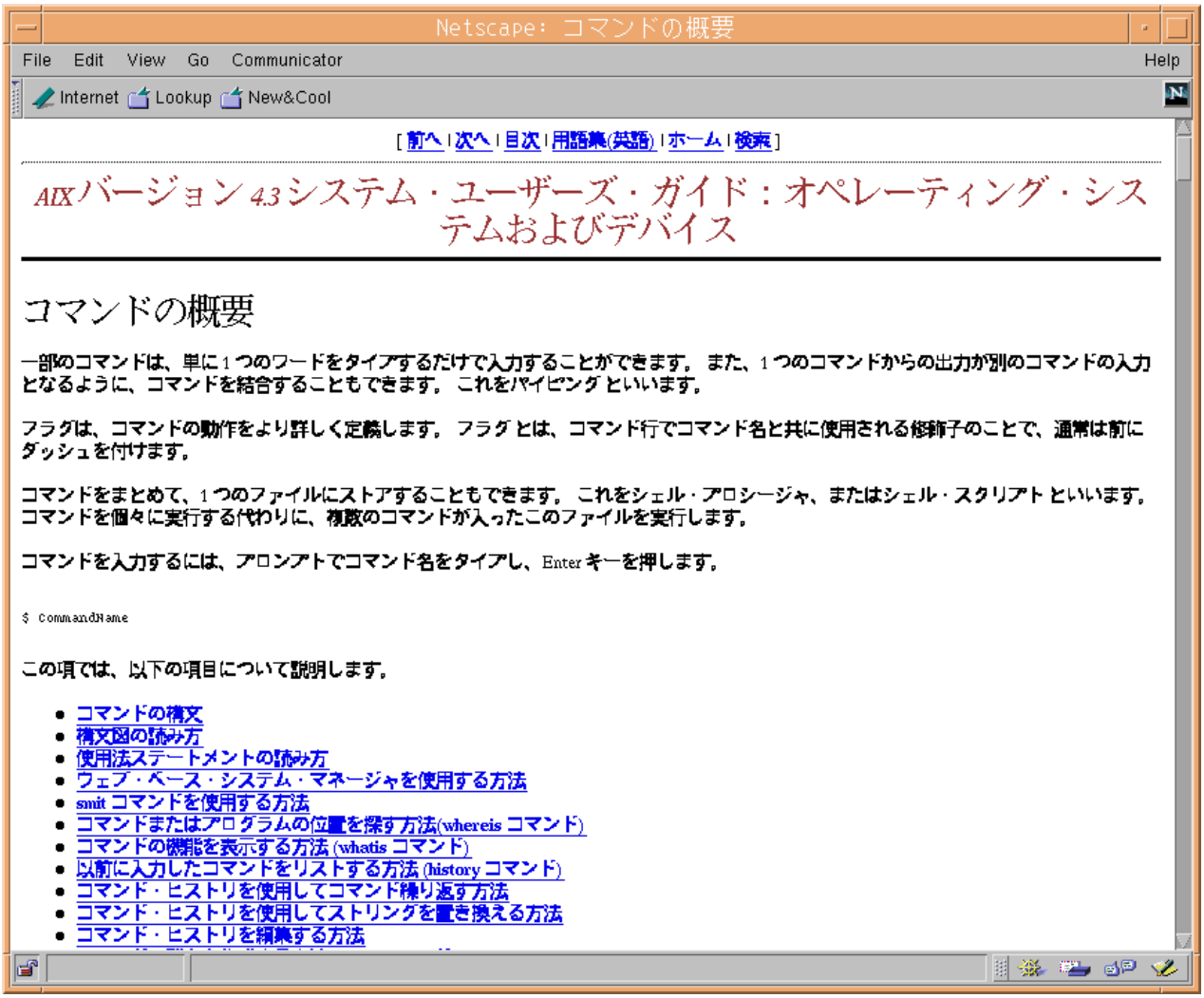


Figure 61. A Japanese Book

10.7.3.2 Simplified Chinese Search

A Chinese document search can be done in a Chinese CDE environment. If you are in other language's CDE environment, you need to set the LANG to zh_CN and map the keyboard using the following command:

```
#xmodmap /usr/lpp/X11/defaults/xmodmap/zh_CN/keyboard
```

When you enter:

```
#docsearch
```

you see the screen similar to Figure 62:

Figure 62. Chinese Search Form

After remapping the keyboard, using a US keyboard, you can input the Chinese words you want to search into the search field, as shown in Figure 63. Note that the entire product library is not installed for this example.

Figure 63. Input Chinese Character

The searching result is similar to Figure 64 on page 293:

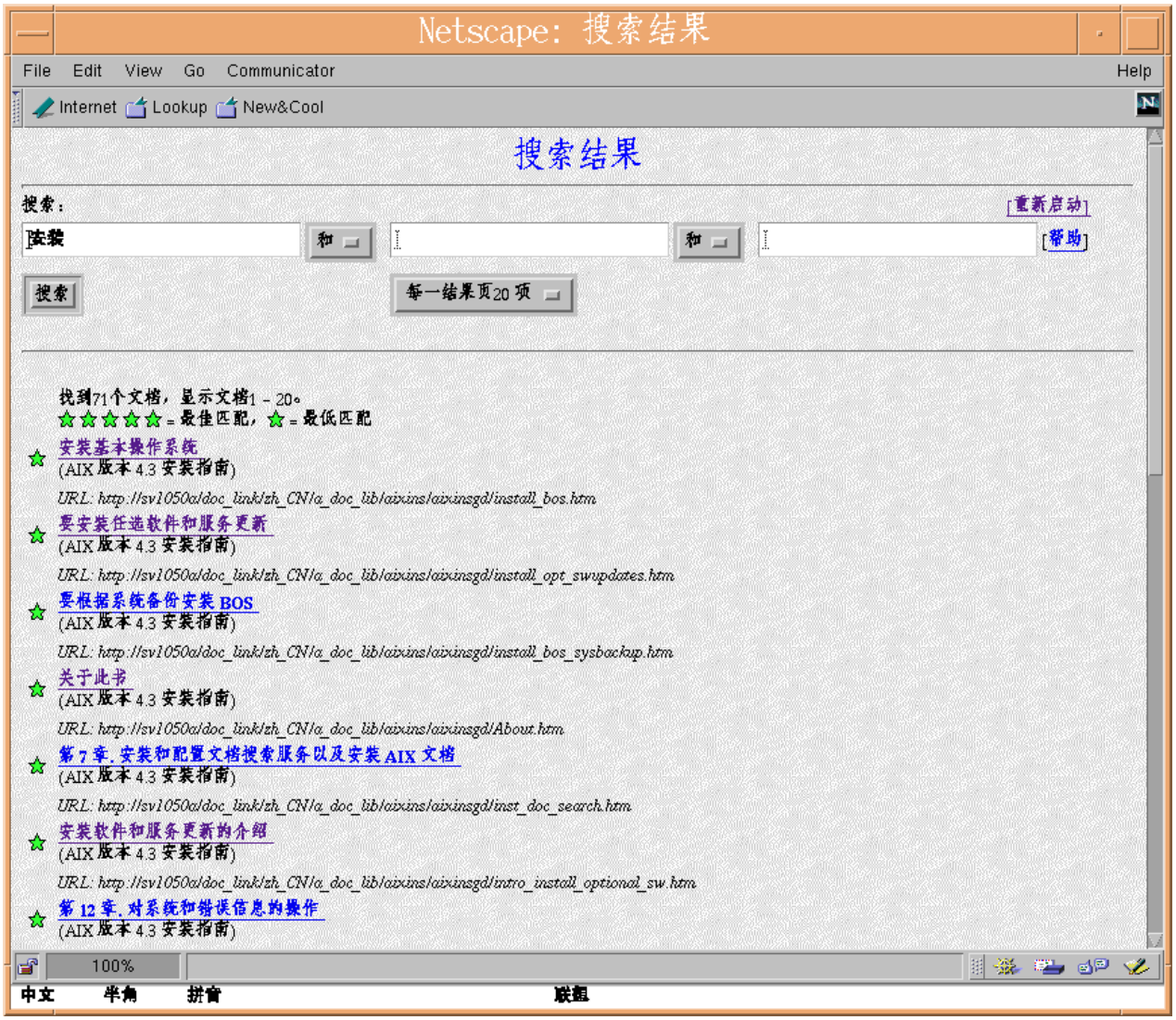


Figure 64. Chinese Searching Result

The results page lists documents that contain the term you searched for. To see one of the documents containing the term you searched for, click on the title of the document that you want to see. It is a hyperlink to that document. For example, Figure 65 on page 294 displays a page containing the term searched for in this example.

Figure 65. A Chinese Book for Installation

10.7.4 Binary Compatibility

Single byte indexes created using AIX 4.3 will work correctly with the AIX 4.3.2 single byte Documentation Search Service. The Documentation Search Service should be maintained at the latest level.

Chapter 11. AIX Stand-Alone LDAP Directory Product

The AIX Stand-alone Lightweight Directory Access Protocol (LDAP) product provides client access to directory data on a server using standard Internet protocols (LDAP and HTTP). The following discussions are representative of AIX Version 4.3.0.

11.1 Typical Configurations

Figure 66 shows a typical client/server network with examples of many of the capabilities provided. A description of each component follows:

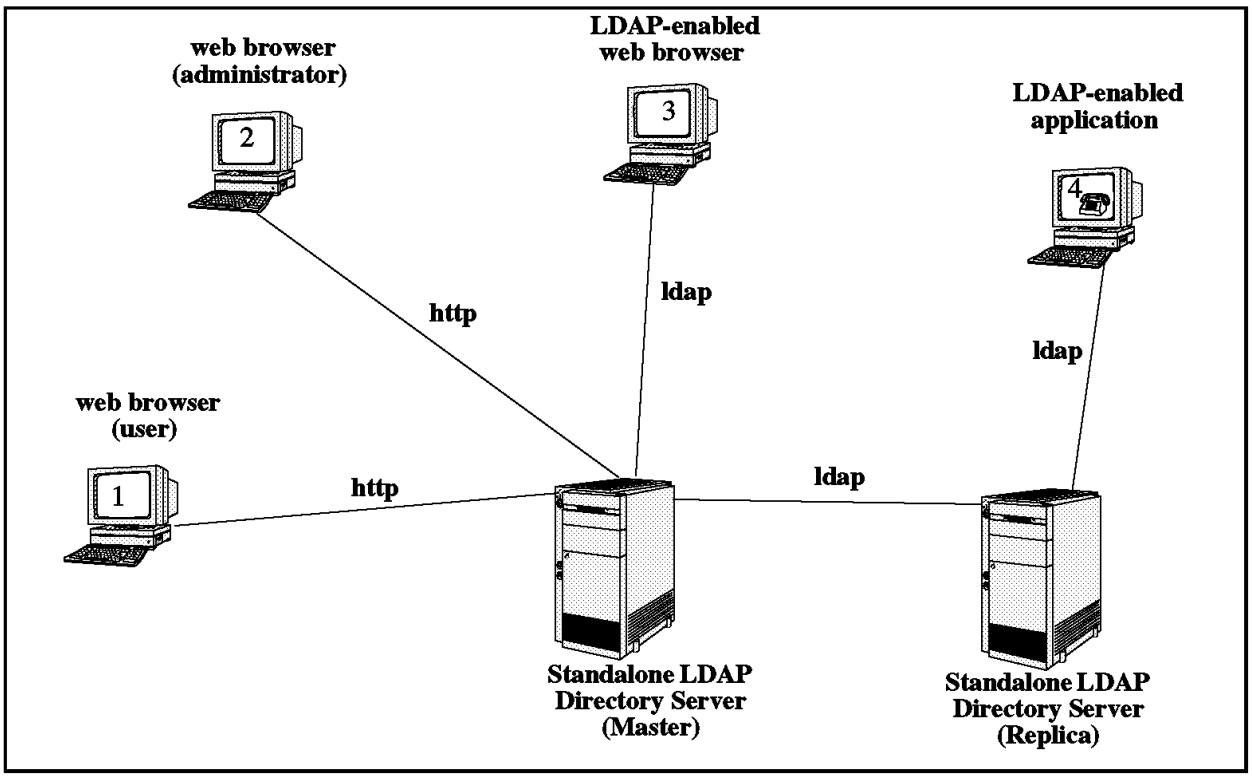


Figure 66. Typical AIX Stand-Alone LDAP Client/Server Configuration

The components of the previous figure are as follows:

Client 1

This represents an end-user of the directory working with a generic Web browser that has no built-in LDAP support. The user may connect to the server using standard HTTP and supply the URL of the HTTP gateway provided with the product. The user is then presented with a form to fill in to specify the desired search parameters. The HTTP gateway performs the search on behalf of the client and returns the matching entries from the directory to the Web browser.

Client 2

At this client, a directory administrator uses a Web browser to monitor or configure the directory. The Web browser requires no specific LDAP

capabilities. In this case, the administrator connects through an install-time-selected port to a Web server located on the system containing the directory server. The HTML panels of the administration interface are presented to the administrator through the Web browser and guides them through viewing or setting configuration options for the directory.

Client 3

This represents an end-user who has a Web browser that is LDAP-enabled. The protocol flow from this client to the server is LDAP; therefore, it requires no protocol conversion on the LDAP server system.

Client 4

This client is running an application that is LDAP-enabled. The application may have been built using the LDAP Client Toolkit provided with this product, or it may have been built using an LDAP client library from another source. Note that the client is shown connecting to the Replica server but is able to search the directory by connecting to either the Master or Replica.

Replica directory server

This server is shown with the AIX Stand-alone LDAP Directory product installed. However, because the replication is achieved using a standard LDAP connection, the replica server could be any LDAP server that supports Version 2 of the LDAP. Replicas are read-only. A given master directory server may have multiple replicas configured.

Master directory server

This system runs the server software from the AIX Stand-alone LDAP Directory product.

11.2 LDAP Protocol Support

AIX Stand-alone LDAP supports Version 3 of the LDAP protocol. There is currently no LDAP Version 3 RFC that is approved. This product has been developed using Internet Draft "Lightweight Directory Access Protocol (v3) <draft-ietf-asid-ldapv3-protocol-04.txt>", that replaces LDAP (v2) RFC 1777.

Both V2 and V3 LDAP clients and servers are supported. The following combinations of clients and servers have been tested:

- AIX Stand-alone LDAP version 3 client with Netscape version 2 server
- A University of Michigan version 2 client with AIX Stand-alone LDAP server
- Netscape Version 2 client with AIX Stand-alone LDAP server

11.3 LDAP Client Toolkit

The LDAP client is represented as a toolkit that provides two types of LDAP interface. The first is a set of C APIs with associated header files and shared libraries. The second is a set of Java classes providing functionality equivalent to the C interface. The toolkit includes `man` pages describing the LDAP programming interface.

The LDAP client toolkit includes the following components:

- LDAP APIs (for Version 3, based on the Internet Draft "The LDAP Application Program Interface", <draft-howes-ldap-api-00.txt>)
- LDAP command line utilities
  - `ldapsearch`
  - `ldapadd`
  - `ldapmodify`
  - `ldapdelete`
  - `ldapmodrdn`
- C header files
- LDAP client Java classes
- Sample directory data
- Sample applications using LDAP

The application is linked with the shared library that exports the APIs and contains the routines behind the APIs that handle the client-side protocol.

11.4 Stand-Alone LDAP Directory Server

The key distinguishing feature between the AIX Stand-alone LDAP server implementation and other LDAP server implementations is the use of DB2 as the back-end data store. See Figure 67 for the major components included in the server package. The numbers along the top of the diagram refer back to the client types represented in Figure 66 as examples of the sort of clients that generate HTTP or LDAP flows to the server.

The features included in the AIX Stand-alone LDAP product are:

- DB2 back-end
- ODBC Driver Manager and DB2 driver
- RDB Glue
- SLAPD
- Server replication
- Administration utilities
- Administration GUI
- HTTP gateway

Note: The references to Oracle and Oracle driver in Figure 67 are merely an example of future possibilities. DB2 is the only supported database in this release.

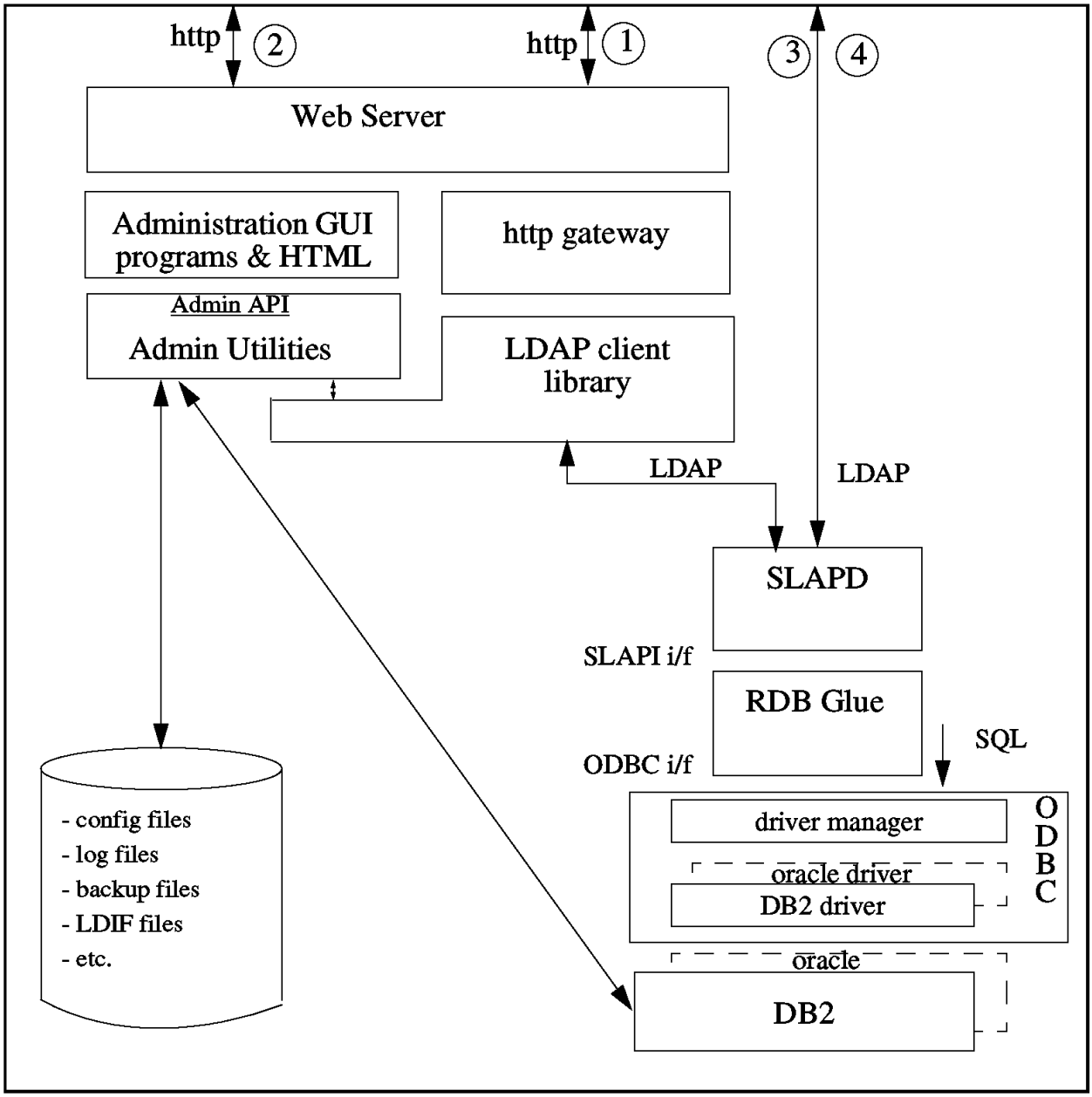


Figure 67. Stand-Alone LDAP Directory Server - Details

11.4.1 DB2 Back End

DB2 provides a robust, scalable, industry-tested basis for storage of the directory data. It includes support of Binary Large Objects (BLOBs) that facilitates use of the directory as an efficient object store. The Open DataBase Connectivity (ODBC) interface is used to connect DB2 as the directory back end. Using ODBC as the interface allows for the future inclusion of other relational databases as back-ends.

11.4.2 ODBC

This includes ODBC Driver Manager and the DB2 (ODBC) Driver. The ODBC Driver Manager provides the ODBC API to the LDAP directory server. The DB2 driver plugs into the ODBC framework and connects with the DB2 interfaces. Both of these components of ODBC are provided with the single-user version of DB2 that is shipped with the AIX Stand-alone LDAP Directory product.

11.4.3 RDB Glue

The Relational DataBase (RDB) Glue code ties together two architected interfaces to provide the data store for the directory. The SLAPD component handles incoming LDAP requests and generates calls to a set of APIs defined as the SLAPI interface. The RDB Glue provides a matching set of routines that plug into this API, take the previously mentioned API calls and generate SQL statements in the form required by the ODBC interface to read or write information to DB2.

11.4.4 SLAPD

SLAPD is the portion of the directory server that understands LDAP. It is a multithreaded daemon that receives client requests, works with the DB2 back-end to process them, and returns the results.

11.4.5 Server Replication

SLAPD process threads also monitor the replication log file and pass the corresponding update requests on to the replica server(s).

No shutdown of LDAP server is necessary to copy directory data to initialize a replica server. The process of setting up a replica server involves the following steps:

- Put the master directory into read-only mode.
- Create a backup of the directory contents.
- Use the backup file to populate the replica directory.
- Update the master directory configuration with all the information about the replica server.
- Dynamically reconfigure the master directory server (SLAPD) to pick up the new configuration.
- Take the master out of read-only mode (back to read-write).

11.4.6 HTTP Access to Directory

An HTTP gateway is provided to allow web browsers that are not LDAP-enabled to do searches on the directory. The gateway is a cgi-bin program that presents a form to the user, through the browser, to gather the parameters for the search (such as a search base, scope, search filter, and so on). Once the search information has been passed to the gateway program, it acts as an LDAP client, generating the requests to do the search, then receiving the results and passing them back to the browser for display to the end-user.

11.5 Security

The AIX Stand-alone LDAP client and server implementation supports SSL (Version 2.0 or higher), an emerging standard for World Wide Web security. SSL

provides encryption of data and transport of X.509v3 public-key certificates and revocation lists. The server may be configured to run with or without the SSL support. When the server is configured to support SSL (accepting connections over a secure port - defaults to 636), it still accepts connections from clients that choose not to use SSL (these clients still specify the standard unsecure port - defaults to 389). The LDAP either flows directly over TCP/IP (using the standard sockets interfaces) or over the SSL.

11.5.1 Authentication

The following authentication options are supported:

- No authentication
- Simple authentication (password)
- X.509v3 public-key certificate at the SSL

Note: Kerberos authentication is not supported.

11.6 Installation

The Stand-alone LDAP Directory is a component of the Base Operating System for AIX 4.3; therefore, installation is done using standard installp facilities. In this release, there are no other BOS components that have dependencies on LDAP so users have the option of installing LDAP or not, depending on their requirements.

11.6.1 Software Prerequisites

The web-based interface for administration of the AIX Stand-alone LDAP Directory server requires a web server to be present, located with the directory server. Only the Netscape Fasttrack Web server has been tested, but any other web server that supports HTML 3.0/3.2 should work.

DB2 Single-User (Version 3) is automatically shipped with the directory server. DB2 is considered an install prereq. on the server system. If a supported version of DB2 has been previously installed, the prereq. requirement will be satisfied. Otherwise, the Single-User version is installed with the directory server (as a separate instance of DB2). The supported versions of DB2 are:

- DB2 Version 2.1.1 and above when packaged with any of the following products:
  - DB2 Single-User
  - DB2 Common Server
  - DB2 Parallel Edition
  - DB2 Universal Database
  - IBM Database Server

11.7 Administrative Interface

Most of the administration can be performed through a graphical user interface accessed through a web browser. The features of the interface and additional command line utilities are described below.

11.7.1 Web-Based Graphical User Interface

A Web-based graphical user interface has been provided to ease the task of administration of the AIX Stand-alone LDAP Directory server. An administrator is able to use a Web browser to do initial set-up of the directory, change configuration options, and manage the day-to-day operation of the server. HTML panels are presented through the browser to guide the administrator in viewing or changing the following:

- General server settings
- General database/back end settings
- Performance tuning options
- Directory server activity/performance data
- Directory replication configuration

and management of the following:

- Start up and shutdown of the directory server
- Access control lists
- Group membership
- Security measures (encryption options, server certificate management)
- Database creation, backup, and restore

11.7.2 Command Line Utilities

There are two command line utilities provided:

- `LDIF2DB`
- `DB2LDIF`

The `LDIF2DB` command line utility creates a directory database in DB2 based on the directory entries specified in LDAP Directory Interchange Format (LDIF) format in an input file.

The `DB2LDIF` command line utility creates a standard LDIF format file containing all the data from the specified DB2 directory database.

A set of command line utilities to perform LDAP operations has been included as a sample. These utilities are:

- ldapsearch
- ldapadd
- ldapmodify
- ldapdelete
- ldapmoddn
- ldapcompare

11.7.3 Other Administrative Procedures

Definition of the Directory Schema (object classes, attribute types, and directory structure) is managed through ASCII configuration files. A default schema definition is shipped in the configuration files with the product. To add to the defaults, the administrator must edit these configuration files. In this initial release, the administrator is allowed to add object class definitions, attribute type definitions, or directory structure rules.

Changes to schema definitions previously in effect for a database are not allowed.

11.8 LDAP-Related RFCs and Internet Drafts Implemented

The following lists contain Internet drafts and RFCs for LDAP and X.500 implemented in the AIX Stand alone LDAP Directory product.

11.8.1 Internet Drafts

Internet drafts may be viewed at: *http://www.internic.net/internet-drafts*

- Protocol Definition (March 25, 1997)
  - <draft-ietf-asid-ldapv3-protocol-04.txt>
- Standard and Pilot Attribute Definitions (March 1997)
  - <draft-ietf-asid-ldapv3-attributes-04.txt>
- A String Representation of LDAP Search Filters (March 1997)
  - <draft-ietf-asid-ldapv3-filter-00.txt> (obsolete)
  - Replaced by <draft-ietf-asid-ldapv3-filter-02.txt> (May 1997)
- Extensions for Dynamic Directory Services (March 25, 1996)
  - <draft-ietf-asid-ldapv3ext-03.txt> (obsolete)
  - Replaced by <draft-ietf-asid-ldapv3ext-04.txt> (May 1997)
- A UTF-8 String Representation of Distinguished Names (March 1997)
  - <draft-ietf-asid-ldapv3-dn-02.txt> (obsolete)
  - Replaced by <draft-ietf-asid-ldapv3-dn-03.txt> (April 1997)
- The LDAP Application Program Interface (October 1996)
  - <draft-howes-ldap-api-00.txt>
- Use of Language Codes in LDAP V3 (March 1997)
  - <draft-ietf-asid-ldapv3-lang-01.txt>
- Definition of an Object Class to Hold LDAP Change (March 25 1997)
  - <draft-ietf-asid-changelog-00.txt>
- LDAP Multi-master Replication Protocol (March 20, 1997)
  - <draft-ietf-asid-ldap-mult-mast-rep-00.txt>
- The LDAP Data Interchange Format(LDIF) (Nov 25, 1996)(March 24 1997)
  - <draft-ietf-asid-ldif-00.txt>

11.8.2 LDAP-Related RFCs

For more information on LDAP and its associated components please refer to the RFCs shown in Table 58.

Table 58. LDAP-Related RFCs

RFC Number	RFC Title
1558	A String Representation of LDAP Search Filters
1738	Uniform Resource Locators

RFC Number	RFC Title
1777	Lightweight Directory Access Protocol
1778	The String Representation of Standard Attribute Syntaxes
1779	A String Representation of Distinguished Names
1798	Connectionless LDAP
1823	The LDAP Application Program Interface
1959	An LDAP URL Format
1960	String format of LDAP search filter

11.8.3 X.500-Related RFCs

For information on X.500, refer to the RFCs shown in Table 59. They may be useful as a source of background information for the LDAP RFCs.

Table 59. X.500-Related RFCs

RFC Number	RFC Title
1274	The COSINE and Internet X.500 Schema
1275	Replication Requirements to Provide an Internet Directory Using X.500
1276	Replication and Distributed Operations Extensions to Provide an Internet Directory Using X.500
1279	X.500 and Domains
1308	Executive Introduction to Directory Services Using the X.500 Protocol
1309	Technical Overview of Directory Services Using the X.500 Protocol
1484	Using the OSI Directory to Achieve User Friendly Naming
1485	A String Representation of Distinguished Names
1487	X.500 Lightweight Directory Access Protocol
1488	The X.500 String Representation of Standard Attribute Syntaxes
1491	A Survey of Advanced Uses of X.500
1558	A String Representation of LDAP Search Filters
1564	DSA Metrics
1608	Representing IP Information in the X.500 Directory
1609	Charting Networks in the X.500 Directory
1617	Naming and Structuring Guidelines for X.500 Directory Pilots
1684	Introduction to White Pages Services based on X.500

Appendix A. Special Notices

This publication is intended to help AIX system administrators, developers, and support professionals understand the key technical differences between AIX Version 4.3 and previous releases. It takes AIX Version 4.2 as the basis for these differences. The information in this publication is not intended as the specification of any programming interfaces that are provided by AIX Version 4.3. See the PUBLICATIONS section of the IBM Programming Announcement for AIX Version 4.3 for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM ("vendor") products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other operating

environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

The following document contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples contain the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

AIX®
AS/400®
BookManager®
eNetwork
IBM ®
IBMLink
Magstar®
Micro Channel®
Network Station
OS/2®
POWERparallel®
PowerPC®
PowerPC Architecture
PowerPC 604
POWER2 Architecture
POWER3 Architecture
RETAIN®
RISC System/6000®
RS/6000®
Service Director®
SP
System/390
TURBOWAYS®
VisualAge®

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc.

Java and HotJava are trademarks of Sun Microsystems, Incorporated.

Microsoft, Windows, Windows NT, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

Pentium, MMX, ProShare, LANDesk, and ActionMedia are trademarks or registered trademarks of Intel Corporation in the U.S. and other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names may be trademarks or service marks of others.

Appendix B. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

B.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see, “How to Get ITSO Redbooks” on page 311.

- *A Technical Introduction to PCI-Based RS/6000 Servers*, SG24-4690
- *Managing AIX V4 on PCI-Based RISC System/6000 Workstations*, SG24-2581
- *TCP/IP Tutorial and Technical Overview*, GG24-3376
- *Learning Practical TCP/IP for AIX V3.2/4.1 Users: Hints and Tips for Debugging and Tuning*, SG24-4381
- *Managing One or More AIX Systems - Overview*, GG24-4160
- *AIX/6000 X.25 LPP Cookbook*, SG24-4475
- *AIX Version 4.1 Software Problem Debugging and Reporting for the RISC System/6000*, GG24-2513
- *AIX Version 4.2 Differences Guide*, SG24-4807
- *A Holistic Approach to AIX V4.1 Migration, Planning Guide*, SG24-4651
- *RS/6000 Performance Tools In Focus*, SG24-4989
- *RS/6000 Technical and Scientific Computing: POWER3 Introduction and Tuning Guide*, SG24-5155

B.2 Redbooks on CD-ROMs

Redbooks are also available on CD-ROMs. **Order a subscription** and receive updates 2-4 times a year at significant savings.

CD-ROM Title	Subscription Number	Collection Kit Number
System/390 Redbooks Collection	SBOF-7201	SK2T-2177
Networking and Systems Management Redbooks Collection	SBOF-7370	SK2T-6022
Transaction Processing and Data Management Redbook	SBOF-7240	SK2T-8038
Lotus Redbooks Collection	SBOF-6899	SK2T-8039
Tivoli Redbooks Collection	SBOF-6898	SK2T-8044
AS/400 Redbooks Collection	SBOF-7270	SK2T-2849
RS/6000 Redbooks Collection (HTML, BkMgr)	SBOF-7230	SK2T-8040
RS/6000 Redbooks Collection (PostScript)	SBOF-7205	SK2T-8041
RS/6000 Redbooks Collection (PDF Format)	SBOF-8700	SK2T-8043
Application Development Redbooks Collection	SBOF-7290	SK2T-8037

B.3 Other Publications

These publications are also relevant as further information sources:

- *AIX Version 4 Getting Started*, SC23-2527
- *AIX Version 4.2 Installation Guide*, SC23-1924
- *AIX Version 4.2 Quick Installation Guide*, SC23-1925
- *AIX Version 4.2 Network Installation Management Guide and Reference*, SC23-1926
- *AIX Version 4 System Management Guide: Operating System and Devices*, SC23-2525
- *AIX Version 4 System Management Guide: Communications and Networks*, SC23-2526
- *AIX Version 4 System User's Guide: Operating System and Devices*, SC23-2544
- *AIX Version 4 System User's Guide: Communications and Networks*, SC23-2545
- *AIX Version 4 Problem Solving Guide and Reference*, SC23-2606
- *AIX Version 4 Messages Guide and Reference*, SC23-2641
- *AIX Versions 3.2 and 4 Performance Tuning Guide*, SC23-2365
- *AIX Version 4 Files Reference*, SC23-2512
- *AIX Version 4 Commands Reference*, SBOF-1851 (Contains the following publications that may also be ordered separately.)
  - *AIX Version 4 Commands Reference, Volume 1*, SC23-2537
  - *AIX Version 4 Commands Reference, Volume 2*, SC23-2538
  - *AIX Version 4 Commands Reference, Volume 3*, SC23-2539
  - *AIX Version 4 Commands Reference, Volume 4*, SC23-2540
  - *AIX Version 4 Commands Reference, Volume 5*, SC23-2639
  - *AIX Version 4 Commands Reference, Volume 6*, SC23-2640
  - *Go Solo 2*, SR28-5705
- *AIX Version 4 General Programming Concepts: Writing and Debugging Programs*, SC23-2533
- *AIX Version 4 Communications Programming Concepts*, SC23-2610
- *AIX Version 4 Technical Reference*, SBOF-1852 (Contains the following publications that may also be ordered separately.)
  - *AIX Version 4 Technical Reference, Volume 1: Base Operating System and Extensions*, SC23-2614
  - *AIX Version 4 Technical Reference, Volume 2: Base Operating System and Extensions*, SC23-2615
  - *AIX Version 4 Technical Reference, Volume 3: Communications*, SC23-2616
  - *AIX Version 4 Technical Reference, Volume 4: Communications*, SC23-2617
  - *AIX Version 4 Technical Reference, Volume 5: Kernel and Subsystems*, SC23-2618

  - *AIX Version 4 Technical Reference, Volume 6: Kernel and Subsystems*, SC23-2619
  - *AIX Version 4 Technical Reference, Volume 7: AIXwindows*, SC23-2620
  - *AIX Version 4 Technical Reference, Volume 8: Enhanced Xwindows*, SC23-2621
  - *AIX Version 4 Technical Reference, Volume 9: Enhanced Xwindows*, SC23-2622
  - *AIX Version 4 Technical Reference, Volume 10: Enhanced Xwindows*, SC23-2623
  - *AIX Version 4 Technical Reference, Volume 11: Master Index*, SC23-2624
- *AIX Version 4 Quick Reference*, SC23-2529
- *AIX Version 4 iFOR/LS Tips and Techniques*, SC23-2666
- *AIX Version 4 AIXwindows Programming Guide*, SC23-2632
- *AIX Version 4 Enhanced Xwindows Programming Guide*, SC23-2636
- *Common Desktop Environment 1.0*, SBOF-1869 (Contains the following Publications that may also be ordered separately.)
  - *Common Desktop Environment 1.0: Application Builder User's Guide*, SC23-2785
  - *Common Desktop Environment 1.0: Desktop KornShell User's Guide*, SC23-2786
  - *Common Desktop Environment 1.0: Help System Author's and Programmer's Guide*, SC23-2787
  - *Common Desktop Environment 1.0: Internationalization Programmer's Guide*, SC23-2788
  - *Common Desktop Environment 1.0: Programmer's Overview*, SC23-2789
  - *Common Desktop Environment 1.0: Programmer's Guide*, SC23-2790
  - *Common Desktop Environment 1.0: Style Guide and Certification Checklist*, SC23-2791
  - *Common Desktop Environment 1.0: ToolTalk Messaging Overview*, SC23-2792
- *AIX Version 4.3 General Programming Concepts: Writing and Debugging Programs,* SC23-2533

B.4 Internet Sites

The following are valuable resources located on the Internet.

- `http://www.gigabit-ethernet.org`
- `http://grouper.ieee.org/groups/802/`
- `http://www.developer.ibm.com/devcon/`
- `http://www.rs6000.ibm.com/software/Apps/LPPmap.html`
- `http://www.ietf.org/html.charters/ipngwg-charter.html`
- `http://www.rs6000.ibm.com/doc_link/en_US/a_doc_lib/aixgen/`

- http://www.lexmark.com
- http://www.ietf.org/html.charters/ipngwg-charter.html
- http://europa.eu.int/euro

How to Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, CD-ROMs, workshops, and residencies. A form for ordering books and CD-ROMs is also provided.

This information was current at the time of publication, but is continually subject to change. The latest information may be found at `http://www.redbooks.ibm.com/`.

How IBM Employees Can Get ITSO Redbooks

Employees may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **Redbooks Web Site on the World Wide Web**

  `http://w3.itso.ibm.com/`

- **PUBORDER** – to order hardcopies in the United States
- **Tools Disks**

  To get LIST3820s of redbooks, type one of the following commands:

  ```
  TOOLCAT REDPRINT
  TOOLS SENDTO EHONE4 TOOLS2 REDPRINT GET SG24xxxx PACKAGE
  TOOLS SENDTO CANVM2 TOOLS REDPRINT GET SG24xxxx PACKAGE (Canadian users only)
  ```

  To get BookManager BOOKs of redbooks, type the following command:

  ```
  TOOLCAT REDBOOKS
  ```

  To get lists of redbooks, type the following command:

  ```
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET ITSOCAT TXT
  ```

  To register for information on workshops, residencies, and redbooks, type the following command:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1998
  ```

- **REDBOOKS Category on INEWS**
- **Online** – send orders to: USIB6FPL at IBMMAIL or DKIBMBSH at IBMMAIL

Redpieces

For information so current it is still in the process of being written, look at "Redpieces" on the Redbooks Web Site (`http://www.redbooks.ibm.com/redpieces.html`). Redpieces are redbooks in progress; not all redbooks become redpieces, and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

How Customers Can Get ITSO Redbooks

Customers may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **Online Orders** – send orders to:

	IBMMAIL	**Internet**
In United States	usib6fpl at ibmmail	usib6fpl@ibmmail.com
In Canada	caibmbkz at ibmmail	lmannix@vnet.ibm.com
Outside North America	dkibmbsh at ibmmail	bookshop@dk.ibm.com

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	(long distance charges apply)
(+45) 4810-1320 - Danish	(+45) 4810-1020 - German
(+45) 4810-1420 - Dutch	(+45) 4810-1620 - Italian
(+45) 4810-1540 - English	(+45) 4810-1270 - Norwegian
(+45) 4810-1670 - Finnish	(+45) 4810-1120 - Spanish
(+45) 4810-1220 - French	(+45) 4810-1170 - Swedish

- **Mail Orders** – send orders to:

IBM Publications Publications Customer Support P.O. Box 29570 Raleigh, NC 27626-0570 USA	IBM Publications 144-4th Avenue, S.W. Calgary, Alberta T2P 3N5 Canada	IBM Direct Services Sortemosevej 21 DK-3450 Allerød Denmark

- **Fax** – send orders to:

United States (toll free)	1-800-445-9269
Canada	1-800-267-4455
Outside North America	(+45) 48 14 2207 (long distance charge)

- **1-800-IBM-4FAX (United States)** or **(+1) 408 256 5422 (Outside USA)** – ask for:

  Index # 4421 Abstracts of new redbooks
  Index # 4422 IBM redbooks
  Index # 4420 Redbooks for last six months

- **On the World Wide Web**

Redbooks Web Site	http://www.redbooks.ibm.com
IBM Direct Publications Catalog	http://www.elink.ibmlink.ibm.com/pbl/pbl

Redpieces

For information so current it is still in the process of being written, look at "Redpieces" on the Redbooks Web Site (`http://www.redbooks.ibm.com/redpieces.html`). Redpieces are redbooks in progress; not all redbooks become redpieces, and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

IBM Redbook Order Form

Please send me the following:

Title	Order Number	Quantity

First name | Last name

Company

Address

City | Postal code | Country

Telephone number | Telefax number | VAT number

☐ Invoice to customer number

☐ Credit card number

Credit card expiration date | Card issued to | Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

List of Abbreviations

ABI	Application Binary Interface
AH	Authentication Header
ANSI	American National Standards Institute
API	Application Programming Interface
ARP	Address Resolution Protocol
ASR	Address Space Register
AUI	Attached Unit Interface
BLOB	Binary Large Object
CDE	Common Desktop Environment
CDLI	Common Data Link Interface
CEC	Central Electronics Complex
CGE	Common Graphics Environment
CHRP	Common Hardware Reference Platform
CISPR	International Special Committee on Radio Interference
CLVM	Concurrent LVM
CMOS	Complimentary Metal-Oxide Semiconductor
DAD	Duplicate Address Detection
DASD	Direct Access Storage Device
DBE	Double Buffer Extension
DCE	Distributed Computing Environment
DES	Data Encryption Standard
DMA	Direct Memory Access
DSMIT	Distributed SMIT
DTE	Data Terminating Equipment
EA	Effective Address
ECC	Error Checking and Correcting
EIA	Electronic Industries Association
EMU	European Monetary Union
EOF	End of File
ESID	Effective Segment ID
ESP	Encapsulating Security Payload
FCC	Federal Communication Commission
FCAL	Fibre Channel Arbitrated Loop
FDDI	Fiber Distributed Data Interface
FIFO	First-In First-Out
FLASH EPROM	Flash Erasable Programmable Read-Only Memory
GAI	Graphic Adapter Interface
GPR	General Purpose Register
HCON	IBM AIX host connection program/6000
HFT	High Function Terminal
I/O	Input/Output
ICCCM	Inter-Client Communications Conventions Manual
ICE	Inter-Client Exchange
ICElib	Inter-Client Exchange library
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IHV	Independent Hardware Vendor
IJG	Independent JPEG Group
ILS	International Language Support
IM	Input Method
IPL	Initial Program Load
IPSec	IP Security
ISA	Industry Standard Architecture
ISAKMP/Oakley	Internet Security Association Management Protocol
ISO	International Organization for Standardization
ISV	Independent Software Vendor
ITSO	International Technical Support Organization
JFS	Journaled File System
LDAP	Lightweight Directory Access Protocol

LDIF LDAP Directory Interchange Format

LFT Low Function Terminal

LID Load ID

LP Logical Partition

LPI Lines Per Inch

LPP Licensed Program Products

LPR/LPD Line Printer/Line Printer Daemon

LP64 Long-Pointer 64

LTG Logical Track Group

LVM Logical Volume Manager

L2 Level 2

MBCS Multi-Byte Character Support

MCA Micro Channel Architecture

MDI Media Dependent Interface

MII Media Independent Interface

MST Machine State

NBC Network Buffer Cache

ND Neighbor Discovery

NDP Neighbor Discovery Protocol

NFS Network File System

NIM Network Install Manager

NIS Network Information System

NL National Language

NLS National Language Support

NVRAM Non-Volatile Random Access Memory

ODBC Open DataBase Connectivity

ODM Object Data Manager

OEM Original Equipment Manufacturer

ONC + Open Network Computing

OOUI Object-Oriented User Interface

OSF Open Software Foundation, Inc.

PCI Peripheral Component Interconnect

PEX PHIGS extension to X

PHB Processor Host Bridges

PHY Physical Layer Device

PID Process ID

PPC PowerPC

PSE Portable Streams Environment

PTF Program Temporary Fix

RAID Redundant Array of Independent Disks

RAN Remote Asynchronous Node

RDB Relational DataBase

RDISC ICMP Router Discovery

RFC Request for comments

RIO Remote I/O

RPC Remote Procedure Call

RPL Remote Program Loader

SBCS Single-Byte Character Support

SCSI Small Computer System Interface

SCSI-SE SCSI-Single Ended

SDRAM Synchronous DRAM

SHLAP Shared Library Assistant Process

SID Segment ID

SIT Simple Internet Transition

SKIP Simple Key Management for IP

SLB Segment Lookaside Buffer

SM Session Management

SMIT System Management Interface Tool

SMP Symmetrical Multi-Processor

SNG Secured Network Gateway

SP Service Processor

SPOT Shared Product Object Tree

SRC System Resource Controller

SSA Serial Storage Architecture

SSL Secure Socket Layer

STP Shielded Twisted Pair

SVC supervisor or system call

SYNC synchronization

TCE Translate Control Entry

TCP/IP Transmission Control Protocol/Internet Protocol

UCS Universal Coded Character Set

UIL User Interface Language

ULS	Universal Language Support
USLA	User-Space Loader Assistant
UTF	UCS Transformation Format
UTM	Uniform Transfer Model
UTP	Unshielded Twisted Pair
VFB	Virtual Frame Buffer
VMM	Virtual Memory Manager
VP	Virtual Processor
VSM	Visual System Manager
XCOFF	Extended Common Object File Format
XIE	X Image Extension
XIM	X Input Method
XKB	X Keyboard Extension
XPM	X Pixmap

Index

Symbols

Numerics

A

B

C

D

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

ITSO Redbook Evaluation

AIX Version 4.3 Differences Guide
SG24-2014-01

Your feedback is very important to help us maintain the quality of ITSO redbooks. **Please complete this questionnaire and return it using one of the following methods:**

- Use the online evaluation form found at http://www.redbooks.ibm.com
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Which of the following best describes you?
_ **Customer** _ **Business Partner** _ **Solution Developer** _ **IBM employee**
_ **None of the above**

Please rate your overall satisfaction with this book using the scale:
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction _____

Please answer the following questions:

Was this redbook published in time for your needs? Yes___ No___

If no, please explain:

_____

_____

_____

_____

What other redbooks would you like to see published?

_____

_____

_____

Comments/Suggestions: (THANK YOU FOR YOUR FEEDBACK!)

_____

_____

_____

_____

_____

SG24-2014-01
Printed in the U.S.A.