AIX Version 7.2

Networks and communication management

AIX Version 7.2

Networks and communication management

Note
Before using this information and the product it supports, read the information in "Notices" on page 651.

This edition applies to AIX Version 7.2 and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

About this document

This document provides application programmers with complete information about enabling applications for globalization for the AIX® operating system. It also provides system administrators with complete information about enabling networked environments for globalization for the AIX operating system. Programmers and system administrators can use this document to gain knowledge of globalization guidelines and principles. Topics include locales, code sets, input methods, subroutines, converters, character mapping, culture-specific information, and the message facility.

Highlighting

The following highlighting conventions are used in this document:

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

Case-sensitivity in AIX

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

ISO 9000

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

Networks and communication management

Both system administrators and users perform a variety of network communications tasks. System administrators can find information in this topic on how to perform such tasks as configuring TCP/IP settings, improving network security, and monitoring your system. Users can find complete information on how to perform such tasks as using communications applications and services for the operating system. Other information explains configuring and troubleshooting Mail, Message Handler (MH), Network File System (NFS), High Availability-NFS (HA-NFS), Transmission Control Protocol/Internet Protocol (TCP/IP), Basic Networking Utilities (BNU), serial communications and TTY devices, Asynchronous Terminal Emulation (ATE), and Simple Network Management Protocol (SNMP). Information on receiving and sending mail and messages, transferring files (**ftp** command), printing files from and to a remote system, running commands on other systems, communicating between local and remote systems, and customizing the communications environment is included. This topic is also available on the documentation CD that is shipped with the operating system.

What's new in Networks and communication management

Read about new or significantly changed information for the Networks and communication management topic collection.

How to see what's new or changed

In this PDF file, you might see revision bars (|) in the left margin that identify new and changed information.

February 2018

The following information is a summary of updates made to this topic collection:

- Updated information about SMBv1 in the following topics:
  - "SMBFS installation" on page 531
  - "SMBFS mounting" on page 531

October 2017

The following information is a summary of updates made to this topic collection:

- Obsolete information about Asynchronous Transfer Mode (ATM) adapters was removed.

April 2017

- Updated information about removing an adapter from EtherChannel in the "Making changes to an EtherChannel using Dynamic Adapter Membership" on page 360 topic.

Communications and networks

Understanding the general principles of computer networking has a conceptual foundation. System administrators unfamiliar with general networking principles need to read this topic. Those familiar with UNIX networking can safely skip this topic.

A network is the combination of two or more computers and their connecting links. A *physical* network is the hardware (equipment such as adapter cards, cables, and telephone lines) that makes up the network. The software and the conceptual model make up the *logical* network. Different types of networks and emulators provide different functions.

Communications

Networks allow for several user and application communication functions.

For example, they enable a user to do the following:

- Send electronic mail (e-mail)
- Emulate another terminal or log in to another computer
- Transfer data
- Run programs that reside on a remote node.

One of the most popular applications for computer networks is email, which allows a user to send a message to another user. The two users may be on the same system (in which case a communications network is not needed), different systems in different buildings, or even in different countries. The underlying layers of software and hardware, as well as the physical network, allow a user to generate, send, receive, and process messages, letters, memos, invitations, and data files. These communications can be to or from any other user who resides on the physical network. Electronic mail has the capability for message annotation, message sequencing, message packing, date sorting, and mail folder management.

Through a communications network, one computer can *emulate*, or mimic, another and access information as if it were a different type of computer or terminal. Remote login capabilities provides users with an interactive command line interface to log in to a remote system and access the same programs and files as if they were using the machine locally.

Networks also allow for the transfer of data from one system to another. Files, directories, and entire file systems can be migrated from one machine to another across a network, enabling remote backup of data, as well as assuring redundancy in case of machine failure. Password protection is usually provided as part of the protocol. With a file transfer, there is a client/server relationship between the user initiating the request and the remote system the user is accessing. Often a file transfer protocol includes functions for display and control so that users with read/write access can display, define, or delete files and directories.

Several different protocols exist that allow users and applications on one system to invoke procedures and applications on other systems. This can be useful for a number of environments, including the offloading of many computer-intensive routines in engineering and scientific applications.

Networks

The complexity of modern computer networks has given rise to several conceptual models for explaining how networks work.

One of the most common of these models is the International Standards Organization's Open Systems Interconnection (OSI) Reference Model, also referred to as the OSI seven-layer model.

The seven layers of the OSI model are numbered as follows:

Item	Description
7	**Application**
6	**Presentation**
5	**Session**
4	**Transport**
3	**Network**
2	**Data Link**
1	**Physical**

Levels 1 through 3 are network-specific, and differ depending on what physical network you are using. Levels 4 through 7 comprise network-independent, higher-level functions. Each layer describes a

particular function (instead of a specific protocol) that occurs in data communications. The seven layers function from lowest level (machine level) to highest level (the level at which most human interaction takes place), as follows:

Item	Description
Application	Comprises the applications that use the network.
Presentation	Ensures that data is presented to the applications in a consistent fashion.
Session	Manages the connections between applications.
Transport	Ensures error-free data transmission.
Network	Manages the connections to other machines on the network.
Data Link	Provides reliable delivery of data across the physical layer (which is usually inherently unreliable).
Physical	Describes the physical media of the network. For example, the fiber optic cable required for a Fiber Distributed Data Interface (FDDI) network is part of the physical layer.

Note: While the OSI Reference Model is useful for discussing networking concepts, many networking protocols do not closely follow the OSI model. For example, when discussing Transmission Control Protocol/Internet Protocol (TCP/IP), the Application and Presentation layer functions are combined, as are the Session and Transport layers and the Data Link and Physical layers.

Each layer in the OSI model communicates with the corresponding layer on the remote machine as shown in the OSI Reference Model figure.

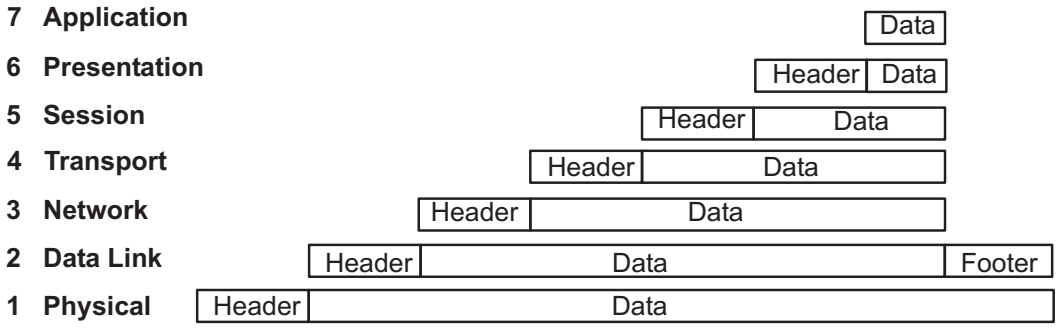


Figure 1. OSI Reference Model

This illustration shows the various communication levels of the OSI Model as described in the above text.

The layers pass data only to the layers immediately above and below. Each layer adds its own header information (and footer information, in the case of the Data Link), effectively encapsulating the information received from the higher layers.

Individual users as well as organizations use networks for many reasons, including:

- Data entry
- Data queries
- Remote batch entry
- Resource sharing
- Data sharing
- Electronic mail.

Data entry consists of entering data directly into either local or remote data files. Increased accuracy and efficiency are natural by-products of a one-step data transfer. Data queries entail searching data files for specified information. Data updating involves altering, adding, or deleting data stored in local or remote files. Remote batch entry consists of entering batches of data from a remote location, an activity often

performed at night or during periods of low system usage. Because of such diverse capabilities, communications and networks are not only desirable but necessary.

Sharing resources is another function of networks. Users can share data as well as programs, file-storage space, and peripheral devices like printers, modems, terminals, and fixed disks. Sharing of system resources is cost effective because it eliminates the problems of keeping multiple copies of programs and it keeps data consistent (in the case of program and file sharing).

Physical networks

The physical network consists of the cables (coaxial cable, twisted pair, fiber optic, and telephone lines) that connect the different hardware residing on the network, the adapter cards used on computers connected to the network (hosts), and any concentrators, repeaters, routers, or bridges used in the network.

Physical networks vary both in size and in the type of hardware used. The two common kinds of networks are *local area networks* (LANs) and *wide area networks* (WANs). A LAN is a network where communications are limited to a moderately sized geographic area of 1 to 10 km (1 to 6 miles), such as a single office building, warehouse, or campus. A WAN is a network providing data communications capability throughout geographic areas larger than those serviced by LANs, such as across a country or across continents. An intermediate class of networks exists also, called *metropolitan area networks* (MANs). This guide does not generally distinguish MANs; they are grouped with WANs.

LANs commonly use Standard Ethernet, IEEE 802.3 Ethernet, or token-ring hardware for the physical network, while WANs and asynchronous networks use communications networks provided by common carrier companies. Operation of the physical network in both cases is usually controlled by networking standards from organizations such as the Electronics Industry Association (EIA) or the International Telecommunication Union (ITU).

Network systems

All network communications involve the use of hardware and software. The system hardware and software communications support is determined by the hardware being used and the software necessary to run that hardware and interface with the network.

Hardware consists of the physical equipment connected to the physical network. *Software* consists of the programs and device drivers pertaining to the operation of a particular system. The system hardware consists of adapter cards or other devices that provide a path or interface between the system software and the physical network. An adapter card requires an input/output (I/O) card slot in the system. The adapter card connects the *data terminal equipment* (DTE) to the *data circuit-terminating equipment* (DCE); that is, it provides physical local addressing to a DTE port. Other devices, such as modems, can be attached to one of the standard ports on the computer.

An adapter card prepares all inbound and outbound data; performs address searches; provides drivers, receivers, and surge protection; supports different interfaces; and in general relieves the system processor of many communications tasks. Adapter cards support the standards required by the physical network (for example, EIA 232D, Smartmodem, V.25 bis, EIA 422A, X.21, or V.35) and may, at the same time, support software *protocols*, for example, synchronous data link control (SDLC), high-level data link control (HDLC), and bisynchronous protocols. If the adapter does not contain software support, then this support must be provided by the adapter device driver.

Protocols

All communications software use *protocols*, sets of semantical and syntactical rules that determine the behavior of functional units in achieving communication.

Protocols define how information is delivered, how it is enclosed to reach its destination safely, and what path it follows. Protocols also coordinate the flow of messages and their acknowledgments.

Protocols exist at different levels within the kernel and cannot be manipulated directly. However, they are manipulated indirectly by what the user chooses to do at the application programming interface (API) level. The choices a user makes when invoking file transfer, remote login, or terminal emulation programs define the protocols used in the execution of those programs.

Addresses

Addresses are associated with both software and hardware. The address is the means by which the sending or control station selects the station to which it sends data.

Addresses identify receiving or storage locations. A physical address is a unique code assigned to each device or workstation connected to a network.

For example, on a token-ring network, the **netstat -iv** command displays the token-ring card address. This is the physical network address. The **netstat -iv** command also displays class-level and user-level address information. Addresses are often defined by software but can be created by the user as well.

Domains

An aspect of addresses common to many communications networks is the concept of *domains*. Domains put the data processing resources in a network under a common control.

For example, the structure of the Internet illustrates how domains define the Internet Protocol (IP) address. The Internet is an extensive network made up of many different smaller networks. To facilitate routing and addressing, Internet addresses are hierarchically structured in domains, with very broad categories at the top such as `com` for commercial users, `edu` for educational users, and `gov` for government users.

Within the `com` domain are many smaller domains corresponding to individual businesses; for example, `ibm`. Within the `ibm.com` domain are even smaller domains corresponding to the Internet addresses for various locations, such as `austin.ibm.com` or `raleigh.ibm.com`. At this level, we start seeing names of *hosts*. A host, in this context, is any computer connected to the network. Within `austin.ibm.com`, there may be hosts with the names `hamlet` and `lear`, which are addressed `hamlet.austin.ibm.com` and `lear.austin.ibm.com`.

Gateways and bridges

A wide variety of networks reside on the Internet, often using different hardware and running different software. *Gateways* and *bridges* enable these different networks to communicate with each other.

A bridge is a functional unit that connects two LANs that possibly use the same logical link control (LLC) procedure, such as Ethernet, but different medium access control (MAC) procedures. A gateway has a broader range than a bridge. It operates above the link layer and, when required, translates the interface and protocol used by one network into those used by another distinct network. Gateways allow data transfers across the various networks that constitute the Internet.

Data routing

Using domain names for addressing and gateways for translation greatly facilitates the *routing* of the data being transferred. Routing is the assignment of a path by which a message reaches its destination.

The domain name effectively defines the message destination. In a large network like the Internet, information is routed from one communications network to the next until that information reaches its destination. Each communications network checks the domain name and, based on the domains with which that network is familiar, routes the information on to the next logical stop. In this way, each communications network that receives the data contributes to the routing process.

Local and remote nodes

A physical network is used by the hosts that reside on that network. Each host is a *node* on the network. A node is an addressable location in a communications network that provides host-processing services. The intercommunication of these various nodes are defined as *local* or *remote*.

Local pertains to a device, file, or system accessed directly from your system, without the use of a communications line. *Remote* pertains to a device, file, or system accessed by your system over a communications line. Local files reside on your system, while remote files reside on a file server or at another node with which you communicate using a physical network, for example, Ethernet, token-ring, or phone lines.

Client and server

A *server* is a computer that contains data or provides facilities to be accessed by other computers on the network. A *client* is a computer requesting services or data from a server.

Common server types are file servers, which store files; name servers, which store names and addresses; and application servers, which store programs and applications; print servers, which schedule and direct print jobs to their destination.

A client can request updated program code or the use of applications from a code server. To obtain a name or address, a client contacts a name server. A client could also request files and data for data entry, inquiry, or record updating from a file server.

Communication with other operating systems

Different types of computers can be connected on a network. The computers can be from different manufacturers or be different models from the same manufacturer. Communication programs bridge the differences in operating systems of two or more types of computers.

Sometimes these programs require that another program has previously been installed on the network. Other programs may require that such communications connectivity protocols as TCP/IP or Systems Network Architecture (SNA) exist on the network.

Host emulation applications

An *emulator* is a software application that allows your system to function as if you were using a different terminal or printer.

A *terminal emulator* connects to a host system to access data or applications. Some terminal emulators provide a facility to transfer files to and from the host. Others provide an application programming interface (API) to allow program-to-program communication and automation of host tasks.

A *printer emulator* allows the host either to print files on a local printer or store them in printable form to be printed or edited later.

Several applications are available to allow your system to emulate other types of terminals. This topic provides information on terminal or printer emulators.

> **Note:** The **bterm** command emulates terminals in bidirectional (BIDI) mode.

TCP/IP commands for emulation

The Transmission Control Protocol/Internet Protocol (TCP/IP) software includes the **telnet** and **rlogin** commands, which allow you to connect to and access a remote TCP/IP system.

Item	Description
telnet	Allows a user to log in to a remote host by implementing the **TELNET** protocol. It is different from the **rlogin** command in that it is a trusted command. A *trusted* command is one that meets all security levels configured on your computer. Systems that require extra security should allow only trusted commands. Standards for trusted commands, processes, and programs are set and maintained by the U.S. Department of Defense.
tn	Performs the same function as the **telnet** command.
rlogin	Allows a user to log in to a remote host. It is different from the **telnet** command in that it is a *nontrusted* command and can be disabled if your system needs extra security.

For more information about **TCP/IP**, see "Transmission Control Protocol/Internet Protocol" on page 100.

BNU commands for emulation

The Basic Networking Utilities (BNU) software includes the **ct**, **cu**, and **tip** commands, which allow you to connect to a remote system that uses the AIX operating system.

Item	Description
ct	Enables a user on a remote terminal, such as a 3161, to communicate with another terminal over a telephone line. The user on the remote terminal can then log in and work on the other terminal. The **ct** command is similar to the **cu** command but not as flexible. For example, you cannot issue commands on the local system while connected to a remote system through the **ct** command. However, you can instruct the **ct** command to continue dialing until the connection is established or to specify more than one telephone number at a time.
cu	Connects your terminal to another terminal connected to either a UNIX or non-UNIX system. After the connection is established, you can be logged in on both systems at the same time, executing commands on either one without dropping the BNU communication link. If the remote terminal is also running under UNIX, you can transfer ASCII files between the two systems. You can also use the **cu** command to connect multiple systems, and commands can then be executed on any of the connected systems.
tip	Connects your terminal to a remote terminal and enables you to work on the remote terminal as if logged in directly. You can use the **tip** command to transfer files to and from the remote system. You can use scripting to record the conversations you have with the **tip** command. **Note:** You must have a login on the remote system to use the **tip** command.

For more information about BNU, see "Basic Networking Utilities" on page 406.

Asynchronous Terminal Emulation

The Asynchronous Terminal Emulation (ATE) program enables your terminal to connect to most systems that support asynchronous terminals, including any system that supports RS-232C or RS-422A connections.

ATE allows the remote system to communicate with your terminal either as an asynchronous display or as a DEC VT100 terminal.

ATE enables you to run commands on the remote system, send and receive files, and check the data integrity in the files transferred between systems. You can also use a capture file to record, or *capture*, incoming data from the remote system. ATE is menu-driven and uses subcommands.

When installed, ATE is accessible only to users who have been registered as a member of the UUCP group by a user with root authority.

For more information about ATE, see "Asynchronous Terminal Emulation" on page 593.

Communications system commands

This describes commands available for displaying information that identifies users on your system, the system you are using, and users logged in to other systems.

See the following topics for the various commands used to provide system and user information.

Displaying your login name

Use the **whoami** command to determine your login name.

To display the name of the current user, enter:

```
whoami
```

A display similar to the following is returned:

```
denise
```

In this example, the login name is `denise`.

Displaying your system name

Use the **uname** command to determine your system name.

1. To display the name of your system if you are on a network, enter:

   ```
   uname -n
   ```

   A display similar to the following is returned:

   ```
   barnard
   ```

   In this example, the system name is `barnard`.
2. To find the node name of another system, request that a user on that system enter the **uname -n** command.

Determining whether your system has access

Use the **host** to determine whether your system has access to information that defines the other system.

To access another system on the network, your local system must have access to information that defines the other system. To determine if your local system has this information, enter the **host** command with the name of the other system.

To determine if your local system has routing information for system `zeus`, enter:

```
host zeus
```

If your system has the proper information, a display similar to the following is returned:

```
zeus is 192.9.200.4 (300,11,310,4)
```

You can then send a message to system `zeus`. The address `192.9.200.4` is used by the system to route the mail. If your system does not have the information, a display similar to the following is returned:

```
zeus: unknown host
```

If you receive an `unknown host` message, then the requested system name:

- Is not correct (check the spelling in the address)
- Is on your network but not defined to your system (contact the person responsible for setting up your network)
- Is on another network (see "Addressing mail to users on a different network" on page 23) and requires more detailed addressing
- Is not connected to your network

You can also receive the `unknown host` message if your network is not operating and your local system depends on a remote system to supply network addresses.

Displaying information about logged-in users

Use the **finger** or **f** command to display information about the current users on a specified host.

This information can include the user's login name, full name, and terminal name, as well as the date and time of login.

1. To display information about all users logged in to host `@alcatraz`, enter:

   ```
   finger @alcatraz
   ```

   A display similar to the following is returned:

   ```
   brown  Console Mar 15 13:19
   smith  pts0    Mar 15 13:01
   jones  tty0    Mar 15 13:01
   ```

   User `brown` is logged in at the console, user `smith` is logged in from a pseudo teletype line pts0, and user `jones` is logged in from a tty0.
2. To get information about the user `brown` from the preceding example, enter:

   ```
   finger brown@alcatraz
   ```

   or

   ```
   finger brown
   ```

   A display similar to the following is returned:

   ```
   Login name: brown
   In real life: Marta Brown
   Directory:/home/brown  Shell: /bin/ksh
   On since May 8 07:13:49 on console
   No Plan.
   ```

Mail management

The mail facility provides a method for exchanging electronic mail (e-mail) with users on the same system or on multiple systems connected by a network. The mail system, the standard mail user interface, the **Internet Message Access Protocol (IMAP)**, and the **Post Office Protocol (POP)** are described here.

The mail system is an internetwork mail delivery facility that consists of a user interface, a message routing program, and a message delivery program (or mailer). The mail system relays messages from one user to another on the same host, between hosts, and across network boundaries. It also performs a limited amount of message-header editing to put the message into a format that is appropriate for the receiving host.

A mail *user interface* enables users to create and send messages to, and receive messages from, other users. The mail system provides two user interfaces, **mail** and **mhmail**. The **mail** command is the standard mail user interface available on all UNIX systems. The **mhmail** command is the Message Handler (MH) user interface, an enhanced mail user interface designed for experienced users.

A *message routing program* routes messages to their destinations. The mail system message routing program is the **sendmail** program, which is part of the Base Operating System (BOS) and is installed with BOS. The **sendmail** program is a daemon that uses information in the `/etc/mail/sendmail.cf` file, the `/etc/mail/aliases` file to perform the necessary routing.

Depending on the type of route to the destination, the **sendmail** command uses different *mailers* to deliver messages.

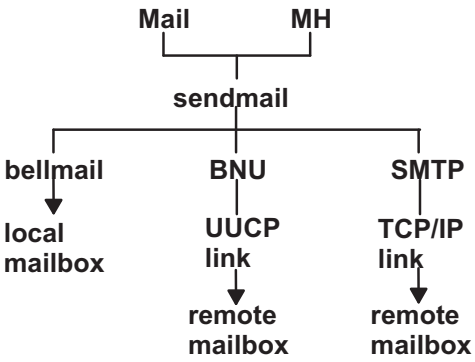


Figure 2. Mailers used by the sendmail command

This illustration is a type of top-down organizational chart with Mail and MH at the top. Branching from them are bellmail, BNU and SMTP. Underneath the previous level are local mailbox, UUCP link, and **TCP/IP** link respectively. Beneath UUCP link is remote mailbox and under **TCP/IP** link is remote mailbox.

As the figure illustrates:

- To deliver local mail, the **sendmail** program routes messages to the **bellmail** program. The **bellmail** program delivers all local mail by appending messages to the user's system mailbox, which is in the /var/spool/mail directory.
- To deliver mail over a UNIX-to-UNIX Copy Program (UUCP) link, the **sendmail** program routes messages using Basic Network Utilities (BNU).
- To deliver mail routed through **Transmission Control Protocol/Internet Protocol (TCP/IP)**, the **sendmail** command establishes a **TCP/IP** connection to the remote system then uses **Simple Mail Transfer Protocol (SMTP)** to transfer the message to the remote system.

Mail user-agent programs

Before you can use the mail system, you must select a user-agent program. You can choose a mail program (**mail**), message handler (**mh**), or the **bellmail** command.

A user-agent program provides facilities for creating, receiving, sending, and filing mail. In addition, you need a transport-agent program, **sendmail**, which distributes incoming mail from other systems or packages and distributes each outgoing mail item and then transmits it to a similar program in one or more remote systems.

> **Note:** The **mail** and **mh** programs are incompatible in the way they store mail; you must choose one mail handler or the other.

Mail program interface

The **mail** program provides you with a user interface to handle mail to and from both a local network user and a remote system user.

A mail message can be text, entered using an editor, or an ASCII file. In addition to a typed message or a file, you can send:

Item	Description
system message	Informs users the system has been updated. A system message is similar to a broadcast message but is sent on the local network only.
secret mail	Used to send classified information. A secret mail message is encrypted. The recipient must enter a password to read it.
vacation message	Informs users you are on vacation. When your system receives mail in your absence, it sends a message back to the origin. The message states you are on vacation. Any mail you receive while on vacation can also be forwarded.

When you receive mail using the **mail** subcommands, you can:

- Leave the mail in the system mailbox.
- Read and delete the mail.
- Forward the mail.
- Add comments to the mail.
- Store the mail in your personal mailbox (mbox).
- Store the mail in a folder you have created.
- Create and maintain an alias file or a distribution file, which directs the mail and mail messages.

The installation of **sendmail** is automatic.

For more information about the **mail** program, refer to "Mail functions" on page 12.

Message handler (mh)

The **mh** program is a collection of commands that enables you to perform each mail processing function directly from the command line.

These commands provide a broader range of function than the subcommands of **mail**. Also, because the commands can be issued at any time the command prompt is displayed, you gain power and flexibility in creating mail and in processing received mail. For example, you can read a mail message, search a file or run a program to find a particular solution, and answer the message, all within the same shell.

The **mh** program enables you to create, distribute, receive, view, process, and store messages using the following commands:

Item	Description
ali	Lists mail aliases and their addresses.
anno	Annotates messages.
ap	Parses and reformats addresses.
burst	Explodes digests into messages.
comp	Starts an editor for creating or modifying a message.
dist	Redistributes a message to additional addresses.
dp	Parses and reformats dates.
folder	Selects and lists folders and messages.
folders	Lists all folders and messages in the mail directory.
forw	Forwards messages.
inc	Incorporates new mail into a folder.
mark	Creates, modifies, and displays message sequences.
mhl	Produces a formatted listing of messages.
mhmail	Sends or receives mail.
mhpath	Prints full path names of messages and folders.
msgchk	Checks for messages.
msh	Creates a mail handler (mh) shell.
next	Shows the next message.
packf	Compresses the contents of a folder into a file.
pick	Selects messages by content and creates and modifies sequences.
prev	Shows the previous message.

Item	Description
refile	Moves files between folders.
repl	Replies to a message.
rmf	Removes folders and the messages they contain.
rmm	Removes messages from active status.
scan	Produces a one-line-per-message scannable listing.
send	Sends a message.
show	Shows messages.
sortm	Sorts messages.
vmh	Starts a visual interface for use with **mh** commands.
whatnow	Starts a prompting interface for draft disposition.
whom	Manipulates **mh** addresses.

For more information on **mh** commands, refer to the *Commands Reference, Volume 3*.

bellmail command

The **bellmail** command is the original AT&T UNIX mail command, which handles mail for users on the same system and also for users on remote systems that can be accessed by means of Basic Network Utilities (BNU), sometimes known as the UNIX-to-UNIX Copy Program (UUCP).

These programs support only networks of systems connected by dialup or leased point-to-point communication lines. The command opens a shell whose subcommands allow you to:

- Take data from standard input (typed in or redirected from an existing file), add one or more addresses (supplied as arguments to the command itself) and a timestamp, then append a copy to each addressee's system mailbox file (`/var/spool/mail/`*`UserID`*).
- Read mail items from your system mailbox file.
- Append mail items to your personal mailbox file (`$HOME/mbox`) or to a specified file.
- Send mail using BNU to a user on another system.
- Automatically redirect all mail from your system mailbox to one on another system by adding a *.forward* statement to the beginning of your system mailbox file.

However, you must have some skill as a UNIX user before you can make full use of this mail handler. For more information, refer to the **bellmail** command in the *Commands Reference, Volume 1*.

Mail functions

The features of the **mail** program are introduced here.

The **mail** program enables you to receive, create, and send mail to users on a local or remote system.

Mail storage

Mail is stored in different ways, depending on the specific situation.

When mail is sent to your address, it is stored in a system directory that is specifically for mail. This system directory contains a file for every user on the local system. This directory holds your mail until you do something with it.

System mailbox:

The system mailbox is similar to a post office box: the post office delivers letters addressed to the person who owns that box.

Similarly, the system mailbox is a file where messages are delivered to a particular user. If the file does not exist when mail arrives, it is created. The file is deleted when all messages have been removed.

System mailboxes reside in the /var/spool/mail directory. Each system mailbox is named by the user ID associated with it. For example, if your user ID is karen, your system mailbox is:

```
/var/spool/mail/karen
```

Default personal mailbox:

Your personal mailbox is similar to an in-basket in an office. You put mail in the in-basket after you have received it, but before you have filed it.

Each user has a personal mailbox. When you read mail from the system mailbox, and if it is not marked for deletion or saved to a file, it is written to your personal mailbox, $HOME/mbox ($HOME is your login directory). The mbox file exists only when it contains a message.

dead.letter file for incomplete messages:

If you need to interrupt a message you are creating to complete other tasks, the system saves incomplete messages in the dead.letter file in the $HOME directory.

If the dead.letter file does not exist, the file is created. Later you can edit the file to complete your message.

> **Attention:** Do not use the dead.letter file to store messages. The content of this file is overwritten each time an interrupt is issued to save a partial message to the dead.letter file.

Mail folders:

Folders enable you to save messages in an organized fashion. Using the mail program, you can put a message into a folder from the system mailbox, a personal mailbox, or another folder.

Each folder is a text file. Each folder is placed in the directory you specify in your .mailrc file with the **set folder** option. You must create this directory before using folders to store messages. When the directory exists, the mail program creates the folders in that directory as needed. If you do not specify a directory in your .mailrc file, folders are created in the current directory. See "Organizing mail" on page 19.

> **Note:** Several programs are available to send and receive mail, including Message Handler (MH) and the **bellmail** program. Which program you use depends on what is installed and configured on your system. For information on your system configuration, contact your system administrator.

Mail handling and receiving

The **mail** program enables you to examine each message in a mailbox and then delete or file a message in a personal mail directory.

The command shell notifies you that mail has arrived. The notification is displayed before the next prompt, provided that the **MAIL** environment variable is set and provided that the interval specified by **MAILCHECK** has elapsed since the shell last checked for mail. The notification message is the value of the **MAILMSG** environment variable. Depending on which shell you are using (bourne, korn, or C shell), the notification is similar to the following:

```
YOU HAVE NEW MAIL
```

Mailbox startup:

Use the **mail** command to read and remove messages from your system mailbox.

Do not use the system mailbox to store messages. Store messages in your personal mailbox and in mail folders.

Checking your system mailbox for mail:

Use the **mail** command to check your system mailbox for mail.

At your system command line prompt, enter the **mail** command:

```
mail
```

If there is no mail in your system mailbox, the system responds with a message:

```
No mail for YourID
```

If there is mail in your mailbox, the system displays a listing of the messages in your system mailbox:

```
Mail Type ? for help.
"/usr/mail/lance": 3 messages 3 new
  >N    1 karen Tue Apr 27 16:10 12/321 "Dept Meeting"
   N    2 lois  Tue Apr 27 16:50 10/350 "System News"
   N    3 tom   Tue Apr 27 17:00 11/356 "Tools Available"
```

The current message is always prefixed with a greater-than symbol (>). Each one-line entry displays the following fields:

Item	Description
status	Indicates the class of the message.
number	Identifies the piece of mail to the mail program.
sender	Identifies the address of the person who sent the mail.
date	Specifies the date the message was received.
size	Defines the number of lines and characters contained in the message (this includes the header).
subject	Identifies the subject of the message, if it has one.

The status can be any of the following:

Item	Description
N	A new message.
P	A message that will be preserved in your system mailbox.
U	An unread message. This is a message that was listed in the mailbox the last time you used the mail program, but the contents were not examined.
*	A message that was saved or written to a file or folder.

A message without a status indicator is a message that has been read but has not been deleted or saved.

Checking your personal mailbox or mail folder for mail:

You can use the **mail** command to check your personal mailbox or mail folder for mail.

At your system command line prompt, you can use the **mail** command in the ways shown in the following steps:

1. To display a listing of the messages in your personal mailbox, `$HOME/mbox`, enter:

   ```
   mail -f
   ```

   If there is no mail in your personal mailbox, the system responds with a message similar to the following:

   ```
   "/u/george/mbox": 0 messages
   ```

   or

   ```
   A file or directory in the path name does not exist
   ```

2. To display a listing of the messages in the `dept` folder, enter:

```
mail -f +dept
```

If there is no mail in your mail folder, the system responds with a message similar to the following:

```
A file or directory in the path name does not exist
```

Mailbox content display options:

From the mailbox prompt, you can enter mailbox subcommands to manage the contents of the mailbox.

Prerequisites

1. The mail program must be installed on your system.
2. The mail program must be started.
3. There must be mail in your mailbox.

Ranges of messages:

Use the **h** subcommand to view a message contained within a list of messages that you determine so that you do not have to browse through all your messages.

At your mailbox prompt, you can use the **h** subcommand in the ways shown in the following examples:

Item	Description
h	Approximately 20 messages are displayed at a time. The actual number displayed is determined by the type of terminal being used and the **set screen** option in your `.mailrc` file. If you enter the **h** subcommand again, the same range of messages is displayed.
h 21	Message 21 and subsequent messages, up to and including message 40 (if you have that number of messages in your mailbox), are displayed. Continue typing the **h** subcommand with the subsequent message number until all messages have been displayed.
h 1	To return to the first group of 20 messages, enter any number within the range of 1-20.

Mailbox scrolling:

Use the **z** subcommand to scroll through your mailbox.

At your mailbox prompt, you can use the **z** subcommand in the ways shown in the following examples:

Item	Description
z	Approximately 20 messages are displayed at a time. The actual number displayed is determined by the type of terminal being used and the **set screen** option in your `.mailrc` file. enter the **z** subcommand again to scroll to the next 20 messages.
z +	The plus sign (**+**) argument scrolls to the next 20 messages. Message 21 and subsequent messages, up to and including message 40 (if you have that number of messages in your mailbox), are displayed. Continue typing the **z+** subcommand until all messages have been displayed. The system will respond with the following message: `On last screenful of messages.`
z -	The minus sign (**-**) argument scrolls to the previous 20 messages. When you reach the first set of messages, the system will respond with the following message: `On first screenful of messages.`

Message filtering for specific information:

At your mailbox prompt, you can use the **f** subcommand in the ways shown in the following examples to filter messages according to the information that you want.

Item	Description
f	Displays header information for the current message.
f 1 4 7	Displays header information for the specific messages 1, 4, and 7.
f 1-10	Displays header information for a range of messages 1 through 10.
f *	Displays all messages.
f ron	Messages, if any, from user ron are displayed. The characters entered for an address do not need to exactly match the address; therefore, the request for address ron in either uppercase or lowercase letters matches all of the following addresses: RoN ron@topdog hron rOn
fmeet	Messages, if any, where the **Subject:** field contains the letters meet are displayed. The characters entered for a pattern do not need to exactly match the **Subject:** field. They must only be contained in the **Subject:** field in either uppercase or lowercase letters; therefore, the request for subject meet matches all of the following subjects: Meeting on Thursday Come to meeting tomorrow MEET ME IN ST. LOUIS

Current message numbers:

The **=** subcommand displays message numbers.

At your mailbox prompt, you can use the **=** subcommand in the way shown in the following example:

Item	Description
=	The current message number is displayed.

Total number of messages in your mailbox:

Use the **folder** subcommand to check how many messages are in your mailbox.

At your mailbox prompt, you can use the **folder** subcommand in the way shown in the following example:

Item	Description
folder	Lists information about your folder or mailbox. The system will respond similarly to the following: "/u/lance/mbox": 29 messages.

Reading mail options:

You can read your mail in several ways. Examples of each method are described here.

Choose the method you are most comfortable with, and use it to read your mail. Before attempting to read your mail, make sure the following conditions are true:

1. The mail program must be installed on your system.
2. The mail program must be started.
3. There must be mail in your system mailbox.

Reading messages in your mailbox:

Use the **t** or **p** subcommand to read messages in your mailbox.

At your mailbox prompt, you can use the **t** or **p** subcommands in the ways shown in the following examples:

Item	Description
3	If you use the number of the message, by default, the text of the message is displayed.
t	If you use the **t** subcommand, by default, the text of the current message is displayed.
t 3	The text of message 3 is displayed.
t 2 4 9	The text for messages 2, 4, and 9 is displayed.
t 2-4	The text for the range of messages 2 through 4 is displayed.
t	If you use the **p** subcommand, by default, the text of the current message is displayed.
p 3	The text of message 3 is displayed.
p 2 4 9	The text for messages 2, 4, and 9 is displayed.
p 2-4	The text for the range of messages 2 through 4 is displayed.

Reading the next message in your mailbox:

Use the **n** subcommand to read the next message in your mailbox.

At your mailbox prompt, you can use the (**n**)ext or plus sign (+) subcommand in the way shown in the following example:

Item	Description
n or +	Displays the text of the next message, and this message becomes the current message.

You can also press the Enter key to display the text of the next message.

Reading the previous message in your mailbox:

Use the **-** subcommand to read the previous message.

At your mailbox prompt, you can use the **-** subcommand in the way shown in the following example:

Item	Description
-	The text of the previous message is displayed.

Deleting mail:

When deleting a message, you can delete the current message, delete a specific message, or delete a range of messages.

You can also delete the current message and display the next message by combining subcommands. Ensure that the following conditions are met:

1. The mail program must be installed on your system.
2. There must be mail in your system mailbox.
3. The mail program must be started.

Deleting messages:

Use various forms of the **d** subcommand to delete messages.

At your mailbox prompt, you can use the (**d**)elete subcommand in the ways shown in the following examples:

Item	Description
d	The current message is deleted.
dp or **dt**	The current message is deleted and the next message is displayed. This also can be accomplished by including the **set autoprint** option in the `.mailrc` file, which will set the **d** subcommand to function like the **dp** or **dt** subcommand combination.
d `4`	Deletes the specific message 4.
d `4-6`	Deletes a range of messages 4 through 6.
d `2 6 8`	Deletes messages 2, 6, and 8.

Undeleting messages:

Use the **u** subcommand for undeleting messages.

At your mailbox prompt, you can use the **u** subcommand in the ways shown in the following examples:

Item	Description
u	The current message is undeleted.
u `4`	Undeletes the specific message 4.
u `4-6`	Undeletes a range of messages 4 through 6.
u `2 6 8`	Undeletes messages 2, 6, and 8.

Exiting mail:

Ensure that the following requirements are met before exiting the mail program.

1. The mail program must be installed on your system.
2. There must be mail in your system mailbox.
3. The mail program must be started.

Exiting mail and saving changes:

Use the **q** subcommand to exit mail and save changes.

If you are exiting the system mailbox:

Item	Description
q	The **q** subcommand leaves the system mailbox and returns to the operating system. When you leave the mailbox, all messages marked to be deleted are removed from the mailbox and cannot be recovered. The mail program saves the messages you read in your personal mailbox (**mbox**). If you did not read any of your mail, the messages remain in the system mailbox until acted upon.

If you are exiting your personal mailbox or a mail folder:

Item	Description
q	When using the **q** subcommand in your personal mailbox or a mail folder, messages read and not read will remain in your personal mailbox or in a mail folder until acted upon.

Exiting mail without saving changes:

Use the **x** or **ex** subcommand to exit mail without making mailbox changes.

Item	Description
x or **ex**	The **x** or **ex** subcommand allows you to leave the mailbox and return to the operating system without changing the original contents of the mailbox. The program ignores any requests you made prior to the **x** request; however, if you did save a message to another folder, the save will occur.

Organizing mail:

Use folders to save messages in an organized fashion.

You can create as many folders as you need. Give each folder a name that pertains to the subject matter of the messages it contains, similar to file folders in an office filing system. Each folder is a text file that is placed in the directory you specify in your `.mailrc` file with the **set folder** option. You must create this directory before using folders to store messages. When the directory exists, the mail program creates the folders in that directory as needed. If you do not specify a directory with the **set folder** option in your `.mailrc` file, the folder is created in your current directory. Using the mail program, you can put a message into a folder from the system mailbox, a personal mailbox, or another folder.

You can add the contents of a message to a file or folder using the **s** or **w** subcommands. Both of these subcommands append information to an existing file or create a new file if it does not exist. Information currently in the file is not destroyed. If you save a message from your system mailbox to a file or folder, the message is deleted from your system mailbox and transferred to the file or folder specified. If you save a message from your personal mailbox or folder to another file or folder, the message is not deleted from your personal mailbox but is copied to the specified file or folder. When using the **s** subcommand, you can read the folder like a mailbox because the messages and the header information are appended at the end of the folder. When using the **w** subcommand, you can read the folder like a file because the message is appended without header information at the end of the file.

Before organizing mail, ensure that the following requirements are met:

1. The mail program must be installed on your system.
2. There must be mail in your system mailbox, personal mailbox, or a folder you have defined.
3. The mail program must be started.

Creating a letters mailbox directory to store messages in folders:

Messages can be saved in a mailbox directory folder using the **set folder** subcommand.

Use the following procedure to store messages in folders:

1. To check if the **set folder** option has been enabled in the `.mailrc` file, enter the following subcommand at the mailbox prompt:

   ```
   set
   ```

   The **set** subcommand displays a list of the enabled mail options in your `.mailrc` file.

   If the **set folder** option has been enabled, the system responds with a message similar to the following:

   ```
   folder /home/george/letters
   ```

   In this example, `letters` is the directory in which mail folders will be stored.
2. If the **set folder** option has not been enabled, add a line similar to the following in the `.mailrc` file:

   ```
   set folder=/home/george/letters
   ```

   In this example, `/home/george` is George's home directory and `letters` is the directory in which mail folders will be stored. The **set folder** option enables you to use the plus sign (+) shorthand notation at your mailbox prompt to save messages in your `letters` directory.
3. You must create a `letters` directory in your home directory. In your home directory at the system command line prompt, type:

```
mkdir letters
```

Saving messages with headers:

The **s** subcommand saves messages with headers.

Use the **s** subcommand in the following ways:

Item	Description
s 1-4 notes	Saves messages 1, 2, 3 and 4 with their header information to a folder called notes in the current directory. The mail program responds with the following message: "notes" [Appended] 62/1610
s +admin	Saves the current message to an existing folder called admin in your folder directory. If the folder directory is defined as /home/george/letters in your .mailrc file, the system responds with: "/home/george/letters/admin" [Appended] 14/321
s 6 +admin	Saves message 6 to an existing folder called admin in your folder directory. If the folder directory is defined as /home/george/letters in your .mailrc file, the system responds with: "/home/george/letters/admin" [Appended] 14/321

Saving messages without headers:

Use the **w** subcommand to save a message as a file instead of as a folder.

To read or edit a file saved with the **w** subcommand, you must use **vi** or some other text editor. At your mailbox prompt, you can use the **w** subcommand in the following ways:

Item	Description
w 6 pass	Saves only the text of message 6 to a file called pass in the current directory. If the pass file does not already exist, the system responds with the following message: "pass" [New file] 12/30 If the pass file exists, the system responds with the following message: "pass" [Appended] 12/30
w 1-3 safety	Saves only the text of the specific messages 1, 2, and 3 to a file called safety in the current directory. The text of the messages in this example will be appended one after the other into one file. If the safety file does not already exist, the system responds with the following message: "safety" [New file] 12/30

Determining the current mailbox or folder:

Use the **folder** subcommand to determine the current mailbox or folder.

Although the **mail** command displays the name of the current mailbox when it starts, you might lose track of which mailbox you are in. At your mailbox prompt, you can use the **folder** subcommand shown in the following example:

Item	Description
folder	Finds the name of your current mailbox or folder. If the current mailbox is `/home/lance/mbox`, the following is displayed: `/home/lance/mbox: 2 messages 1 deleted` This message indicates that `/home/lance/mbox` is the current mailbox you are in, it contains two messages, and one of those messages will be deleted when you finish with this mailbox.

Changing to another mailbox:

Changing to another mailbox is like quitting a mailbox or folder.

Any messages that you marked to be deleted are deleted when you leave that mailbox. The deleted messages cannot be recovered. At your mailbox prompt, you can use the **file** or **folder** subcommand shown in the following example:

Item	Description

folder `+project`	After the mail program is started with one mailbox, use the **file** or **folder** subcommands to change to another mailbox.
	
	If you change from the `mbox` file to the `mbox` folder and you have deleted all the messages in the `mbox` file, the mail program displays:
	`/home/dee/mbox removed`
	`+project: 2 messages 2 new`
	
	followed by a list of the messages in the `project` folder.

Creating and sending mail

You can use the **mail** program to create, send, reply, and forward messages to other users or to send ASCII files to other users.

An ASCII file might, for example, be a document you have written using a preferred editor or a source file for a program.

You can send messages and files to a user on your local system, on your network, or to a user on another connected network. The recipient does not need to be logged on to the system when you send the information. Mail is sent to a user's address.

Addressing mail:

Mail is sent to a user's address. The address, containing the login name and system name, directs the delivery of the mail message.

Generally, to send a message to another user, you must enter the **mail** command and the address as follows:

`mail` *`User@Address`*

The format of the *Address* parameter depends upon the location of the recipient. The concept is similar to how you might address a note to a fellow worker in an office. To send a note to Ryan, who works in a small department of six to eight people, you might write the name on an envelope and put it in the office mail system. However, if Ryan is in another department, you might have to provide more information on the envelope:

```
Ryan
Payroll
```

If Ryan is in another geographic location, you might need even more information to ensure that the message reaches him:

```
Ryan
Payroll
Gaithersburg
```

To send mail electronically, use a similar addressing progression:

Item	Description
mail `ryan`	To send mail to a user on your local system, the login name is the only part of the address required.
mail `ryan@tybalt`	To send mail to a user on your local network, enter the full system (node) address.
mail `ryan@mars.aus.dbm.com`	To send mail to a user on another connected network, enter the full system address and network address.
mail `dept71`	You can send mail to a specific group of people by using an alias or distribution list. To do so, you must create an alias or distribution list in your `.mailrc` file. If you need information on creating aliases, see "Aliases and distribution lists" on page 37.

Addressing mail to more than one user:

To address mail to more than one user at the same time, separate each user name with a space.

For example:

```
ryan@tybalt suemc@julius dmorgan@ophelia
```

Addressing mail to users on your local system:

To send a message to a user on your local system (to someone whose login name is listed in your `/etc/passwd` file), use the login name for the address.

At your system command line prompt, you can use the **mail** command shown in the following example:

```
mail LoginName
```

Item	Description
mail `ryan`	If Ryan is on your system and has the login name `ryan`, this command activates the mail program, enables you to create a message, and tries to send the message to a local login name of `ryan`. If the message is delivered successfully, you receive no notification. If Ryan is not on your system, the mail system immediately returns an error message and returns the unsent message to your system mailbox.

Addressing mail to users on your network:

Use the **mail** command to send a message to users on your network. Include the user's login name and system name in the address.

To send a message through a local network to a user on another system, at the command line, type:

Item	Description
mail *LoginName@SystemName*	For example, if Ryan is on system `zeus`, use the following command to create and send a message to him: `mail ryan@zeus` This command activates the mail program, enables you to create a message, and tries to send the message to login name `ryan` on system `zeus`. If the message is delivered successfully, you receive the system prompt with no notification. If the mail address is incorrect, you receive an error message.

Note: To send a message through a local network to a user on another system, you must know the login name and the name of the other system. For more information about displaying information that identifies users, see "Communications system commands" on page 8.

Addressing mail to users on a different network:

If your network is connected to other networks, you can send mail to users on the other networks.

The address parameters differ depending on how your network and the other networks address each other and how they are connected. Depending on how your network configuration, take one of these actions:

- If you are using a central database of names and addresses, use the **mail** command shown in the following example:

  mail *LoginName@SystemName*

  If the networks use a central database of names, you do not need any additional information to send mail to users on the connected networks. Use the same addressing format as for users on your local network.

  This type of addressing works well when the nature of the network allows a central database of names to be maintained.
- If your network uses domain name addressing, use the **mail** command shown in the following example:

  mail *LoginName@SystemName.DomainName*

  For networks that span large, unrelated networks in widespread locations, a central database of names is not possible. The *DomainName* parameter defines the remote network, relative to your local network, within the defined structure for the larger group of interconnected networks.

  For example, if you enter the following command:

  ```
  mail kelly@merlin.odin.valryanl
  ```

  your mail is sent to user `kelly` on the system `merlin`, which is on a local network named `odin` that is connected to a second network whose domain is called `valryanl`.

Mail addresses over a BNU or UUCP link:

You can send messages to users on another system over a Basic Networking Utilities (BNU) or UNIX-to-UNIX Copy Program (UUCP) link.

To send a message to a user on another system connected to your system by the BNU or another version of UUCP, you must know:

- The login name
- The name of the other system
- The physical route to that other system

The person responsible for connecting your system to other systems should be able to provide routing information to address the other system.

When Your Computer Has a BNU or UUCP Link: At your system command line prompt, use the **mail** command as shown in the following examples:

Item	Description
mail *UUCPRoute*`!`*LoginName*	If your local computer has a BNU or UUCP connection that can be used to reach the remote system, use the format in this example to address a message. The variable *LoginName* is the login name on the remote system for the message recipient. The variable *UUCPRoute* describes the physical route the message must follow along the UUCP network. If your system is connected to the remote system without any intermediate UUCP systems between, this variable is the name of the remote system.
mail`arthur!lancelot!merlin!ken`	If your message must travel through one or more intermediate UUCP systems before reaching the desired remote system, this variable is a list of each of the intermediate systems. The list starts with the nearest system and proceeds to the farthest system, separated by an exclamation mark (!). You can follow this example, if the message must travel through systems `arthur` and `lancelot` (in that order) before reaching `merlin`.
mail `merlin!ken`	If your local system has a UUCP link to a system called `merlin` and there are no other UUCP systems between your system and `merlin`, you can send a message to `ken` on that system.

When the BNU or UUCP Link Is on Another Computer: In a local or wide area network environment, one of the systems on the network may have a BNU or other type of UUCP connection to a remote system. You can use that UUCP connection to send a message to a user on that remote UUCP system. At your system command line prompt, use the **mail** command as shown in the following example:

mail @arthur:merlin!ken
: Sends mail to `ken` on UUCP system `merlin` from the Internet system `arthur`. The delimiter @ is for the internet addressing, the delimiter ! is for the UUCP addressing, and the delimiter : connects the two addresses. Notice that in this format you are not sending mail to a user at any of the intermediate systems, so no login name precedes the @ in the domain address.

mail @arthur:odin!acct.dept!kelly
: Sends mail to `kelly` on UUCP system `acct.dept` through system `odin` from the Internet system `arthur`.

mail@odin.uucp:@dept1.UUCP:@dept2:bill@dept3
: Sends mail to `bill@dept3` over `odin` and `dept1` UUCP links and then over the local network link between systems `dept2` and `dept3`. The `/etc/sendmail.cf` file must be configured accordingly to use this type of UUCP address notation. See your system administrator for information.

If you often send mail to users on other networks, creating aliases that include the users' addresses can save you time. See “Aliases and distribution lists” on page 37.

Starting the mail editor:

The **mail** program provides a line-oriented editor for creating messages.

1. The mail program must be installed on your system.
2. The mail program must be started.

This editor enables you to type each line of the message, press the Enter key to get a new line, and type more text. You cannot change a line after you have pressed the Enter key. However, before pressing the Enter key, you can change information on that one line by using the Backspace and Delete keys to erase. You can also use mail editor subcommands to enter a full-screen editor and change the message.

When you create mail with the mail editor, the **date:** and **from:** fields are automatically completed by the system. You have the option to complete the **subject:** and **cc:** fields. These fields are similar to the body of a standard business letter.

The mail editor includes many control subcommands that enable you to perform other operations on a message. Each of these subcommands must be entered on a new line and must begin with the special *escape* character. By default, the escape character is a tilde (~). You can change it to any other character by including the **set escape** option in your `.mailrc` file.

At your system command line prompt or your mailbox prompt, you can use the **mail** command as shown in the following examples:

Item	Description
mail *User@Address*	Issue this command from the command line prompt. The message is addressed to *User@Address*. The *Address* parameter depends upon the location of the recipient.
m*User@Address*	Issue this subcommand from the mailbox prompt. The message is addressed to *User@Address*. The *Address* parameter depends upon the location of the recipient.

The mail editor is also activated, if you use the **R** or **r** subcommands to reply to a message. For more information on how to reply to a message, see "Sending mail" on page 30 and "Replying to mail" on page 30.

Message editing:

While in your mailbox, you can add information to an existing message by typing the (**e**)dit or (**v**)isual subcommand at the mailbox prompt.

While in the mail editor, you cannot change information on a line after you have pressed the Enter key and gone on to the next line. You can change the content of your message before sending it by editing the message with another editor.

Before editing a message in another editor, make sure the following conditions are true:

1. The mail program must be installed on your system.
2. The alternate editor must be defined in the `.mailrc` file with:

   `set EDITOR=`*`PathName`*

   This defines the editor you activate with the **~e** subcommand. The value of *PathName* must be the full path name to the editor program that you want to use. For example, the definition `set EDITOR=/usr/bin/vi` defines the **vi** editor for use with the **~e** subcommand.
3. To add information to a message in your mailbox, you must have started the **mail** command to read mail in the system mailbox, other mailbox, or folder.
4. To start an alternate editor while creating a message, you must be in the mail editor prompt.

Adding information to a specific message in your mailbox:

To add information to a message in your mailbox, enter the **e** subcommand or the **v** subcommand, followed by the message number.

At the mailbox prompt, you can use either the **e** subcommand or the **v** subcommand as shown in the following examples:

Item	Description
e 13	To add a note to message 13 using the **e** editor (or whatever editor is defined in the `.mailrc` file).
v 15	To add a note to message 15 using the **vi** editor (or whatever editor is defined in the `.mailrc` file).

If you do not specify a message number, the **mail** command activates the editor using the current message. When you leave the editor, you return to the mailbox prompt to continue processing the messages in the mailbox.

Changing the current message while in the mail editor:

At the beginning of a line while in the mail editor, you can use either the **~e** subcommand or the **~v** subcommand as shown in these examples.

Item	Description
~e	Activates the **e** editor or other editor that you define in the `.mailrc` file.
~v	Activates the **vi** editor or other editor that you define in the `.mailrc` file.

This enables you to edit the text of the current message. When you leave the different editor, you return to the mail editor.

Displaying lines of a message while in the mail editor:

Use the **~p** subcommand to display message lines while in the mail editor.

1. The mail program must be installed on your system.
2. To display a message while in the mail editor, you must have started the mail editor. If you need information on this, see "Starting the mail editor" on page 24.

At the beginning of a line while in the mail editor, use the **~p** subcommand as shown in the following example:

Item	Description
~p	The editor displays the contents of the message including the header information for the message. The text scrolls up from the bottom of the display. The end of the message is followed by the mail editor's `(Continue)` prompt.

If the message is larger than one screen and you have not set the page size for your terminal by using the **stty** command, the text scrolls off the top of the screen to the end. To look at the content of large messages, use the mail editor subcommands to view the message with another editor. If you need information on this, see "Message editing" on page 25.

Quitting the mail editor:

To quit the mail editor without sending the message, use the **~q** subcommand or the interrupt key sequence (usually the Alt-Pause or Ctrl-C key sequence).

1. The mail program must be installed on your system.
2. To display a message while in the mail editor, you must have started the mail editor. If you need information on this, see "Starting the mail editor" on page 24.

If you have entered any text, the **mail** command saves the message in the `dead.letter` file.

At the beginning of a line while in the mail editor, you can use the **~q** subcommand as shown in the following example:

Item	Description
~q	Quits the mail editor and the message is not sent. The message is saved in the `dead.letter` file in your home directory, unless you have not entered any text. The system prompt is displayed.

Ctrl-C	To quit the editor using an interrupt key sequence, press the break (Ctrl-C key sequence) or the interrupt (Alt-Pause key sequence). The following message is displayed:
	`(Interrupt -- one more to kill letter)`
	Press break or interrupt again.
	`(Last Interrupt -- letter saved in dead.letter)`
	The message is not sent. The message is saved in the `dead.letter` file in your home directory, unless you have not entered any text. The system prompt is displayed.

Note: When you exit the mail editor without sending the message, the previous content of the `dead.letter` file is replaced with the incomplete message. To retrieve the file, see "Options for adding a file and a specific message within a message."

Options for adding a file and a specific message within a message:

Several requirements must be met before adding a file and specific message within a mail message.

Prerequisites

1. The mail program must be installed on your system.
2. You must know the name and address of the mail recipient.
3. The mail editor must be started.

Including files in a message:

Use the **~r** subcommand to add files to a message.

At the beginning of a line while in the mail editor, you can use the **~r** subcommand as shown in the following example:

Item	Description
~r `schedule`	Where `schedule` is the name of the file to include. In this example, the information in the file `schedule` is included at the current end of the message being written.

Including a specific message within a message:

Use the **~f** or **~m** subcommand to include a specific message within your message.

At the beginning of a new line while in the mail editor, you can use either the **~f** or **~m** subcommand as shown in the following examples:

Item	Description
~f *MessageList*	Appends the indicated message or messages to the end of the current message, but does *not* indent the appended message. Also, use this subcommand to append messages for reference whose margins are too wide to embed with the **~m** subcommand. **Note:** The parameter *MessageList* is a list of integers that refer to valid message numbers in the mailbox or folder being handled by mail. You can enter simple ranges of numbers also. For example: **~f 1-4** Appends messages 1, 2, 3, and 4 to the end of the message being written. These messages are aligned with the left margin (not indented).
~m 2	Appends the indicated message to the end of the current message. The included message is indented one tab character from the normal left margin of the message. In this example, message 2 is appended to the current message.

Item	Description
~m 1 3	Appends message 1 and then message 3 to the end of the message being written, indented one tab from the left margin.

Adding the contents of the dead.letter file into your current message:

Use the **~d** subcommand to add `dead.letter` contents to your message.

At the beginning of a new line while in the mail editor, you can use the **~d** subcommand as shown in the following example:

Item	Description
~d	Retrieves and appends the contents of the `dead.letter` file to the end of the current message. At the `(Continue)` prompt, continue by adding to the message or by sending the message.

Editing the header information:

The header of a message contains routing information and a short statement of the subject. You must specify at least one recipient of the message.

1. The mail program must be installed on your system.
2. Start the mail editor and begin editing a message. For more information, seeStart the mail editor.

The rest of the header information is not required. The information in the header can include the following:

Item	Description
To:	Contains the address or addresses for sending the message.
Subject:	Contains a short summary of the topic of the message.
Cc:	Contains the address or addresses for sending copies of the message. The contents of this field are part of the message sent to all who receive the message.
Bcc:	Contains the address or addresses for sending *blind* copies of the message. This field is *not* included as part of the message sent to all who receive the message.

You can customize the mail program to automatically ask for the information in these fields by putting entries in your `.mailrc` file. For more information, see “Mail program customization options” on page 34.

Setting or resetting the Subject: field:

Use the **~s** subcommand to set the **Subject:** field to a particular phrase or sentence.

Using this subcommand replaces the previous contents (if any) of the **Subject:** field. At the beginning of a new line while in the mail editor, you can use the **~s** subcommand as shown in the following example:

Item	Description

~s `Fishing Trip`	This changes the current **Subject:** field:
	`Subject: Vacation`
	
	To the following:
	`Subject: Fishing Trip`
	Note: You cannot append to the **Subject:** field with this subcommand. Use the **~h** subcommand, as described in “Editing the header information.”

Adding users to the To:, Cc:, and Bcc: fields:

Use the **~t**, **~c**, or the **~b** subcommand to add users to the header fields.

At the beginning of a new line while in the mail editor, you can use the **~t**, **~c** , or **~b** subcommands as shown in the following examples:

Item	Description

~t `geo@austin mel@gtwn`	This changes the current **To:** field:
	`To: mark@austin`
	
	to the following:
	`To: mark@austin geo@austin mel@gtwn`

~c `geo@austin mel@gtwn`	This changes the current **Cc:** field:
	`Cc: mark@austin amy`
	
	to the following:
	`Cc: mark@austin amy geo@austin mel@gtwn`

~b `geo@austin mel@gtwn`	This changes the current **Bcc:** field:
	`Bcc: mark@austin`
	
	to the following:
	`Bcc: mark@austin geo@austin mel@gtwn`

> **Note:** You cannot use the **~t**, **~c**, or **~b** subcommands to change or delete the contents of the **To:**, **Cc:**, and **Bcc:** fields. Use the **~h** subcommand, as described in "Editing the header information" on page 28.

Message reformats in the mail editor:

After typing the message and before sending it, you can reformat the message to improve its appearance by using the **fmt** shell program.

Before reformatting a message, make sure the following conditions are true:

1. The mail program must be installed on your system.
2. The **fmt** command must be installed on your system.

At the beginning of a new line while in the mail editor, you can use the **fmt** command as shown in the following example:

Item	Description
~| fmt	Changes the appearance of the message by re-flowing the information for each paragraph within defined margins (a blank line must separate each paragraph). The pipe (|) subcommand pipes the message to standard input of the command and replaces the message with the standard output from that command.

> **Attention:** Do not use the **fmt** command if the message contains embedded messages or pre-formatted information from external files. The **fmt** command reformats the header information in embedded messages and may change the format of pre-formatted information. Instead, use the **~e** or **~v** subcommand to enter a full-screen editor and reformat the message.

Checking for misspelling in the mail editor:

The **spell** command checks the spelling in your message.

Before checking a message for misspellings, make sure the following conditions are true:

1. The mail program must be installed on your system.
2. The text-formatting programs must be installed on your system.

Use the **spell** command to check your message for misspelled words, from the mail editor:

1. Write the message to a temporary file. For example, to write the message to the `checkit` file, type:
   ```
   ~w checkit
   ```
2. Run the **spell** command using the temporary file as input. Type:
   ```
   ~! spell checkit
   ```
   In this example, the exclamation point (!) is the subcommand that starts a shell, runs a command, and returns to the mailbox. The **spell** command responds with a list of words that are not in its list of known words, followed by an exclamation point (!) to indicate that you have returned to the mail program.
3. Examine the list of words. Determine if you need to use an editor to make corrections.
4. Type the following to delete the temporary file:
   ```
   ~! rm checkit
   ```

Sending mail:

Use this procedure to send a message after you have created it.

- The mail program must be installed on your system.
- You must know the name and address of the mail recipient.

1. Enter the **mail** command on the command line, followed by the name and address of the recipient (or recipients) of the message. For example:
   ```
   >mail jan@brown
   ```
   The system responds with:
   ```
   Subject:
   ```
2. Type the subject of the message. For example:
   ```
   Subject: Dept Meeting
   ```
   and press Enter. You can now type the body of the text.
3. Type your message. For example:
   ```
   There will be a short department meeting this afternoon
   in my office. Please plan on attending.
   ```
4. To send a message you have typed with the mail editor, press the end-of-text character, which is usually the Ctrl-D key sequence or a period (`.`), while at the beginning of a new line in the message.
   The system displays the **Cc:** field:
   ```
   Cc:
   ```
5. Type the names and addresses of those users who should receive copies of the message. For example:
   ```
   Cc: karen@hobo cliff@cross
   ```

   Note: If you do not want to send copies, press Enter without typing.

When you press the Enter key, the message is delivered to the specified address.

> **Note:** If you enter an address unknown to the system, or not defined in an alias or distribution list, the system responds with the login name followed by an error message: `[user ID]... User unknown`.

Replying to mail:

At the mailbox prompt, you can use the **r** and **R** subcommands to reply to mail as shown in the following examples.

1. The mail program must be installed on your system.
2. There must be mail in your system mailbox.

Item	Description
r	Creates a new message that is addressed to the sender of the selected message and copied to the people in the **Cc:** field (if any). The **Subject:** field of the new message refers to the selected message. The default value of the **r** subcommand is the current message. This default can be overridden by typing the message number after the **r**.
R	Starts a reply only to the sender of the message. The default value of the **R** subcommand is the current message.

R 4	Starts a reply only to the sender of the message. You can override the current message by typing the message number after the **R**. This example starts a reply to message 4. The system responds with a message similar to the following:
	`To: karen@thor`
	`Subject: Re: Department Meeting`
	
	You then can type your response:
	`I'll be there.`
	
	When you finish typing the text, press either the period (**.**) or the Ctrl-D key sequence to send the message. After the reply is sent, you are returned to the mailbox prompt.

Creating a new message while in the mailbox:

At the mailbox prompt, you can use the **m** subcommand as shown in the following example to create new messages.

Item	Description
m *Address*	The *Address* parameter is any correct user address. This subcommand starts the mail editor and enables you to create a new message while in the mailbox. When you send the message, you will be returned to the mailbox prompt.

Forwarding mail:

While reading your mail, you might want to forward a specific note to another user.

1. The mail program must be installed on your system.
2. If forwarding a selected message, start the mail facility with the **mail** command. Make a note of the number of the mail message that you wish to forward.

This task can be accomplished by using the **~f** and **~m** subcommands.

If you are going to be away from your normal network address, you can have your mail sent to another network address by creating the **.forward** file. See ".forward files" on page 32. The new address can be any valid mail address on your network or a network connected to yours. It can be the address of a coworker who will handle your messages while you are away. When you choose to forward your network mail, you do not receive a copy of any incoming mail in your mailbox. All mail is forwarded directly to the address or addresses that you have specified.

Forwarding selected messages from within the mailbox:

Use this procedure to forward specific mail messages within the mailbox.

To forward specific mail messages:

1. Create a new message by using the **m** subcommand and specify a recipient by typing the following at the mailbox prompt:

   `m User@Host`

   where *User* refers to the login name of another user, and *Host* is the name of the user's system. If the user is on your system, you can omit the `@Host` part of the address.
2. Type a subject name at the **Subject:** prompt.
3. To specify the number of the mail message to be forwarded, type:

```
~f MessageNumber
```

OR

```
~m MessageNumber
```

MessageNumber identifies the piece of mail to forward.

The **mail** command displays a message similar to the following:

```
Interpolating: 1
(continue)
```

4. To exit mail, type a period (`.`) on a blank line. At the **Cc:** prompt, type any additional names to whom you wish to forward a mail message.

Forwarding all mail:

Use this procedure to forward all of your mail to another person.

To forward all your mail to another person:

1. Enter the **cd** command with no parameters to ensure you are in your home directory. For example, type the following for login name `mary`:

   ```
   cd
   pwd
   ```

   The system responds with:

   ```
   /home/mary
   ```

2. Create a `.forward` file in your home directory. See ".forward files."

   Note: You will not receive any mail until you delete the `.forward` file.

.forward files:

The `.forward` file contains the network address or addresses that will receive your forwarded network mail.

Addresses must take the form of *User@Host*. *User* refers to the login name of another user, and *Host* is the name of the user's system. If the user is on your system, you can omit the `@`*Host* part of the address. You can use the **cat** command to create a `.forward` file as follows:

```
cat > .forward
mark
joe@saturn
[END OF FILE]
```

`[END OF FILE]` represents the end of file character, which is the Ctrl-D key sequence on most terminals. This must be typed on a blank line.

The `.forward` file contains the addresses of the users you want your mail forwarded to. Your mail will be forwarded to `mark` on your local system, and to `joe` on system `saturn`.

This file must contain valid addresses. If it is a null file (zero length), your mail is not forwarded and is stored in your mailbox.

Note: You will not receive any mail until you delete the `.forward` file.

Canceling forwarded mail:

To stop forwarding mail, delete the `.forward` file as follows.

Use the **rm** command to remove the `.forward` file from your home directory:

```
rm .forward
```

Sending a vacation message notice:

Use this procedure to prepare and send a vacation message notice.

The mail program must be installed on your system.

1. Initialize the vacation message in your $HOME (login) directory by typing:

   ```
   vacation -I
   ```

   This creates a .vacation.dir file and a .vacation.pag file where names of the people who send messages are kept.
2. Modify the .forward file. For example, carl types the following statement in the .forward file:

   ```
   carl, |"/usr/bin/vacation carl"
   ```

   The first carl entry is the user name to which mail is forwarded. The second carl entry is the user name of the sender of the vacation message. The sender of the mail message receives one vacation message from carl per week, regardless of how many messages are sent to carl from the sender. If you have your mail forwarded to someone else, the mail message from the sender is forwarded to the person defined in your .forward file.

   Use the **-f** flag to change the frequency intervals at which the message is sent. For example, carl types the following statement in the .forward file:

   ```
   carl, |"/usr/bin/vacation -f10d carl"
   ```

   The sender of mail messages receives one vacation message from carl every ten days, regardless of how many messages are sent to carl from the sender.
3. To send a message to each person who sends you mail, create the file $HOME/.vacation.msg and add your message to this file. The following is an example of a vacation message:

   ```
   From: carl@odin.austin (Carl Jones)
   Subject: I am on vacation.
   I am on vacation until October 1. If you have something urgent,
   please contact Jim Terry <terry@zeus.valhalla>.
     --carl
   ```

   The sender receives the message that is in the $HOME/.vacation.msg file, or if the file does not exist, the sender receives the default message found in the /usr/share/lib/vacation.def file. If neither of these files exists, no automatic replies are sent to the sender of the mail message, and no error message is generated.

To cancel the vacation message, remove the .forward file, .vacation.dir file, .vacation.pag file, and .vacation.msg file from your $HOME (login) directory as follows:

```
rm .forward .vacation.dir .vacation.pag .vacation.msg
```

Sending and receiving secret mail:

To send secret mail, at the system command line prompt, use the **xsend** command in the way shown in the following example.

1. The mail program must be installed on your system.
2. A password must have been set up using the **enroll** command.

Item	Description
xsend `barbara`	In this example, secret mail is being addressed to the login name `barbara`. When you press Enter, a single line editor is used to type the text of the message. When you are finished typing your message, press the Ctrl-D key sequence or a period (.) to exit the mail editor and send the message. The **xsend** command encrypts the message before it is sent.

1. To receive secret mail, at your system command line prompt, type:

   ```
   mail
   ```

   The system displays the list of messages in your system mailbox. The secret mail program sends you a notification that you have received secret mail. The message line will be similar to the following:

   ```
   Mail [5.2 UCB] Type ? for help.
   "/usr/spool/mail/linda": 4 messages 4 new
   >N 1 robert Wed Apr 14 15:23 4/182 "secret mail from robert@Zeus"
   ```

   The message text directs you to read your secret mail on your host using the **xget** command.
2. At the system command line prompt, type:

   ```
   xget
   ```

   You are prompted for the password that was previously set up using the **enroll** command. After you type your password, the **xget** command prompt is displayed, followed by a listing of any secret mail. The mail program is used to display any secret mail. You must enter the **q** subcommand if you want to leave read and unread messages in the secret mailbox and prevent the **xget** command from deleting the messages.

Mail help information

You can obtain help information on using the mail program by using the **?**, **man**, or **info** commands.

Item	Description
To get help in the Mailbox	Enter `?` or `help` at the mailbox prompt. The **?** and **help** subcommands display a summary of common mailbox subcommands. You can display a list of all mailbox subcommands (with no summary) by entering the **(l)**ist subcommand.
To get help in the Mail Editor	Type ~? at the mail editor prompt. The **~?** subcommand displays a summary of common mail editor subcommands.
To get help in secret mail	Type ? at the mail editor prompt. The **?** subcommand displays a summary of common secret mail subcommands.
To get help using manual pages	Type `man mail` at the system command line prompt. In this example, `mail` is the command name being searched. The system will provide you with ASCII documentation about the **mail** command. At the continuation marker (:), press Enter to view the rest of the document. The **man** command provides information in ASCII form for reference topics on commands, subroutines, and files.

Mail program customization options

Commands and options in the `.mailrc` and `/usr/share/lib/Mail.rc` files can be customized to fit your personal mailing needs.

See “Mail enabling and disabling options” for information about mail options.

Characteristics of a mail session that you can customize include:

- **Prompts for the subject of a message.** When you enter the **mail** command, the program asks you to complete a **Subject:** field. When this prompt is displayed, you can fill in a summary of the subject matter of the message. That summary is included at the start of the message when it is read by the recipient. See “Subject: and Carbon Copy (Cc:) field prompts” on page 36.
- **Prompts for users to get a copy of a message.** You can customize the `.mailrc` file so that when you send a message, the mail program prompts you for the names of other users who should receive copies of the message. See “Subject: and Carbon Copy (Cc:) field prompts” on page 36.
- **Aliases or distribution lists.** If you send mail on a large network or often send the same message to a large number of people, typing long addresses for each receiver can become tedious. To simplify this process, create an alias or a distribution list in your `.mailrc` file. An *alias* is a name you define that can be used in place of a single user address. A *distribution list* is a name you define that can be used in place of a group of user addresses. See “Aliases and distribution lists” on page 37.
- **Number of lines displayed when reading messages.** You can change the number of lines of message headers or of message text that scroll across the screen. See “Changes to the number of message headers or message text lines displayed in the mail program” on page 38.
- **Information listed in messages.** You can turn off message headers, such as the machine-set `message-id` field. See “Displaying information in a message” on page 39.
- **Folder directory for storing messages.** You can create a special directory for storing messages. You can use the shorthand plus sign (+) subcommand to designate that directory when storing messages or looking at folders. See “Creating default folders to store messages” on page 41.
- **Log file for recording outgoing messages.** You can instruct the **mail** program to record all your outgoing messages in a file or a subdirectory in your home directory. See “Creating default folders to store messages” on page 41.
- **Editors for typing messages**. In addition to the mail editor, you can choose two different editors to edit messages. See “Text editors for typing messages” on page 41.

For more information on customizing the mail program, see the following topics.

Mail enabling and disabling options:

Options can be either binary or valued.

Binary options are either **set** or **unset**, while valued options can be **set** to a specific value.

> **Note:** The form **unset** *option* is equivalent to **set no** *option*.

Use the **pg** command to view the `/usr/share/lib/Mail.rc` file. The contents of the `/usr/share/lib/Mail.rc` file define the configuration of the mail program. Alter the system configuration for your mail program by creating a `$HOME/.mailrc` file. When you run the **mail** command, subcommands in the **.mailrc** file override similar subcommands in the `/usr/share/lib/Mail.rc` file. The `.mailrc` options can be customized and are valid each time you use the mail program.

To execute mail commands that are stored in a file, use the **source** subcommand.

`Prerequisites`

The mail program must be installed on your system.

Enabling mail options:

These mailbox subcommands are most commonly used to alter the characteristics of a mail session.

Item	Description
set	Enables mail options.
source	Enables mail options that are stored in a file. When reading mail, you can issue this subcommand at the mailbox prompt: `source PathName` where *PathName* is the path and file containing the mail commands. Commands in this file override the previous settings of any similar commands for the duration of the current session. You can also alter the characteristics of the current mail session by typing commands at the mailbox prompt.

You can set these options while in the mailbox or by making entries in the `.mailrc` file.

Viewing enabled mail options:

When reading your mail, enter the **set** subcommand without any arguments to list all of the enabled `.mailrc` options.

In this list, you can also see if a folder directory is selected and if a log file is set up to record outgoing messages.

At the mailbox prompt, type:

```
set
```

A message similar to the following is displayed:

```
ask
metoo
toplines 10
```

In this example, two binary options are enabled: **ask** and **metoo**. There is no **askcc** entry in the list. This indicates that the **askcc** option is not enabled. The **toplines** option has been assigned the value `10`. The **ask**, **metoo**, **askcc**, and **toplines** options are described in .mailrc File Format section of the *Files Reference*.

Disabling mail options:

These mailbox subcommands are most commonly used to alter the characteristics of a mail session.

Item	Description
unset	Disables mail options.
unalias	Deletes the specified alias names.
ignore	Suppresses message header fields.

You can set these options while in the mailbox or by making entries in the `.mailrc` file.

> **Note:** The form **unset** *option* is equivalent to **set no** *option*.

Subject: and Carbon Copy (Cc:) field prompts:

When the **Subject:** and **Cc:** field prompts are edited, the following prerequisite must be met.

```
Prerequisites
```

The mail program must be installed on your system.

Enabling or disabling Subject: field prompting:

Use the **set** and **unset** commands to enable and disable the **Subject:** field.

You can enable or disable the **Subject:** field in the way shown in the following examples:

Item	Description
set ask	**Subject:** field prompting is enabled by editing the `.mailrc` file `ask` option.
unset ask	**Subject:** field prompting is disabled by editing the `.mailrc` file `ask` option.

Enabling or disabling Carbon Copy (Cc:) field prompting:

Use the **set** and **unset** commands to enable and disable the **Cc:** field.

You can enable or disable the **Cc:** field in the way shown in the following examples:

Item	Description
set askcc	Carbon copy (**Cc:**) field prompting is enabled by editing the `.mailrc` file **askcc** option.
unset askcc	Carbon copy (**Cc:**) field prompting is disabled by editing the `.mailrc` file **askcc** option.

Aliases and distribution lists:

By creating aliases and distribution lists, you can manage the addressees and addresses that you commonly use with greater ease.

Before creating an alias or distribution list, make sure that the following conditions are true:

1. The mail program must be installed on your system.
2. You must know the names and addresses of users you want to include in your alias or distribution list.

You can create an alias or distribution list in the following ways:

Item	Description

alias	`kath kathleen@gtwn`
	
	In this example, the alias `kath` has been listed for user `kathleen` at address `gtwn`. After you have added this line to your `$HOME/.mailrc` file, to send a message to Kathleen, type the following at the command line prompt:
	`mail kath`
	
	You are now able to send mail to Kathleen using the `kath` alias.

alias	`dept dee@merlin anne@anchor jerry@zeus bill carl`
	
	After you have added this line to your `$HOME/.mailrc` file, to send a message to your department, type the following at the command line prompt:
	`mail dept`
	
	The message you now create and send will go to `dee` on system `merlin`, `anne` on system `anchor`, `jerry` on system `zeus`, and to `bill` and `carl` on the local system.

To list the aliases and distribution lists, type the following at the mailbox prompt:

```
alias
```

OR

```
a
```

A list of aliases and distribution lists is displayed.

Changes to the number of message headers or message text lines displayed in the mail program:

By changing the `.mailrc` file, you can customize the ability to scroll through mailbox lists or through actual messages.

In order to make these changes, the mail program must be installed on your system.

Changing the number of displayed lines of the message list:

Each message in your mailbox has a one-line heading in the message list. If you have more than 24 messages, the first headings from the message list scroll past the top of your screen. The **set screen** option controls how many lines of the list are displayed at a time.

To change the number of lines of the message list that are displayed at a time, in your **$HOME/.mailrc** file, type:

```
set screen=20
```

In this example, the system will display 20 message headers at a time. Use either the **h** or **z** subcommand to view additional groups of headers. You can also type this subcommand at the mailbox prompt.

Changing the number of displayed lines in a long message:

If you display a message with more than 24 lines, the first lines of the message scroll past the top of the screen. You can use the **pg** command from within mail to browse through long messages if you have included the **set crt** option in the `.mailrc` file.

The **set crt** option controls how many lines a message must contain before the **pg** command is started.

For example, if you use the **t** subcommand to read a long message, only one screen (or page) is displayed. The page is followed by a colon prompt to let you know there are more pages. Press the Enter key to display the next page of the message. After the last page of the message is displayed, there is a prompt similar to the following:

```
EOF:
```

At the prompt, you can enter any valid **pg** subcommand. You can display previous pages, search the message for character strings, or quit reading the message and return to the mailbox prompt.

The **set crt** option is entered in the `.mailrc` file as:

set crt=*Lines*

For example:

```
set crt=20
```

specifies that a message must be 20 lines before the **pg** command is started. The **pg** command is started when you read messages with more than 20 lines.

Changing the number of displayed lines at the top of a message:

The **top** subcommand enables you to scan through a message without reading the entire message.

You control how many lines of a message are displayed by setting the **toplines** option as follows:

set toplines=*Lines*

In this subcommand, the *Lines* variable is the number of lines, starting from the top and including all header fields, that are displayed with the **top** subcommand.

For example, if user Amy has the following line in her .mailrc file:

```
set toplines=10
```

when Amy runs the **mail** command to read her new messages, the following text is displayed:

```
Mail Type ? for help.
"/usr/mail/amy": 2 messages 2 new>
N 1 george  Wed Jan 6 9:47  11/257 "Dept Meeting"
N 2 mark    Wed Jan 6 12:59 17/445 "Project Planner"
```

When Amy uses the **top** subcommand to browse through her messages, the following partial message is displayed:

```
top 1
Message 1:
From george Wed Jan 6 9:47 CST 1988
Received: by zeus
    id AA00549; Wed, 6 Jan 88 9:47:46 CST
Date: Wed, 6 Jan 88 9:47:46 CST
From: george@zeus
Message-Id: <8709111757.AA00178>
To: amy@zeus
Subject: Dept Meeting
Please plan to attend the department meeting on Friday
at 1:30 in the planning conference room. We will be
```

The message is partially displayed because **toplines** is set to 10. Only lines 1 (the **Received:** field) through 10 (the second line of the message body) are displayed. The first line, From george Wed Jan 6 9:47 CST 1988, is always present and does not count in the **toplines** option.

Displaying information in a message:

By changing the .mailrc file, you can control what header information is displayed in a message.

Some header information might be already turned off. Examine your /usr/share/lib/Mail.rc file for ignored header fields.

```
Prerequisites
```

The mail program must be installed on your system.

Preventing the Date, From, and To headers from being displayed:

Every message has several header fields at the top. These header fields are displayed when you read a message. You can use the **ignore** subcommand to suppress the display of header fields when a message is read.

The format for the **ignore** subcommand is:

```
ignore [FieldList]
```

The *FieldList* can consist of one or more field names that you want to ignore when you display a message. For example, if user Amy includes the following line in her .mailrc file:

```
ignore date from to
```

and the file /usr/share/lib/Mail.rc has the line:

```
ignore received message-id
```

the result of using the **t** subcommand is:

```
t 1
Message 1:
From george Wed Jan 6 9:47 CST 1988
Subject: Dept Meeting
Please plan to attend the department meeting on Friday
at 1:30 in the planning conference room. We will be
discussing the new procedures for using the project
planning program developed by our department.
```

The **Received:**, **Date:**, **From:**, **Message-Id:**, and **To:** fields are not displayed. To display these fields, use a **T** or **P** subcommand or the **top** subcommand.

> **Note:** In the example, the **From** line is displayed. This is not the same as the **From:** field that has been listed in the *FieldList* for the **ignore** subcommand.

Listing the ignored header fields:

Use the **ignore** subcommand to list ignored header fields.

To get a list of the currently ignored header fields, at the mailbox prompt, type:

```
ignore
```

A list of all currently ignored headers is displayed. For example:

```
mail-from
message-id
return-path
```

Resetting the header fields:

To reset the header fields, use the **retain** subcommand.

For example:

```
retain date
```

Listing the retained header fields:

Use the **retain** subcommand to list retained header fields.

To see which header fields are currently retained, enter the **retain** subcommand without a header field parameter.

Preventing the banner from displaying:

The mail banner is the line at the top of the list of messages that shows the name of the mail program when you issue the **mail** command.

It is similar to the following line:

```
Mail [5.2 UCB] [Workstation 3.1] Type ? for help.
```

To prevent the banner from displaying when you start the mail program, add the following line to your `$HOME/.mailrc` file:

```
set quiet
```

Another option that suppresses the **mail** banner is:

```
set noheader
```

With this option in the `.mailrc` file, the list of messages in your mailbox is not displayed. When you start the **mail** program, the only response is the mailbox prompt. You can get a list of messages by typing the (**h**)eader subcommand.

Combining the delete and print commands:

Use the autoprint option to combine the delete and print subcommands.

After you read a message, you can delete it with the **d** subcommand. You can display the next message with the **p** subcommand. Combine these subcommands by typing the following line in your `.mailrc` file:

```
set autoprint
```

With the **set autoprint** option in the `.mailrc` file, the **d** subcommand deletes the current message and displays the next.

Creating default folders to store messages:

Default folders allow you to store messages.

The mail program must be installed on your system.

Use the following procedure to create a letters mailbox directory to store messages in folders:

1. To check if the **set folder** option has been enabled in the `.mailrc` file, at the mailbox prompt, type:

   ```
   set
   ```

   If the **set folder** option has been enabled, the system responds with the following:

   ```
   folder /home/george/letters
   ```

   In this example, `letters` is the directory in which mail folders will be stored.
2. If the **set folder** option has not been enabled, make a **set folder** entry in the `.mailrc` file:

   ```
   set folder=/home/george/letters
   ```

   In this example, `/home/george` is George's home directory and `letters` is the directory in which mail folders will be stored. The **set folder** option will allow you to use the plus sign (+) shorthand notation to save messages in your `letters` directory.
3. If a `letters` directory does not exist, you must create a `letters` directory in your home directory. From your home directory, at your system command line, type:

   ```
   mkdir letters
   ```

Use the following procedure to keep a record of messages you send to others:

1. Type the following statement in your `.mailrc` file:

   ```
   set record=letters/mailout
   ```
2. If a `letters` directory does not exist, you must create a `letters` directory in your home directory. From your home directory, at your system command line, type:

   ```
   mkdir letters
   ```
3. To read copies of the messages you have sent to others, type:

   ```
   mail -f +mailout
   ```

   In this example, the file `mailout` contains copies of the messages you have sent to others.

Text editors for typing messages:

Use the **set EDITOR=***PathName* option to define the text editor with which you type messages.

The mail program must be installed on your system.

Item	Description
set EDITOR=*PathName*	This option in your `.mailrc` file defines the editor that you activate with the **~e** key sequence. The value of *PathName* must be the full path name to the editor program you want to use. To change to the e editor, while in the mail program, type: `~e` This sequence activates the e editor or other editor that you have defined in the `.mailrc` file. Edit your mail message using this editor.
set VISUAL=*PathName*	This option in your `.mailrc` file defines the editor that you activate with the **~v** key sequence. The value of *PathName* must be the full path name to the editor program that you want to use. The default is `/usr/bin/vi`. To change to the vi editor while in the mail program, type: `~v` This sequence activates the vi editor or other editor that you have defined in the `.mailrc` file. Edit your mail message using this editor.

Mail command subcommands

The **mail** command uses various subcommands that perform different functions.

This topic is used as a reference for the **mail** command and its subcommands.

Commands to execute mail:

Use these system commands to execute mail.

Item	Description
mail	Displays the system mailbox.
mail -f	Displays your personal mailbox (**mbox**).
mail -f +*folder*	Displays a mail folder.
mail*user@address*	Addresses a message to the specified user.

Mailbox subcommands in the mail program:

When the mail program is processing a mailbox, it displays the mailbox prompt to indicate that it is waiting for input.

The mailbox prompt is an ampersand (&) displayed at the beginning of a new line. At the prompt, you can enter any of the mailbox subcommands.

Mail program control subcommands:

Use these subcommands for controlling the mail program.

Item	Description
q	Quits and applies the mailbox subcommands entered this session.
x	Quits and restores the mailbox to original state.
!	Starts a shell, runs a command, and returns to the mailbox.
cd *dir*	Changes directory to `dir` or `$HOME`.

Mail program display subcommands:

Use these subcommands to control mail program displays.

Item	Description
t	Displays the messages in *msg_list* or the current message.
n	Displays the next message.
f *msg_list*	Displays the headings of messages in *msg_list* or of the current message if *msg_list* is not provided.
h *num*	Displays the headings of groups containing message *num*.
top *num*	Displays a partial message.
set	Displays a list of all enabled **.mailrc** options.
ignore	Displays a list of all the ignored header fields.
folder	Displays the number of messages in the current folder along with the path name of the folder.

Message handling:

Use these subcommands to edit, delete, recall, append, or retain messages.

Item	Description
e *num*	Edits the message *num* (default editor is e).
d *msg_list*	Deletes the messages in *msg_list* or the current message.
u *msg_list*	Recalls deleted messages in *msg_list*.
s *msg_list* **+***file*	Appends messages (with headings) to *file*.
w *msg_list* **+***file*	Appends messages (text only) to *file*.
pre *msg_list*	Keeps messages in the system mailbox.

New mail subcommands:

Use these subcommands when creating new mail messages.

Item	Description
m *addrlist*	Creates and sends a new message to the addresses in *addrlist*.
r *msg_list*	Sends a reply to senders and recipients of messages.
R *msg_list*	Sends a reply only to senders of messages.
a	Displays a list of aliases and their addresses.

Mail editor subcommands:

When the mail editor is processed, it displays the mail editor prompt to indicate that it is waiting for input.

At the prompt, you can enter any of the mail editor subcommands.

Mail editor control subcommands:

Use the following subcommands to control the mail editor.

Item	Description
~q	Quits the editor without saving or sending the current message.
~p	Displays the contents of the message buffer.
~: *mcmd*	Runs a mailbox subcommand (*mcmd*).
EOT	Sends message (Ctrl-D on many terminals).
.	Sends the current message.

Add to heading subcommands:

Use these subcommands to add different heading items to messages.

Item	Description
~h	Adds to the **To:**, **Subject**:, **Cc:**, and **Bcc:** fields.
~t *addrlist*	Adds the user addresses in *addrlist* to the **To:** field.
~s *subject*	Sets the **Subject** field to the string specified by *subject*.
~c *addrlist*	Adds the user addresses in *addrlist* to the **Cc:** (carbon copy) field.
~b *addrlist*	Adds the user addresses in *addrlist* to the `Bcc:` (blind carbon copy) field.

Add to message subcommands:

Use these subcommands to add contents to a message.

Item	Description
~d	Appends the contents of `dead.letter` to the message.
~r *filename*	Appends the contents of *filename* to the message.
~f *numlist*	Appends the contents of message numbers *numlist*.
~m*numlist*	Appends and indents the contents of message numbers *numlist*.

Change message subcommands:

Use these subcommands to edit messages.

Item	Description
~e	Edits the message using the e editor (default is e).
~v	Edits the message using the vi editor (default is vi).
~w *filename*	Writes the message to *filename*.
~! *command*	Starts a shell, runs *command*, and returns to the editor.
~| *command*	Pipes the message to standard input of *command* and replaces the message with the standard output from that command.

Secret mail subcommands:

When the secret mail program processes a secret mailbox, it displays the secret mailbox prompt to indicate that it is waiting for input.

The secret mailbox prompt is a question mark (?) displayed at the beginning of a new line. At the prompt, you can enter any of the secret mailbox subcommands.

Secret mail subcommands:

Use the following subcommands to send secret mail.

Item	Description
xsend `barbara`	Addresses a message to the specified user.
xget	Displays the secret mailbox.

Mailbox tasks:

The following subcommands accomplish various mailbox tasks.

Item	Description
q	Quits, leaving unread messages.
n	Deletes the current message and displays the next message.
d	Deletes the current message and displays the next message.
Return key	Deletes the current message and displays the next message.
!	Executes a shell command.
s	Saves the message in the named file or `mbox`.
w	Saves the message in the named file or `mbox`.

Mail management tasks

The mail manager is responsible for these tasks.

1. Configure the `/etc/rc.tcpip` file so that the **sendmail** daemon will be started at system boot time. See "Configuring the /etc/rc.tcpip file to start the sendmail daemon."
2. Customize the configuration file `/etc/mail/sendmail.cf`. The default `/etc/mail/sendmail.cf` file is configured so that both local mail and TCP/IP mail can be delivered. In order to deliver mail through BNU, you must customize the `/etc/mail/sendmail.cf` file. See the sendmail.cf File in *Files Reference* for more information.
3. Define system-wide and domain-wide mail aliases in the `/etc/mail/aliases` file. See "Mail aliases" for more information.
4. Manage the Mail Queue. See "Mail queue" on page 47 for more information.
5. Manage the Mail Log. See "Mail logging" on page 51 for more information.

Configuring the /etc/rc.tcpip file to start the sendmail daemon

To configure the `/etc/rc.tcpip` file so that the **sendmail** daemon will be started at system boot time, use this procedure.

1. Edit the `/etc/rc.tcpip` file with your favorite text editor.
2. Find the line that begins with `start /usr/lib/sendmail`. By default, this line should be uncommented, that is, there is no # (pound sign) at the beginning of the line. However, if it is commented, delete the pound sign.
3. Save the file.

With this change, the system will start the **sendmail** daemon at boot time.

Mail aliases

Aliases map names to address lists using personal, system-wide, and domain-wide alias files.

You can define three types of aliases:

Item	Description
personal	Defined by individual users in the user's `$HOME/.mailrc` file.
local system	Defined by the mail system administrator in the `/etc/mail/aliases` file. These aliases apply to mail handled by the **sendmail** program on the local system. Local system aliases rarely need to be changed.
domainwide	By default, **sendmail** reads `/etc/alias` to resolve aliases. To override the default and use NIS, edit or create `/etc/netsvc.conf` and add the line: `aliases=nis`

/etc/mail/aliases file

The properties, contents, and location of the `/etc/mail/aliases` file are described here.

The `/etc/mail/aliases` file consists of a series of entries in the following format:

Alias: *Name1*, *Name2*, ... *NameX*

where *Alias* can be any alphanumeric string that you choose (not including special characters, such as @ or !). *Name1* through *NameX* is a series of one or more recipient names. The list of names can span one or more lines. Each continued line begins with a space or a tab. Blank lines and lines beginning with a # (pound sign) are comment lines.

The `/etc/mail/aliases` file must contain the following three aliases:

Item	Description
MAILER-DAEMON	The ID of the user who is to receive messages addressed to the mailer daemon. This name is initially assigned to the root user: `MAILER-DAEMON: root`
postmaster	The ID of the user responsible for the operation of the local mail system. The **postmaster** alias defines a single mailbox address that is valid at each system in a network. This address enables users to send inquiries to the **postmaster** alias at any system, without knowing the correct address of any user at that system. This name is initially assigned to the root user: `postmaster: root`
nobody	The ID that is to receive messages directed to programs such as **news** and **msgs**. This name is initially assigned to `/dev/null`: `nobody: /dev/null` To receive these messages, define this alias to be a valid user.

Whenever you change this file, you must recompile it into a database format that the **sendmail** command can use. See "Alias database building" on page 47.

Creating a local alias for mail

Creating local mail aliases allows you to create groups or distribution lists to which mail can be sent.

In this scenario, `geo@medussa`, `mark@zeus`, `ctw@athena`, and `dsf@plato` will be added to the `testers` mail alias. After the `testers` alias is created, glenda@hera will be given ownership of the alias.

After the `testers` alias is added to the `/etc/mail/aliases` file, the aliases database is recompiled using the **sendmail** command. After the database is recompiled, email can be sent to the `testers` alias.

Things to Consider

- The information in this how-to scenario was tested using specific versions of AIX. The results you obtain might vary significantly depending on your version and level of AIX.

Use the following steps to create a local mail alias:

1. Open the `/etc/mail/aliases` file using your favorite text editor.
2. On a blank line, add the alias name, followed by a colon and a list of comma-separated recipients. For example, the following entry defines the `testers` alias:

   ```
   testers: geo@medussa, mark@zeus, ctw@athena, dsf@plato
   ```

3. Create an owner for the alias. If the **sendmail** command is unsuccessful in sending mail to the alias, it sends an error message to the owner.

   Add a line in the `/etc/mail/aliases` to specify the owner. The format for this line is `owner-`*`groupname`*`:` *`owner`*, where *groupname* is the name of the alias and *owner* is the e-mail address of the owner. In this example, `glenda@hera` is made the owner of the `testers` alias:

   ```
   testers: geo@medussa, mark@zeus, ctw@athena, dsf@plato owner-testers: glenda@hera
   ```

4. After the alias is created, run the **sendmail -bi** command to recompile the aliases database. You will need to run this command each time you update your `/etc/mail/aliases` file.

You can now send email to the `testers` alias.

Alias database building

The **sendmail** command does not use directly the alias definitions in the local system `/etc/mail/aliases` file. Instead, the **sendmail** command reads a processed database manager (dbm) version of the `/etc/mail/aliases` file.

You can compile the alias database using one of the following methods:

- Run the `/usr/sbin/sendmail` command using the **-bi** flag.
- Run the **newaliases** command. This command causes the **sendmail** command to read the local system `/etc/mail/aliases` file and create a new file containing the alias database information. This file is in the more efficient Berkeley format:

  `/etc/mail/aliases.db`
- Run the **sendmail** command using the **Rebuild Aliases** flag. This rebuilds the alias database automatically when it is out-of-date. Auto-rebuild can be dangerous on heavily loaded machines with large alias files. If it might take more than the rebuild timeout (normally five minutes) to rebuild the database, there is a chance that several processes will start the rebuild process simultaneously.

Note:

1. If these files do not exist, the **sendmail** command cannot process mail and will generate an error message.
2. If you have multiple alias databases specified, the **-bi** flag rebuilds all the database types it understands (for example, it can rebuild Network Database Management (NDBM) databases but not NIS databases).

The `/etc/netsvc.conf` file contains the ordering of system services. To specify the service ordering of aliases, add the following line:

```
aliases=service, service
```

where `service` can be either `files` or `nis`. For example:

```
aliases=files, nis
```

tells the **sendmail** command to try the local alias file first; and if that fails, try `nis`. If `nis` is defined as a service, it should be running.

For further information on the `/etc/netsvc.conf` file, see *Files Reference*.

Mail queue

The mail queue is a directory that stores data and controls files for mail messages that the **sendmail** command delivers. By default, the mail queue is `/var/spool/mqueue`.

Mail messages might be queued for many reasons.

For example:

1. The **sendmail** command can be configured to process the queue at certain intervals, rather than immediately. If this is so, mail messages must be stored temporarily.
2. If a remote host does not answer a request for a mail connection, the mail system queues the message and tries again later.

Mail queue printing

The contents of the queue can be printed using the **mailq** command (or by specifying the **-bp** flag with the **sendmail** command).

These commands produce a listing of the queue IDs, the sizes of the messages, the dates the messages entered the queue, and the senders and recipients.

Mail queue files

Each message in the queue has a number of files associated with it.

The files are named according to the following conventions:

TypefID

where *ID* is a unique message queue ID, and *Type* is one of the following letters indicating the type of file:

Item	Description
d	The data file containing the message body without the heading information.
q	The queue-control file. This file contains the information necessary to process the job.
t	A temporary file. This file is an image of the q file when it is being rebuilt. It is quickly renamed to the q file.
x	A transcript file that exists during the life of a session and shows everything that happens during that session.

For example, if a message has a queue ID of AA00269, the following files are created and deleted in the mail queue directory while the **sendmail** command tries to deliver the message:

Item	Description
dfAA00269	Data file
qfAA00269	Control file
tfAA00269	Temporary file
xfAA00269	Transcript file

q control file:

The q control file contains a series of lines, each beginning with a code letter.

Item	Description
B	Specifies the body type. The remainder of the line is a text string defining the body type. If this entire field is missing, the body type is 7-bit by default, and no special processing is attempted. Legal values are **7BIT** and **8BITMIME**.
C	Contains the controlling user. For recipient addresses that are a file or a program, **sendmail** performs delivery as the owner of the file or program. The controlling user is set to the owner of the file or program. Recipient addresses that are read from a **.forward** or **:include:** file will also have the controlling user set to the owner of the file. When **sendmail** delivers mail to these recipients, it delivers as the controlling user, then converts back to root.
F	Contains envelope flags. The flags are any combination of **w**, which sets the **EF_WARNING** flag; **r**, which sets the **EF_RESPONSE** flag; **8**, which sets the **EF_HAS8BIT** flag; and **b**, which sets the **EF_DELETE_BCC** flag. Other letters are silently ignored.
H	Contains a heading definition. There can be any number of these lines. The order in which the **H** lines appear determines their order in the final message. These lines use the same syntax as heading definitions in the /etc/mail/sendmail.cf configuration file.
I	Specifies the inode and device information for the df file; this can be used to recover your mail queue after a disk crash.
K	Specifies the time (as seconds) of the last delivery attempt.
M	When a message is put into the queue because an error occurred during a delivery attempt, the nature of the error is stored in the **M** line.
N	Specifies the total number of delivery attempts.
O	Specifies the original message transfer system (MTS) value from the ESMTP. It is used for Delivery Status Notifications only.
P	Contains the priority of the current message. The priority is used to order the queue. Higher numbers mean lower priorities. The priority increases as the message sits in the queue. The initial priority depends on the message class and the size of the message.
Q	Contains the original recipient as specified by the ORCPT= field in an ESMTP transaction. Used exclusively for Delivery Status Notifications. It applies only to the immediately following **R** line.
R	Contains a recipient address. There is one line for each recipient.
S	Contains the sender address. There is only one of these lines.
T	Contains the message creation time used to compute when to time out the message.
V	Specifies the version number of the queue file format used to allow new **sendmail** binaries to read queue files created by older versions. Defaults to version **zero**. Must be the first line of the file, if present.
Z	Specifies the original envelope ID (from the ESMTP transaction). Used for Delivery Status Notifications only.
$	Contains a macro definition. The values of certain macros (**$r** and **$s**) are passed through to the queue run phase.

The q file for a message sent to amy@zeus would look similar to:

```
P217031
T566755281
MDeferred: Connection timed out during user open with zeus
Sgeo
Ramy@zeus
H?P?return-path: <geo>
Hreceived: by george (0.13 (NL support)/0.01)
        id AA00269; Thu, 17 Dec 87 10:01:21 CST
H?D?date: Thu, 17 Dec 87 10:01:21 CST
H?F?From: geo
Hmessage-id: <8712171601.AA00269@george>
HTo: amy@zeus
Hsubject: test
```

Where:

Item	Description
P217031	Priority of the message
T566755281	Submission time in seconds
MDeferred: Connection timed out during user open with zeus	Status message
Sgeo	ID of the sender
Ramy@zeus	ID of the receiver
H *lines*	Header information for the message

Time values in sendmail

To set the message timeout and queue processing interval, you must use a specific format for the time value.

The format of a time value is:

```
-qNumberUnit
```

where *Number* is an integer value and *Unit* is the unit letter. *Unit* can have one of the following values:

Item	Description
s	Seconds
m	Minutes
h	Hours
d	Days
w	Weeks

If *Unit* is not specified, the **sendmail** daemon uses minutes (**m**) as the default. Here are three examples illustrating time-value specification:

```
/usr/sbin/sendmail -q15d
```

This command tells the **sendmail** daemon to process the queue every 15 days.

```
/usr/sbin/sendmail -q15h
```

This command tells the **sendmail** daemon to process the queue every 15 hours.

```
/usr/sbin/sendmail -q15
```

This command tells the **sendmail** daemon to process the queue every 15 minutes.

Clogged mail queues

In some cases, you might find that the queue is clogged for some reason. You can force a queue to run using the **-q** flag (with no value).

You can also use the **-v** flag (verbose) to watch what happens:

```
/usr/sbin/sendmail -q -v
```

You can also limit the jobs to those with a particular queue identifier, sender, or recipient using one of the queue modifiers. For example, **-qRsally** restricts the queue run to jobs that have the string `sally` in one of the recipient addresses. Similarly, **-qS***string* limits the run to particular senders, and **-qI***string* limits it to particular queue identifiers.

Setting the queue processing interval

The value of the **-q** flag when the daemon starts determines the interval at which the **sendmail** daemon processes the mail queue.

The **sendmail** daemon is usually started by the `/etc/rc.tcpip` file, at system startup. The `/etc/rc.tcpip` file contains a variable called the queue processing interval (QPI), which it uses to specify the value of the **-q** flag when it starts the **sendmail** daemon. By default, the value of **qpi** is 30 minutes. To specify a different queue processing interval:

1. Edit the `/etc/rc.tcpip` file with your favorite editor.
2. Find the line that assigns a value to the *qpi* variable, such as:

   ```
   qpi=30m
   ```
3. Change the value assigned to the *qpi* variable to the time value you prefer.

These changes will take effect at the next system restart. If you want the changes to take effect immediately, stop and restart the **sendmail** daemon, specifying the new **-q** flag value. See "Stopping the sendmail daemon" on page 51 and "Starting the sendmail daemon" on page 51 for more information.

Moving the mail queue

When a host goes down for an extended period, many messages routed to (or through) that host might be stored in your mail queue. As a result, the sendmail command spends a long time sorting the queue, severely degrading your system performance. If you move the queue to a temporary place and create a new queue, the old queue can be run later when the host returns to service.

To move the queue to a temporary place and create a new queue:

1. Stop the **sendmail** daemon by following the instructions in "Stopping the sendmail daemon" on page 51.
2. Move the entire queue directory by entering:

   ```
   cd /var/spool
   mv mqueue omqueue
   ```
3. Restart the **sendmail** daemon by following the instructions in "Starting the sendmail daemon" on page 51.
4. Process the old mail queue by entering:

   ```
   /usr/sbin/sendmail -oQ/var/spool/omqueue -q
   ```

   The **-oQ** flag specifies an alternate queue directory. The **-q** flag specifies to run every job in the queue. To get a report about the progress of the operation, use the **-v** flag.

   Note: This operation can take some time.
5. Remove the log files and the temporary directory when the queue is empty by entering:

   ```
   rm /var/spool/omqueue/*
   rmdir /var/spool/omqueue
   ```

Starting the sendmail daemon

There are two commands that start the **sendmail** daemon.

To start the **sendmail** daemon, enter either of the following commands:

```
startsrc -s sendmail -a "-bd -q15"
/usr/lib/sendmail -bd -q15
```

If the **sendmail** daemon is already active when you enter one of these commands, you see the following message on the screen:

```
The sendmail subsystem is already active. Multiple instances are not supported.
```

If the **sendmail** daemon is not already active, then you see a message indicating that the **sendmail** daemon has been started.

Stopping the sendmail daemon

To stop the **sendmail** daemon, run the **stopsrc -s sendmail** command.

If the **sendmail** daemon was not started with the **startsrc** command:

- Find the **sendmail** process ID.
- Enter the **kill** *sendmail_pid* command (where *sendmail_pid* is the process ID of the **sendmail** process).

Mail logging

The **sendmail** command logs mail system activity through the **syslogd** daemon.

The **syslogd** daemon must be configured and running for logging to occur. Specifically, the `/etc/syslog.conf` file should contain the uncommented line:

```
mail.debug          /var/spool/mqueue/log
```

If it does not, use your favorite editor to make this change; be certain that the path name is correct. If you change the `/etc/syslog.conf` file while the **syslogd** daemon is running, refresh the **syslogd** daemon by typing the following command at a command line:

```
refresh -s syslogd
```

If the `/var/spool/mqueue/log` file does not exist, you must create it by typing the following command:

```
touch /var/spool/mqueue/log
```

Messages in the log file appear in the following format:

Each line in the system log consists of a time stamp, the name of the machine that generated it (for logging from several machines over the local area network), the word `sendmail:`, and a message. Most messages are a sequence of *name=value* pairs.

The two most common lines logged when a message is processed are the **receipt** line and the **delivery attempt** line. The **receipt** line logs the receipt of a message; there will be one of these per message. Some fields may be omitted. These message fields are:

Item	Description
from	Specifies the envelope sender address.
size	Specifies the size of the message in bytes.
class	Indicates the class (numeric precedence) of the message.
pri	Specifies the initial message priority (used for queue sorting).
nrcpts	Indicates the number of envelope recipients for this message (after aliasing and forwarding).
proto	Specifies the protocol used to receive the message, for example ESMTP or UNIX-to-UNIX Copy Program (UUCP).
relay	Specifies the machine from which it was received.

The **delivery attempt** line is logged each time there is delivery attempt (so there can be several per message if delivery is deferred or there are multiple recipients). These fields are:

Item	Description
to	Contains a comma-separated list of the recipients to this mailer.
ctladdr	Specifies the *controlling user*, that is, the name of the user whose credentials are used for delivery.
delay	Specifies the total delay between the time this message was received and the time it was delivered.
xdelay	Specifies the amount of time needed in this delivery attempt.
mailer	Specifies the name of the mailer used to deliver to this recipient.
relay	Specifies the name of the host that actually accepted (or rejected) this recipient.
stat	Specifies the delivery status.

Because such a large amount of information can be logged, the log file is arranged as a succession of levels. Beginning at level 1, the lowest level, only very unusual situations are logged. At the highest level, even the insignificant events are logged. As a convention, log levels ten and under the most useful information. Log levels above 64 are reserved for debugging purposes. Levels from 11-64 are reserved for verbose information.

The types of activities that the **sendmail** command puts into the log file are specified by the **L** option in the /etc/mail/sendmail.cf file.

Log management

Because information is continually appended to the end of the log, the file can become very large. Also, error conditions can cause unexpected entries to the mail queue. To keep the mail queue and the log file from growing too large, run the /usr/lib/smdemon.cleanu shell script.

This script forces the **sendmail** command to process the queue and maintains four progressively older copies of log files, named log.0, log.1, log.2, and log.3. Each time the script runs it moves:

- log.2 to log.3
- log.1 to log.2
- log.0 to log.1
- log to log.0

Running this script allows logging to start over with a new file. Run this script either manually or at a specified interval with the **cron** daemon.

Traffic logs

Use the **-X** flag of the **sendmail** command to set traffic logging.

Many **Simple Mail Transfer Protocols (SMTPs)** implementations do not fully implement the protocol. For example, some personal computer-based **SMTPs** do not understand continuation lines in reply codes. These can be very hard to trace. If you suspect such a problem, you can set traffic logging by using the **-X** flag. For example:

```
/usr/sbin/sendmail -X /tmp/traffic -bd
```

This command logs all traffic in the /tmp/traffic file.

Because this command logs a lot of data very quickly, it should never be used during normal operations. After running the command, force the **errant** implementation to send a message to your host. All message traffic in and out of **sendmail**, including the incoming **SMTP** traffic, will be logged in this file.

Using **sendmail**, you can log a dump of the open files and the connection cache by sending it a **SIGUSR1** signal. The results are logged at **LOG_DEBUG** priority.

Mailer statistics logs

The **sendmail** command tracks the volume of mail being handled by each of the mailer programs that interface with it.

Those mailers are defined in the /etc/mail/sendmail.cf file.

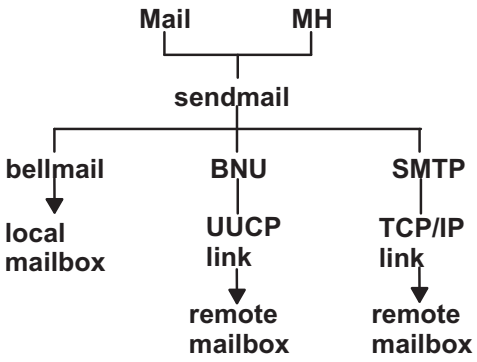


Figure 3. Mailers used by the sendmail command

This illustration is a type of top-down organizational chart with Mail and MH at the top. Branching from them are bellmail, BNU and SMTP. Underneath the previous level are local mailbox, UUCP link, and TCP/IP link respectively. Beneath UUCP link is remote mailbox and under TCP/IP link is remote mailbox.

To start the accumulation of mailer statistics, create the /etc/mail/statistics file by typing the following:

```
touch /etc/mail/statistics
```

If the **sendmail** command encounters errors when trying to record statistics information, the command writes a message through the **syslog** subroutine. These errors do not affect other operations of the **sendmail** command.

The **sendmail** command updates the information in the file each time it processes mail. The size of the file does not grow, but the numbers in the file do. They represent the mail volume since the time you created or reset the /etc/mail/statistics file.

Displaying mailer information

The statistics kept in the /etc/mail/statistics file are in a database format that cannot be read as a text file.

To display the mailer statistics, type the following at a command prompt:

```
/usr/sbin/mailstats
```

This reads the information in the /etc/mail/statistics file, formats it, and writes it to standard output. For information on the output of the **/usr/sbin/mailstats** command, read its description in the *Commands Reference, Volume 3*.

The sendmail Mail Filter API

The sendmail Mail Filter API (referred to as *Milter*) allows third-party programs to access mail messages as they are being processed in order to filter meta-information and content.

sendmail filter requirements

Because filters use threads, filters must be thread-safe. You can configure your filters to ensure their compatibility with threads.

Many operating systems provide support for POSIX threads in the standard C libraries. The compiler flag to link with threading support differs according to the compiler and linker used. If you are unsure of the local flag used, check the Makefile in your appropriate obj.*/libmilter build subdirectory.

Note: Because filters use threads, it might be necessary to alter the process limits on your filter. For example, you might want to use setrlimit to increase the number of open file descriptors if your filter will be busy; otherwise, mail can be rejected.

sendmail filter configurations

Use these guidelines to specify the filters you want while configuring **sendmail**.

Specify filters using the key letter X (for eXternal). In the following example, three filters are specified:

```
Xfilter1, S=local:/var/run/f1.sock, F=R
Xfilter2, S=inet6:999@localhost, F=T, T=C:10m;S:1s;R:1s;E:5m
Xfilter3, S=inet:3333@localhost
```

You can specify filters in your .mc file using the following syntax:

```
INPUT_MAIL_FILTER(`filter1', `S=local:/var/run/f1.sock, F=R')
INPUT_MAIL_FILTER(`filter2', `S=inet6:999@localhost, F=T, T=C:10m;S:1s;R:1s;E:5m')
INPUT_MAIL_FILTER(`filter3', `S=inet:3333@localhost')
```

where filter(*number*) is the name of your filter. The first line of the syntax specifies that the filter attach to a socket in the UNIX domain in the /var/run directory. The second line specifies that the filter use an IPv6 socket on port 999 of the local host. The third line specifies that the filter use an IPv4 socket on port 3333 of the local host.

The F= denotes which of the following flags applies:

Item	Description
R	Rejects the connection if the filter is unavailable.
T	Fails the connection temporarily if the filter is unavailable.

If neither flag is specified, the message is passed through **sendmail** as if the filter were not present.

By specifying a value for T=, you can use the filters to override the default timeouts used by **sendmail**. The T= equate uses the following fields:

Item	Description
C	Timeout for connecting to a filter (if 0, use the system timeout).
S	Timeout for sending information from the MTA to a filter.
R	Timeout for reading replies from the filter.
E	Overall timeout between sending end-of-message notices to the filter and waiting for the final acknowledgement.

As indicated in the preceding example, the separators between each timeout is a semicolon (**;**), and the separator between each equate is a comma (**,**).

The default values for timeouts are as follows:

```
T=C:0m;S:10s;R:10s;E:5m
```

where `s` is seconds and `m` is minutes.

The **InputMailFilters** option determines which filters are used and in what sequence.

Note: If the **InputMailFilters** option is not specified, no filters are used.
The **InputMailFilters** option is set automatically according to the order of the INPUT_MAIL_FILTER commands in your `.mc` file. You can reset this value by setting a value for confINPUT_MAIL_FILTERS in your `.mc` file. For example, if the **InputMailFilters** option is set as:

```
InputMailFilters=filter1, filter2, filter3
```

the three filters will be called in the same order they were specified.

By using MAIL_FILTER() instead of INPUT_MAIL_FILTER() in your `.mc` file, you can define a filter without adding it to the input filter list.

Library control functions

The sendmail filter calls the library control functions to set the **libmilter** parameters before handing over the control to the **libmilter**. The **libmilter** parameters are set by calling the **smfi_main** function. The filter also calls the **smfi_register** function to specifically register its callbacks. Each function returns either the MI_SUCCESS or MI_FAILURE value to indicate the status of the operation. These functions do not communicate with the mail transfer agent (MTA) but alter the library state, which is communicated to the MTA inside the **smfi_main** function.

Table 1. Library control functions

Item	description
smfi_opensocket	**The smfi_opensocket function creates the interface socket.**
smfi_register	The **smfi_register** function registers a filter.
smfi_setconn	The **smfi_setconn** function specifies a socket to use.
smfi_settimeout	The **smfi_settimeout** function sets the timeout.
smfi_setbacklog	The **smfi_setbacklog** function defines the incoming **listen (2)** queue size.
smfi_setdbg	The **smfi_setdbg** function sets the **milter** library debugging (tracing) level.
smfi_stop	The **smfi_stop** function causes an orderly shutdown.
smfi_main	The **smfi_main** function hands over the control to **libmilter**.

smfi_opensocket Function:
Purpose

The **smfi_opensocket** function attempts to create the interface socket mail transfer agents (MTA) that are used to connect to the filter.

Syntax

```
#include <libmilter/mfapi.h>
int smfi_opensocket(
bool rmsocket
);
```

Description

The **smfi_opensocket** function is called only from program mainline, after calling the **smfi_setconn** function and the **smfi_register** function, but before calling the **smfi_main** function. The **smfi_opensocket** function creates the socket specified previously by calling the **smfi_setconn** function which is the interface between MTAs and the filter. The **smfi_opensocket** function allows the calling application to create the socket. If the **smfi_opensocket** function is not called, the **smfi_main** function calls the function implicitly.

Arguments

Table 2. Arguments

Item	Description
rmsocket	A flag indicates if the library must try to remove any existing UNIX domain socket before trying to create a new one.

Return values

The **smfi_opensocket** function returns the MI_FAILURE value in the following cases. Otherwise, the function returns MI_SUCCESS.

- The interface socket could not be created.
- The *rmsocket* value is true, and either the socket could not be examined, or the existing socket could not be removed.
- The **smfi_setconn** function or **smfi_register** function is not called.

Related information

"smfi_register Function"

"smfi_setconn Function" on page 59

smfi_register Function:
Purpose

The **smfi_register** function registers a set of sendmail filter callback functions.

Syntax

```
#include <libmilter/mfapi.h>
int smfi_register(
smfiDesc descr)
);
```

Description

The **smfi_register** function creates a sendmail filter by using the information provided in the **smfiDesc** argument. The **smfi_register** function must be called before **smfi_main** function.

Note: Multiple successful calls to **smfi_register** function within a single process are not allowed. Only a single sendmail filter can be successfully registered. Note, however, that the library cannot check if the restriction is obeyed.

The xxfi_flags field must contain the bitwise or zero or any of the following values, describing the actions the sendmail filter can take.

Table 3. Values

Item	Description
SMFIF_ADDHDRS	The **smfi_addheader** function adds headers.
SMFIF_CHGHDRS	The **smfi_chgheader** function either modifies headers or deletes headers.
SMFIF_CHGBODY	The **smfi_replacebody** function replaces the body during filtering. The filter has significant performance impact if other filters do body filtering after this filter.
SMFIF_ADDRCPT	The **smfi_addrcpt** function adds recipients to the message.
SMFIF_ADDRCPT_PAR	The **smfi_addrcpt_par** function adds recipients including extended simple mail transfer protocol (ESMTP) arguments.
SMFIF_DELRCPT	The **smfi_delrcpt** function removes recipients from the message.
SMFIF_QUARANTINE	The **smfi_quarantine** function quarantines a message.
SMFIF_CHGFROM	The **smfi_chgfrom** function modifies the envelope sender (Mail From).
SMFIF_SETSYMLIST	The **smfi_setsymlist** function sends a set of symbols (macros) that are required.

Arguments

Table 4. Arguments

Item	Description

descr	A filter descriptor of type `smfiDesc` describing the filters functions. The structure has the following members:
	`struct smfiDesc`
	`{`
	` char  *xxfi_name; /* filter name */`
	` int  xxfi_version; /* version code`
	`-- do not change */`
	` unsigned long xxfi_flags; /* flags */`
	
	` /* connection info filter */`
	` sfsistat (*xxfi_connect)(SMFICTX *,`
	`char`
	`*, _SOCK_ADDR *);`
	` /* SMTP HELO command filter */`
	` sfsistat (*xxfi_helo)(SMFICTX *,`
	`char *);`
	` /* envelope sender filter */`
	` sfsistat (*xxfi_envfrom)(SMFICTX *,`
	`char **);`
	` /* envelope recipient filter */`
	` sfsistat (*xxfi_envrcpt)(SMFICTX *,`
	`char **);`
	` /* header filter */`
	` sfsistat (*xxfi_header)(SMFICTX *,`
	`char *,`
	` char *);`
	` /* end of header */`
	` sfsistat (*xxfi_eoh)(SMFICTX *);`
	` /* body block */`
	` sfsistat (*xxfi_body)(SMFICTX *,`
	`unsigned char`
	`*, size_t);`
	` /* end of message */`
	` sfsistat (*xxfi_eom)(SMFICTX *);`
	` /* message aborted */`
	` sfsistat (*xxfi_abort)(SMFICTX *);`
	` /* connection cleanup */`
	` sfsistat (*xxfi_close)(SMFICTX *);`
	
	` /* any unrecognized or unimplemented`
	`command filter */`
	` sfsistat (*xxfi_unknown)(SMFICTX`
	`*, const char *);`
	
	` /* SMTP DATA command filter */`
	` sfsistat (*xxfi_data)(SMFICTX *);`
	
	` /* negotiation callback */`
	` sfsistat (*xxfi_negotiate)(SMFICTX *,`
	`  unsigned long, unsigned long,`
	`unsigned long,`
	` unsigned long,`
	`  unsigned long *, unsigned long *,`
	`unsigned long *,`
	`unsigned long *`
	`);`
	`};`
	
	A NULL value for any callback function indicates that the filter does not process the given type of information, and returns **SMFIS_CONTINUE**.

headerf	The header name is a null-terminated string that is not NULL.
headerv	The header value to be added, can be a non-NULL null-terminated string or an empty string.

Return values

The **smfi_register** function returns the MI_FAILURE value in the following cases. Otherwise, the function returns MI_SUCCESS.

- Memory allocation failed.
- Incompatible version or illegal flags value.

Related information

"smfi_addheader Function" on page 69

"smfi_chgheader Function" on page 71

"smfi_replacebody Function" on page 77

"smfi_addrcpt Function" on page 75

"smfi_addrcpt_par Function" on page 75

"smfi_delrcpt Function" on page 76

"smfi_quarantine Function" on page 79

"smfi_chgfrom Function" on page 74

"smfi_setsymlist Function" on page 93

smfi_setconn Function:
Purpose

The **smfi_setconn** function sets the socket through which this filter can communicate with the **sendmail** command.

Syntax

```
#include <libmilter/mfapi.h>
int smfi_setconn(
char *oconn;
);
```

Description

The **smfi_setconn** function must be called before calling the **smfi_main** function.

The filters must not be run as root when communicating over UNIX or local domain sockets.

The permissions for UNIX or local sockets must be set to `0600` (read or write permission only for owner or group of sockets) or `0660` (read/write permission for the sockets owner and group). These permissions are useful if the **sendmail RunAsUser** option is used.

The permissions for a UNIX or local domain socket are determined by the **umask**, which must be set to `007` or `077`. For operating systems like Solaris operating system that do not use permissions of the socket, place the socket in a protected directory.

Arguments

Table 5. Arguments

Item	Description
oconn	The address of the desired communication socket. The address must be a NULL-terminated string in **proto:address** format: * {unix|local}:/path /to/file -- A named pipe. * inet:port @{hostname|ip-address} -- An IPV4 socket. * inet6:port @{hostname|ip-address} -- An IPV6 socket.

Return values

The **smfi_setconn** function does not fail, if it is an invalid address. However, the **smfi_setconn** function fails to set the socket if there is no memory. The failure is detected only in **smfi_main** function.

Related information

"smfi_main Function" on page 62

smfi_settimeout Function:
Purpose

The **smfi_settimeout** function sets the filters I/O timeout value.

Syntax

```
#include <libmilter/mfapi.h>
int smfi_settimeout((
int otimeout
);
```

Description

The **smfi_settimeout** function is called only from **smfi_main** function. The **smfi_settimeout** function sets the duration (in seconds) for **libmilter** parameter to wait for a mail transfer agent (MTA) communication (read or write) before timing out.

Note: If the **smfi_settimeout** function is not called, the default timeout duration is 7210 seconds.

Arguments

Table 6. Arguments

Item	Description
otimeout	The duration in seconds for the**libmilter** parameter to wait for an MTA before timing out. The *otimeout* value must be greater than zero. If the *otimeout* value is zero, the **libmilter** parameter does not wait for an MTA.

Return values

The **smfi_settimeout** function always returns the MI_SUCCESS value.

Related information

smfi_main

smfi_setbacklog Function:
Purpose

The **smfi_setbacklog** function sets the **listen(2)** backlog value of the filter.

Syntax

```
#include <libmilter/mfapi.h>
int smfi_setbacklog(
int obacklog
);
```

Description

The **smfi_setbacklog** function is called only before calling the **smfi_main** function. The **smfi_setbacklog** function sets the incoming socket backlog, which is used by the **listen(2)** backlog value. If the **smfi_setbacklog** function is not called, the default operating system is used.

Arguments

Table 7. Arguments

Item	Description
obacklog	The number of incoming connections allowed in the listen queue.

Return values

The **smfi_setbacklog** function returns MI_FAILURE value if the *obacklog* argument is set to less than or equal to null.

Related information

"smfi_main Function" on page 62

smfi_setdbg Function:
Purpose

The **smfi_setdbg** function sets the debugging (tracing) level for the **milter** library.

Syntax

```
#include <libmilter/mfapi.h>
int smfi_setdbg(
 int level;
);
```

Description

The **smfi_setdbg** function sets the internal debugging level of the **milter** library to a new level, so that code details can be traced. A level of zero turns off debugging. The higher (more positive) the level, the more detailed the debugging. Six is the current, highest, and useful value.

Arguments

Table 8. Arguments

Item	Description
level	The new debugging level.

Return values

The **smfi_setdbg** function returns the MI_SUCCESS value by default.

smfi_stop Function:
Purpose

The **smfi_stop** function turns off the **milter**. Connections are not accepted after this call.

Syntax

```
#include <libmilter/mfapi.h>
int smfi_stop(void);
);
```

Description

The **smfi_stop** function is called from the Callback functions or error-handling routines at any time. The **smfi_stop** routine does not allow new connections. However, the function does not wait for existing connections (threads) to terminate. This function causes the **smfi_main** function to return to the calling program, which can then exit or warm-restart.

Arguments

Table 9. Arguments

Item	Description
void	This argument does not take any value.

Return values

The **smfi_stop** function returns SMFI_CONTINUE value in the following cases:

- An internal routine causes the **milter** library to stop.
- A routine causes the **milter** library to stop.
- The process that is started cannot be stopped.

Example

```
int ret;
SMFICTX *ctx;
...
ret = smfi_addheader(ctx, "Content-Type",
"multipart/mixed;\n\tboundary=\"foobar\"");
```

Related information

Callback functions

smfi_main Function:
Purpose

The **smfi_main** function hands over the control to the **libmilter** event loop.

Syntax

```
#include <libmilter/mfapi.h>
int smfi_main(
);
```

Description

The **smfi_main** function is called after the initialization of the filter is complete.

Return values

The **smfi_main** function returns the MI_FAILURE value if connection could not be established. Otherwise, the function returns MI_SUCCESS.

Failure occurs for different reasons and the reasons for failure are logged. For example, passing invalid address within the **smfi_setconn** function causes the function to fail.

Related information

"smfi_setconn Function" on page 59

Data access functions

The data access functions are called from within the callback functions that are defined within filters, to access information about the current connection or message.

Table 10. Data access functions

Item	Description
smfi_getsymal	The **smfi_getsymal** function returns the value of a symbol.
smfi_getpriv	The **smfi_getpriv** function fetches the private data pointer.
smfi_setpriv	The **smfi_setpriv** function sets the private data pointer.
smfi_setreply	The **smfi_setreply** function sets the specific reply code to be used.
smfi_setmlreply	The **smfi_setmlreply** function sets the specific multi-line reply to be used.

smfi_getsymval Function:
Purpose

The **smfi_getsymval** function fetches the value of a **sendmail** macro.

Syntax

```
#include <libmilter/mfapi.h>
char* smfi_getsymval(
SMFICTX *ctx,
char *headerf,
char *symname
);
```

Description

The **smfi_getsymval** function is called from any of the **xxfi_* callback** functions to add a header to the message. The macro definition depends on the function that is called.

By default, the following macros are valid:

Table 11. Description

Item	Description
xxfi_connect	**daemon_name, if_name, if_addr, j, _**
xxfi_hello	**tls_version, cipher, cipher_bits, cert_subject, cert_issuer**
xxfi_envfrom	**i, auth_type, auth_authen, auth_ssf, auth_author, mail_mailer, mail_host, mail_addr**
xxfi_envrcpt	**rcpt_mailer, rcpt_host, rcpt_addr**
xxfi_data	None
xxfi_eoh	None
xxfi_eom	**msg_id**

All macros remain in effect from the point they are received until the end of the connection for the **xxfi_connect**, **xxfi_hello** functions.

All macros stay in effect until the end of the message for the **xxfi_envfrom** function, and the **xxfi_eom** function.

All the macros stay in effect for each recipient for **xxfi_envrcpt** function.

The macro list can be changed by using the **confMILTER_MACROS_*** options in the **sendmail.mc**. The scope of such macros is determined by when they are set by the **sendmail** command. For descriptions of macros values, see *Sendmail Installation and Operation Guide*.

Arguments

Table 12. Arguments

Item	Description
ctx	The opaque context structure is maintained in the **libmilter** parameter.
symname	The name of a **sendmail** macro. Single letter macros can optionally be enclosed in braces ("{" and "}"), longer macro names must be enclosed in braces, as in a **sendmail.cf** file.

Return values

The **smfi_getsymval** function returns the value of the given macro as a null-terminated string. Otherwise, the **smfi_getsymval** function returns NULL if the macro is not defined.

Related information

"xxfi_connect callback Function" on page 81

"xxfi_helo callback Function" on page 82

"xxfi_envfrom callback Function" on page 83

"xxfi_envrcpt callback Function" on page 84

"xxfi_data callback Function" on page 84

"xxfi_eoh callback Function" on page 87

"xxfi_eom callback Function" on page 88

smfi_getpriv Function:
Purpose

The **smfi_getpriv** function fetches the connection-specific data pointer for this connection.

Syntax

```
#include <libmilter/mfapi.h>
void* smfi_getpriv(
SMFICTX *ctx
);
```

Description

The **smfi_getpriv** function can be called in any of the **xxfi_* callback** functions.

Arguments

Table 13. Arguments

Item	Description
ctx	The opaque context structure is maintained in the **libmilter** parameter.

Return values

The **smfi_getpriv** function that is stored by returns the private data pointer stored by a call before the **smfi_setpriv** function. Otherwise, the **smfi_setpriv** function returns NULL if the value is not set.

Related information

"smfi_setpriv Function"

smfi_setpriv Function:
Purpose

The **smfi_setpriv** function sets the private data pointer for this connection.

Syntax

```
#include <libmilter/mfapi.h>
int smfi_setpriv
SMFICTX *ctx,
void *privatedata
();
```

Description

The **smfi_setpriv** function is called from any of the **xxfi_* callback** functions, to set the private data pointer for the *ctx*.

Note: There is one private data pointer per connection; multiple calls to the **smfi_setpriv** function with different values cause previous values to be lost. Before a filter terminates, the filter must release the private data and set the pointer to NULL.

Arguments

Table 14. Arguments

Item	Description
ctx	The opaque context structure is maintained in the **libmilter** parameter.
privatedata	The argument points to private data. This value is returned by subsequent calls to the **smfi_getpriv** function by using *ctx*.

Return values

The **smfi_setpriv** function returns the MI_FAILURE value if *ctx* is an invalid context. Otherwise, the function returns MI_SUCCESS.

Related information

"smfi_setpriv Function" on page 65

smfi_setreply Function:
Purpose

The **smfi_setreply** function sets the default simple mail transfer protocol (SMTP) error reply code and accepts only 4XX and 5XX reply codes.

Syntax

```
#include <libmilter/mfapi.h>
int smfi_setreply
SFICTX *ctx,
char *rcode,
char *xcode,
char *message
);
```

Description

The **smfi_setreply** function is called from any of the **xxfi_ callback** functions other than the **xxfi_connect** function. The **smfi_setreply** function sets the SMTP error reply code for the connection. This code is used for subsequent error replies resulting from actions taken by this filter.

The values passed to the **smfi_setreply** function are not checked for standards compliance.

The *message* argument must contain only printable characters. Other characters can lead to undefined behavior. For example, characters like CR or LF causes the call to fail, single '%' characters causes the text to be ignored.

Note: If a '%' string is required in the parameter use '%%' just like for printf(3).

For details about reply codes and their meanings, see RFC 821 or 2821 and RFC 1893 or 2034.

If the *rcode* argument is set as 4XX but, the SMFI_REJECT value is used for the message, the custom reply is not used.

If the *rcode* argument is set as 5XX but, the SMFI_TEMPFAIL value is used for the message, the custom reply is not used.

Note: In the above two cases an error is returned to the **milter** parameter. The **Libmilter** parameter ignores the above reply code.

If the **milter** parameter returns the SMFI_TEMPFAIL value and sets the reply code to 421, the SMTP server terminates the SMTP session with a 421 error code.

Arguments

Table 15. Arguments

Item	Description
ctx	The opaque context structure is maintained in the **libmilter** parameter.
rcode	The three-digit (RFC 821 or 2821) SMTP reply code is a null-terminated string. *rcode* cannot be NULL, and must be a valid 4XX or 5XX reply code.
xcode	The extended (RFC 1893 or 2034) reply code. If *xcode* is NULL, extended code is not used. Otherwise, *xcode* must conform to RFC 1893 or 2034.
message	The text part of the SMTP reply. If message is NULL, an empty message is used.

Return values

The **smfi_setreply** function returns the MI_FAILURE value in the following cases. Otherwise, the function returns MI_SUCCESS.

- The *rcode* or *xcode* argument is invalid.
- A memory-allocation failure occurs.

Related information

"xxfi_connect callback Function" on page 81

smfi_setmlreply Function:
Purpose

The **smfi_setmlreply** function sets the default simple mail transfer protocol (SMTP) error reply code to a multi-line response. The **smfi_setmlreply** function accepts only 4XX and 5XX replies.

Syntax

```
#include <libmilter/mfapi.h>
int smfi_setmlreply(
SMFICTX *ctx,
char *rcode,
char *xcode,
...
);
```

Description

The **smfi_setmlreply** function is called from any of the **xxfi_callback** functions, except for **xxfi_connect** function. The **smfi_setmlreply** function provides SMTP error reply code for the connections mentioned below the *xcode*. The list of arguments must be null-terminated. This code is used for subsequent error replies resulting from actions taken by this filter.

The values passed to the **smfi_setmlreply** function are not checked for standards compliance.

The message parameter must contain only printable characters, other characters can lead to undefined behavior. For example, characters like CR or LF causes the call to fail, single '%' characters causes the text to be ignored.

Note: If a '%' string is required in the `message` parameter use '%%' string similarly like **printf(3)** string is used.

For reply codes and their meanings, see RFC 821 or 2821 and RFC 1893 or 2034.

If the *rcode* is set as `4XX` but, the SMFI_REJECT value is used for the message, the custom reply is not used.

If the *rcode* is set as `5XX` but, the SMFI_TEMPFAIL value is used for the message, the custom reply is not used.

Note: In the above two cases an error is returned to the **milter** parameter, and the **Libmilter** parameter ignores the error.

If the **milter** parameter returns the SMFI_TEMPFAIL value and sets the reply code to `421`, the SMTP server terminates the SMTP session with an `421` error code.

Arguments

Table 16. Arguments

Item	Description
ctx	The opaque context structure is maintained in the **libmilter** parameter.
rcode	The three-digit (RFC 821 or 2821) SMTP reply code, as a null-terminated string. The *rcode* argument cannot be NULL, and must be a valid `4XX` or `5XX` reply code.
xcode	The extended (RFC 1893 or 2034) reply code. If *xcode* is NULL, no extended code is used. Otherwise, *xcode* must conform to RFC 1893 or 2034
...	The remainder of the arguments is single lines of text, up to 32 arguments, which is used as text part of the SMTP reply. The list must be null-terminated.

Return values

The **smfi_setmlreply** function returns the MI_FAILURE value in the following cases. Otherwise, the function returns MI_SUCCESS.

- The *rcode* or *xcode* argument is invalid.
- A memory-allocation failure occurs.
- The text line contains a carriage return or line feed.
- The length of any text line is more than `MAXREPLYLEN(980)`.
- The text replies exceeds more than 32 lines.

Example

```
ret = smfi_setmlreply(ctx, "550", "5.7.0",
"Spammer access rejected",
"Please see our policy at:",
 "http://www.example.com/spampolicy.html",
NULL);
```

The previous example results in the following :

```
550-5.7.0 Spammer access rejected
550-5.7.0 Please see our policy at:
550 5.7.0 http://www.example.com/spampolicy.html
```

Related information

"xxfi_connect callback Function" on page 81

Message modification functions

The message modification functions change the message content and attributes. The functions are called only in by the **xxfi_eom** function. Message modification functions can invoke additional communication with the mail transfer agent (MTA). These functions return either the MI_SUCCESS or MI_FAILURE value to indicate the status of the operation.

Note: Message data (senders, recipients, headers, and body chunks) that are passed to the message modification functions in the parameters is copied and do not need to be preserved (allocated memory can be freed).

To call a message modification function, the filter must set the appropriate flag in the description that is passed to the **smfi_register** function. If the flag is not set, MTA treats the call to the function as a failure of the filter, and terminates the connection.

Note: The status returned by the function indicates whether the message filter was successfully sent to the MTA. The status does not indicate whether the MTA performed the requested operation. For example, the **smfi_header** function, when called with an illegal header name, returns the MI_SUCCESS flag even though the MTA can later decline to add the illegal header.

Table 17. Mod functions

Item	Description	function
smfi_addheader	The **smfi_addheader** function adds a header to the message.	**SMFIF_ADDHDRS**
smfi_chgheader	The **smfi_chgheader** function modifies or deletes a header.	**SMFIF_CHGHDRS**
smfi_insheader	The **smfi_insheader** function inserts a header into the message.	**SMFIF_ADDHDRS**
smfi_chgfrom	The **smfi_chgfrom** function modifies the envelope sender address.	**SMFIF_CHGFROM**
smfi_addrcpt	The **smfi_addrcpt** function adds a recipient to the envelope.	**SMFIF_ADDRCPT**
smfi_addrcpt_par	The **smfi_addrcpt_par** function adds a recipient including extended simple mail transfer protocol (ESMTP) parameter to the envelope.	**SMFIF_ADDRCPT_PAR**
smfi_delrcpt	The **smfi_delrcpt** function deletes a recipient from the envelope.	**SMFIF_DELRCPT**
smfi_replacebody	The **smfi_replacebody** function replaces the body of the message.	**SMFIF_CHGBODY**

smfi_addheader Function:
Purpose

The **smfi_addheader** function adds a header to the current message.

Syntax

```
#include <libmilter/mfapi.h>
int smfi_addheader(
SMFICTX *ctx,
char *headerf,
char *headerv
);
```

Description

The **smfi_addheader** function is called from the **xxfi_eom** function to add a header to the message.

The **smfi_addheader** function does not modify the existing headers of a message.

To modify the current value of a header, use the **smfi_chgheader** function.

A filter that calls the **smfi_addheader** function must set the **SMFIF_ADDHDRS** flag in the `smfiDesc_str` argument. The filter then passes the value to the **smfi_register** function.

The **smfi_addheader** function requires the filter order to be specified. You can view the modifications in the header by using the filters that were created earlier.

The name or the value of the header is not checked for standards compliance. However, each line of the header must be below 998 characters. If you require longer header names, use a multi-line header. If you must create a multi-line header, insert a line feed (`ASCII 0x0a`, or `\n` in C programming language) followed by a whitespace character such as a space (`ASCII 0x20`) or tab (`ASCII 0x09`, or `\t` in C programming language). The line feed cannot be preceded by a carriage return (`ASCII 0x0d`). The mail transfer agent (MTA) adds it automatically. The filter writers responsibility must ensure that no standards are violated.

The MTA adds a leading space to an added header value unless the **SMFIP_HDR_LEADSPC** flag is set, in which case the **milter** parameter must include any desired leading spaces.

Arguments

Table 18. Arguments

Item	Description
ctx	The opaque context structure is maintained in the **libmilter** parameter.
headerf	The header name is a null-terminated string that is not NULL.
headerv	The header value to be added, can be a non-NULL null-terminated string or an empty string.

Return values

The **smfi_addheader** function returns the MI_FAILURE value in the following cases. Otherwise, the function returns MI_SUCCESS.

- The *headerf* or *headerv* argument is NULL.
- Adding headers in the current connection state is invalid.
- Memory allocation fails.
- A network error occurs.
- The **SMFIF_ADDHDRS** flag was not set when the **smfi_register** function was called.

Example

```
int ret;
SMFICTX *ctx;
...
ret = smfi_addheader(ctx, "Content-Type",
"multipart/mixed;\n\tboundary=\"foobar\"");
```

Related information

"xxfi_eom callback Function" on page 88

"smfi_chgheader Function"

"smfi_register Function" on page 56

smfi_chgheader Function:
Purpose

The **smfi_chgheader** function modifies or deletes a message header.

Syntax

```
#include <libmilter/mfapi.h>
int smfi_chgheader(
SMFICTX *ctx,
char *headerf,
mi_int32 hdridx,
char *headerv
);
```

Description

The **smfi_chgheader** function is called from the **xxfi_eom** function to modify a headers value for the current message.

The **smfi_chgheader** function can be used to add new headers. However, it is efficient and safer to use the **smfi_addheader** function.

A filter that calls the **smfi_chgheader** function must set the **SMFIF_CHGHDRS** flag in the `smfiDesc_str` argument. The filter then passes the value to the **smfi_register** function.

The **smfi_chgheader** function requires the filter order to be specified. You can view the modifications in the header by using the filters that were created earlier.

The name or the value of the header is not checked for standards compliance. However, each line of the header must be below 998 characters. If you require longer header names, use a multi-line header. If you must create a multi-line header, insert a line feed (`ASCII 0x0a`, or `\n` in C programming language) followed by a whitespace character such as a space (`ASCII 0x20`) or tab (`ASCII 0x09`, or `\t` in C programming language). The line feed cannot be preceded by a carriage return (`ASCII 0x0d`), mail transfer agent (MTA) adds it automatically. The filter writers responsibility must ensure that no standards are violated.

The MTA adds a leading space to an added header value unless the flag **SMFIP_HDR_LEADSPC** is set, in which case the **milter** parameter must include desired leading spaces itself.

Arguments

Table 19. Arguments

Item	Description
ctx	The opaque context structure is maintained in the **libmilter** parameter.
headerf	The header name is a null-terminated string that is not NULL.
hdridx	The header index value (one-based). A *hdridx* value of 1 modifies the first occurrence of a header named *headerf*. If *hdridx* is greater than the number of times *headerf* appears, a new copy of *headerf* is added.
headerv	The header value to be added, can be a non-NULL null-terminated string or an empty string.

Return values

The **smfi_chgheader** function returns the MI_FAILURE value in the following cases. Otherwise, the function returns MI_SUCCESS.

- The *headerf* argument is NULL.
- Modifying headers in the current connection state is invalid.
- Memory allocation fails.
- A network error occurs.
- The **SMFIF_CHGHDRS** flag was not set when the **smfi_register** function was called.

Example

```
int ret;
SMFICTX *ctx;
...

ret = smfi_chgheader(ctx, "Content-Type", 1,
"multipart/mixed;\n\tboundary=\"foobar\"");
```

Related information

"xxfi_eom callback Function" on page 88

"smfi_addheader Function" on page 69

smfi_insheader Function:
Purpose

The **smfi_insheader** function prepends a header to the current message.

Syntax

```
#include <libmilter/mfapi.h>
int smfi_insheader(
SMFICTX ,
int hdridx,
char *headerf,
char *headerv
);
```

Description

The **smfi_insheader** function is called from the **xxfi_eom** function, to prepend a header to the current message.

The **smfi_insheader** function does not modify the existing headers of a message.

To change the current value of a header, use the **smfi_chgheader** function.

A filter that calls the **smfi_insheader** function must set the **SMFIF_ADDHDRS** flag in the `smfiDesc_str` argument, which is passed in the **smfi_register** function.

The **smfi_insheader** function requires the filter order to be specified. You can view the modifications in the header by using the filters that were created earlier.

A filter receives headers that are sent by the simple mail transfer protocol (SMTP) client and also the headers modified by the earlier filters. The headers inserted by the **sendmail** command and headers inserted by itself are not received. The position to insert the header is dependent on the headers existing in the incoming message and also on the headers that are configured to be added by the **sendmail** command.

For example, the **sendmail** command always adds a **Received: header** at the beginning of the header. On setting the *hdridx* value to `0`, the header is inserted before the **Received: header** parameter. However, the filters later get misled as they receive the added header, but not the **Received: header**, thus making it hard to insert a header at a fixed position.

If the *hdridx* value is greater than the number of headers in the message, the header is appended.

The name or the value of the header is not checked for standards compliance. However, each line of the header must be below 998 characters. If you require longer header names, use a multi-line header. If you must create a multi-line header, insert a line feed (`ASCII 0x0a`, or `\n` in C programming language) followed by a whitespace character such as a space (`ASCII 0x20`) or tab (`ASCII 0x09`, or `\t` in C programming language). The line feed cannot be preceded by a carriage return (`ASCII 0x0d`). The mail transfer agent (MTA) adds the carriage return automatically. The filter writers responsibility must ensure that no standards are violated.

The MTA adds a leading space to an inserted header value unless the **SMFIP_HDR_LEADSPC** flag is set, in which case the **`milter`** parameter must include any desired leading spaces.

Arguments

Table 20. Arguments

Item	Description
ctx	The opaque context structure is maintained in the **`libmilter`** parameter.
headerf	The header name is a null-terminated string that is not NULL.
headerv	The header value to be added, can be a non-NULL null-terminated string or an empty string.

Return values

The **smfi_insheader** function returns the MI_FAILURE value in the following cases, else the function returns MI_SUCCESS.

- The *headerf* or *headerv* argument is NULL.
- Adding headers in the current connection state is invalid.
- The memory allocation fails.
- A network error occurs.
- The **SMFIF_ADDHDRS** flag was not set when **smfi_register** function was called.

Example

```
int ret;
SMFICTX *ctx;
...
ret = smfi_insheader( ctx, 0, "First", "See me?");;
```

Related information

"xxfi_eom callback Function" on page 88

"smfi_register Function" on page 56

"smfi_chgheader Function" on page 71

smfi_chgfrom Function:
Purpose

The **smfi_chgfrom** function modifies the envelope sender (MAIL From) for the current message.

Syntax

```
#include <libmilter/mfapi.h>
int smfi_chgfrom(
SMFICTX *ctx,
const char *mail,
char *args
);
```

Description

The **smfi_chgfrom** function is called from the **xxfi_eom** function, to modify the envelope sender and the MAIL From of the current message.

A filter that calls the **smfi_chgfrom** function must set the **SMFIF_CHGFROM** flag in the `smfiDesc_str` argument. The filter then passes the value to the **smfi_register** function.

All extended simple mail transfer protocol (ESMTP) arguments can be set through the call. But, setting values for some of the arguments like `SIZE` and `BODY` are causing problems. Hence, proper care must be taken when setting the arguments. There is no feedback from mail transfer agent (MTA) to the **milter** parameter if the call is successful.

Arguments

Table 21. Arguments

Item	Description
ctx	The opaque context structure is maintained in the **libmilter** parameter.
mail	The new sender address.
args	Extended simple mail transfer protocol (ESMTP) arguments.

Return values

The **smfi_chgfrom** function returns the MI_FAILURE value in the following cases. Otherwise, the function returns MI_SUCCESS.

- The *mail* argument is NULL.
- Changing the sender in the current connection state is invalid.

- A network error occurs.
- The **SMFIF_CHGFROM** flag was not set when **smfi_register** function was called.

Related information

"xxfi_eom callback Function" on page 88

"smfi_register Function" on page 56

smfi_addrcpt Function:
Purpose

The **smfi_addrcpt** function adds a recipient for the current message.

Syntax

```
#include <libmilter/mfapi.h>
int smfi_addrcpt(
SMFICTX *ctx
char *rcpt
);
```

Description

The **smfi_addrcpt** function is called only from the **xxfi_eom** function, to add a recipient to the message envelope.

Note: The filter that calls the **smfi_addrcpt** function must set the **SMFIF_ADDRCPT** flag in the **smfiDesc_str** structure which is passed to the **smfi_register** function.

Arguments

Table 22. Arguments

Item	Description
ctx	The opaque context structure is maintained in the **libmilter** parameter.
rcpt	The new recipients address.

Return values

The **smfi_addrcpt** function returns the MI_FAILURE value in the following cases. Otherwise, the function returns MI_SUCCESS.

- The *rcpt* argument is NULL.
- Adding recipients in the current connection state is invalid.
- A network error occurs.
- The **SMFIF_ADDRCPT** flag was not set when **smfi_register** function was called.

Related information

"xxfi_eom callback Function" on page 88

"smfi_register Function" on page 56

smfi_addrcpt_par Function:

Purpose

The **smfi_addrcpt_par** function adds a recipient for the current message including extended simple mail transfer protocol (ESMTP) arguments.

Syntax

```
#include <libmilter/mfapi.h>
int smfi_addrcpt_par(
SMFICTX *ctx,
char *rcpt,
char *args
);
```

Description

The **smfi_addrcpt_par** function is called from the **xxfi_eom** function, to add a recipient to the message envelope.

Arguments

Table 23. Arguments

Item	Description
ctx	The opaque context structure is maintained in the **libmilter** parameter.
rcpt	The new recipients address.
args	The new recipients ESMTP parameters.

Return values

The **smfi_addrcpt** function returns the MI_FAILURE value in the following cases. Otherwise, the function returns MI_SUCCESS.

- The *rcpt* argument is NULL.
- Adding recipients in the current connection state is invalid.
- A network error occurs.
- The **SMFIF_ADDRCPT_PAR** flag was not set when **smfi_register** function was called.

Related information

"smfi_addrcpt Function" on page 75

"smfi_register Function" on page 56

smfi_delrcpt Function:
Purpose

The **smfi_delrcpt** function deletes the recipient from the envelope of the current message.

Syntax

```
#include <libmilter/mfapi.h>
int smfi_delrcpt(
 SMFICTX *ctx;
 char *rcpt;
);
```

Description

The **smfi_delrcpt** function is called from the **xxfi_eom** callback function to remove the named recipient from the current messages envelope.

Note: The addresses to be removed must match exactly. For example, an address and its expanded form do not match.

Arguments

Table 24. Arguments

Item	Description
ctx	The opaque context structure is maintained in the **libmilter** parameter.
rcpt	The recipient address to be removed, a non-NULL, null-terminated string.

Return values

The **smfi_delrcpt** function returns the MI_FAILURE value in the following cases. Otherwise the function returns MI_SUCCESS.

- The *rcpt* argument is NULL.
- Deleting the recipients in the current connection state is invalid.
- A network error occurs.
- The SMFIF_DELRCPT flag was not set when the **smfi_register** function was called.

Related information

smfi_register

xxfi_eom

smfi_replacebody Function:
Purpose

The **smfi_replacebody** function replaces the body of the message.

Syntax

```
#include <libmilter/mfapi.h>
int smfi_replacebody(
 SMFICTX *ctx,
 unsigned char *bodyp,
 int bodylen
);
```

Description

The **smfi_replacebody** function replaces the body of the current message. If the function is called more than once, the subsequent calls results in data being appended to the new body. The function might be called more than once.

Because the message body might be large, setting the SMFIF_CHGBODY flag might significantly affect the filter performance.

If a filter sets the SMFIF_CHGBODY flag, but does not call the **smfi_replacebody** function, the original body remains unchanged.

Filter order is important for the **smfi_replacebody** function. New body contents is created by old filters in the new filter files.

Arguments

Table 25. Arguments

Item	Description
ctx	The opaque context structure is maintained in the **libmilter** parameter.
bodyp	A pointer to the start of the new body data, which does not have to be null-terminated. If the *bodyp* is NULL, it is treated as having length == 0. The body data should be in CR or LF form.
bodylen	The number of data bytes pointed to by *bodyp*.

Return values

The **smfi_replacebody** function returns the MI_FAILURE value in the following cases. Otherwise the function returns MI_SUCCESS.

- *bodyp* == NULL and *bodylen* > 0
- Changing the body in the current connection state is invalid.
- A network error occurs.
- The SMFIF_CHGBODY flag was not set when the **smfi_register** function was called.

Related information

smfi_register

Message handling functions

The message handling functions provide special case handling instructions for the **milter** parameter or the mail transfer agent (MTA), without altering the content or status of the message. The Message handling functions can be called only in the **xxfi_eom** function. The **xxfi_eom** function can invoke additional communication with the MTA and return either the MI_SUCCESS or MI_FAILURE value to indicate the status of the operation.

Note: The status returned by the function indicates whether the filters message was successfully sent to the MTA. The status does not indicate whether the MTA performed the requested operation.

Table 26. message handling function

Item	Description
smfi_progress	The **smfi_progress** function reports operation in progress.
smfi_quarantine	The **smfi_quarantine** function quarantines a message.

smfi_progress Function:
Purpose

The **smfi_progress** function reports the progress of the operation.

Syntax

```
#include <libmilter/mfapi.h>
int smfi_progress(
 SMFICTX *ctx;
);
```

Description

The **smfi_progress** function is called from the **xxfi_eom** callback function to notify the mail transfer agent (MTA) that the filter is still working on a message. This function cause the MTA to restart its timeouts.

Arguments

Table 27. Arguments

Item	Description
ctx	The opaque context structure maintained in the **libmilter** parameter.

Return values

The **smfi_progress** function returns the MI_FAILURE value if there is a network error. Otherwise, the function returns MI_SUCCESS.

Related information

xxfi_eom

smfi_quarantine Function:
Purpose

The **smfi_quarantine** function quarantines the message.

Syntax

```
#include <libmilter/mfapi.h>
int smfi_quarantine(
 SMFICTX *ctx;
 char *reason;
);
```

Description

The **smfi_quarantine** function is called from the **xxfi_eom** callback function to quarantine a message by using a given reason.

Arguments

Table 28. Arguments

Item	Description
ctx	The opaque context structure maintained in the **libmilter** parameter.
reason	The quarantine reason, a non-NULL, and non-empty null-terminated string.

Return values

The **smfi_quarantine** function returns the MI_FAILURE value in the following cases. Otherwise, the function returns MI_SUCCESS.

- The *reason* is NULL or empty.
- A network error occurs.
- The `SMFIF_QUARANTINE` flag was not set when the **smfi_register** function was called.

Related information

smfi_register

xxfi_eom

Callback functions

The sendmail filter must implement one or more of the callback functions, which are registered through **smfi_register** function.

Table 29. Call back functions

Item	Description
xxfi_connect	The **xxfi_connect** function is called once at the start of each SMTP connection. The function returns SMFIS_CONTINUE value.
xxfi_hello	The **xxfi_hello** function is called whenever the client sends a HELO/EHLO command.
xxfi_envfrom	The **xxfi_envfrom** function is called at the beginning of a message.
xxfi_envrcpt	The **xxfi_envrcpt** function is called for each recipient.
xxfi_data	The **xxfi_data** function handles the DATA command.
xxfi_unknown	The **xxfi_unknown** function handles unknown simple mail transfer protocol (SMTP) commands.
xxfi_header	The **xxfi_header** function handles a message header.
xxfi_eoh	The **xxfi_eoh** function handles the message headers.
xxfi_body	The **xxfi_body** function handles a chunk of a messages body.
xxfi_eom	The **xxfi_eom** function handles the end of the message.
xxfi_abort	The **xxfi_abort** function handles the messages that are aborted.
xxfi_close	The **xxfi_close** function is called to close the current connection.
xxfi_negotiate	The **xxfi_negotiate** function is called at the start of SMTP connection.

The callback functions must return proper value. If callback functions return any other value than the defined value, it constitutes an error, and the **sendmail** command terminates the connection to the filter.

The **`Milter`** parameter distinguishes between **recipient-**, **message-**, and **connection-oriented** routines:

- The **recipient-oriented** callback functions affect the processing of a single message recipient.
- The **message-oriented** callback functions affect a single message.
- The **connection-oriented** callback functions affect an entire connection (during which multiple messages can be delivered to multiple sets of recipients).
- The **xxfi_envrcpt** function is recipient-oriented. The **xxfi_conect**, **xxfi_hello** and **xxfi_close** functions are connection-oriented. All other callback functions are message-oriented.

Table 30. Callback functions

Item	Description
SMFIS_CONTINUE	Continue processing the current connection, message, or recipient.
SMFIS_REJECT	• For a **connection-oriented** routine, reject this connection; call **xxfi_close**. • For a **message-oriented** routine (except for **xxfi_eom** function, or **xxfi_abort** function), reject this message. • For a **recipient-oriented** routine, reject the current recipient (but continue processing the current message).
SMFIS_DISCARD	• For a **message-** or **recipient-oriented** routine, accept this message, but discard it. • SMFIS_DISCARD must not be returned by a **connection-oriented** routine.
SMFIS_ACCEPT	• For a **connection-oriented** routine, accept this connection without further filter processing; call **xxfi_close** function. • For a **message- or recipient- oriented** routine, accept this message without further filtering.
SMFIS_TEMPFAIL	Return a temporary failure, that is, the corresponding simple mail transfer protocol (SMTP) command returns 4xx status code. • For a **message-oriented** routine (except for **xxfi_envfrom** function), fail for this message. • For a **connection-oriented** routine, fail for this connection; call **xxfi_close** function. • For a **recipient-oriented** routine, fail only for the current recipient; continue processing the message.
SMFIS_SKIP	Skip further callbacks of the same type in this transaction. Currently, this return value is only allowed in **xxfi_body** function. The return value can be used if a **milter** parameter has received sufficient body chunks to make a decision. But if the return value still wants to invoke the message modification functions that are only allowed to be called from **xxfi_eom** function. **Note:** The **milter** parameter must negotiate this behavior with the mail transfer agent (MTA). The **milter** parameter checks whether the protocol action SMFIP_SKIP is available. If the protocol action SMFIP_SKIP is available, the **milter** parameter must request it.
SMFIS_NOREPLY	• Do not send a reply to the MTA. The **milter** parameter must negotiate this behavior with the MTA. The **milter** parameter must check whether the appropriate protocol action SMFIP_NR_* is available. If the protocol action SMFIP_NR_* is available, the **milter** parameter must request it. • If you set the SMFIP_NR_* protocol action for a callback, that callback must always reply with SMFIS_NOREPLY. Using any other reply code is a violation of the application programming interface (API). If in some cases your callback can return another value (due to some resource shortages), you must not set SMFIP_NR_* and you must use SMFIS_CONTINUE as the default return code. Alternatively, you can try to delay reporting the problem to a later callback for which SMFIP_NR_* is not set.

xxfi_connect callback Function: Purpose

The **xxfi_connect** callback function provides the connection information.

Syntax

```
#include <libmilter/mfapi.h>
sfsistat (*xxfi_connect)(
        SMFICTX    *ctx,
        char       *hostname,
        _SOCK_ADDR *hostaddr);
```

Description

The **xxfi_connect** callback function is called once at the start of each simple mail transfer protocol (SMTP) connection and returns the SMFIS_CONTINUE flag.

Note: If an earlier filter rejects the connection in the **xxfi_connect** callback function routine, the **xxfi_connect** callback function of the filter would not be called.

Arguments

Table 31. Arguments

Item	Description
ctx	The opaque context structure is maintained in the **libmilter**.
hostname	The host name of the message sender, as determined by a reverse lookup on the host address. If the reverse lookup fails or if none of the IP addresses of the resolved host name matches the original IP address, the hostname would contain the message sender's IP address, which is enclosed in square brackets (for example. `[a.b.c.d]'). If the simple mail transfer protocol (SMTP) connection is made via stdin, the value is localhost.
hostaddr	The host address, as determined by the **getpeername(2)** call on the SMTP socket. The value is NULL if the type is not supported in the current version or if the SMTP connection is made via **stdin**.

xxfi_helo callback Function:
Purpose

The **xxfi_helo** callback function handles the **HELO or EHLO** command.

Syntax

```
#include <libmilter/mfapi.h>
sfsistat (*xxfi_helo)(
 SMFICTX *ctx,
 char *helohost
);
```

Description

The **xxfi_hello** callback function is called whenever the client sends a **HELO or EHLO** command and returns the SMFIS_CONTINUE flag. The callback might, therefore be called several times or even not called. Some restrictions can be imposed by the mail transfer agent (MTA) configuration.

Arguments

Table 32. Arguments

Item	Description
ctx	The opaque context structure is maintained in the **libmilter** parameter.
helohost	The value passed to **HELO or EHLO** command, should be the domain name of the sending host

xxfi_envfrom callback Function: Purpose

The **xxfi_envfrom** callback function handles the **MAIL** (envelope sender) command.

Syntax

```
#include <libmilter/mfapi.h>
sfsistat (*xxfi_envfrom)(
 SMFICTX *ctx,
 char **argv
);
```

Description

The **xxfi_envfrom** callback function is called when the client uses the **DATA** command and returns the SMFIS_CONTINUE flag.

Note: For more details on ESMTP responses, see RFC 1869.

Arguments

Table 33. Arguments

Item	Description
ctx	The opaque context structure is maintained in the **libmilter** parameter.
argv	The null-terminated SMTP command arguments; `argv[0]` is guaranteed to be the sender address. Later arguments are the extended simple mail transfer protocol (ESMTP) arguments.

Return values

Table 34. Return Values

Item	Description
SMFIS_TEMPFAIL	The sender and the message are rejected with a temporary error; a new sender (and hence a new message) might later be specified and the **xxfi_abort** callback function is not called.
SMFIS_REJECT	The sender and the message are rejected; a new sender and message might be specified and the **xxfi_abort** callback function is not called.
SMFIS_DISCARD	The message is accepted and discarded, and the **xxfi_abort** callback function is not called.
SMFIS_ACCEPT	The message is accepted and the **xxfi_abort** callback function is not called.

Related information

xxfi_abort

xxfi_envrcpt callback Function:
Purpose

The **xxfi_envrcpt** callback function handles the envelope **RCPT** command.

Syntax

```
#include <libmilter/mfapi.h>
sfsistat (*xxfi_envrcpt)(
 SMFICTX *ctx,
 char **argv
);
```

Description

The **xxfi_envrcpt** callback function is called once per recipient, and one or more times per message immediately after the **xxfi_envfrom** callback function and returns the SMFIS_CONTINUE flag.

Note: For more details on extended simple mail transfer protocol (ESMTP) responses, see RFC 1869.

Arguments

Table 35. Arguments

Item	Description
ctx	The opaque context structure is maintained in the **libmilter** parameter.
argv	The null-terminated SMTP command arguments; `argv[0]` is guaranteed to be the recipient address. Later arguments are the extended simple mail transfer protocol (ESMTP) arguments.

Return values

Table 36. Return values

Item	Description
SMFIS_TEMPFAIL	The recipient is temporarily failed; further recipients might still be sent and the **xxfi_abort** callback function is not called.
SMFIS_REJECT	The recipient is rejected; further recipients might still be sent and the **xxfi_abort** callback function is not called.
SMFIS_DISCARD	The message is accepted or discarded, and the **xxfi_abort** callback function is called.
SMFIS_ACCEPT	The recipient is accepted and the **xxfi_abort** callback function is not be called.

Related information

xxfi_envfrom

xxfi_abort

xxfi_data callback Function:
Purpose

The **xxfi_data** callback function handles the **DATA** command.

Syntax

```
#include <libmilter/mfapi.h>
sfsistat (*xxfi_data)(
 SMFICTX *ctx
);
```

Description

The **xxfi_data** callback function is called when the client uses the **DATA** command and returns the SMFIS_CONTINUE flag.

Note: For more details on ESMTP responses, see RFC 1869.

Arguments

Table 37. Arguments

Item	Description
ctx	The opaque context structure maintained in **libmilter** parameter.

Return values

Table 38. Return values

Item	Description
SMFIS_TEMPFAIL	The message is rejected with a temporary error.
SMFIS_REJECT	The message is rejected.
SMFIS_DISCARD	The message is accepted and discarded.
SMFIS_ACCEPT	The message is accepted.

xxfi_unknown callback Function: Purpose

The **xxfi_unknown** callback function handles the unknown and unimplemented simple mail transfer protocol (SMTP) commands.

Syntax

```
#include <libmilter/mfapi.h>
sfsistat (*xxfi_unknown)(
 SMFICTX *ctx,
 const char *arg
);
```

Description

The **xxfi_unknown** callback function is called when the client uses an SMTP command that is either unknown or not implemented by the mail transfer agent (MTA) and returns the SMFIS_CONTINUE flag.

Note: The server always rejects the SMTP command. It is only possible to return a different error code.

Arguments

Table 39. Arguments

Item	Description
ctx	The opaque context structure is maintained in the **libmilter** parameter.
arg	The SMTP command including all arguments.

Return values

Table 40. Return Values

Item	Description
SMFIS_TEMPFAIL	The command is rejected with a temporary error.
SMFIS_REJECT	The command is rejected.

xxfi_header callback Function:
Purpose

The **xxfi_header** callback function handles the message header.

Syntax

```
#include <libmilter/mfapi.h>
sfsistat (*xxfi_header)(
 SMFICTX *ctx,
 char *headerf,
 char *headerv
);
```

Description

The **xxfi_header** callback function is called once for each message header and returns SMFIS_CONTINUE flag.

Note:

- Starting with sendmail 8.14, spaces after the colon in a header field are preserved if requested by using the SMFIP_HDR_LEADSPC flag. For example, the following header:

  ```
  From: sender <f@example.com>
  To:  user <t@example.com>
  Subject:no
  ```

  would be sent to a milter parameter as follows:

  ```
  "From", " sender <f@example.com>"
  "To", "  user <t@example.com>"
  "Subject", "no"
  ```

  while previously (or without the flag SMFIP_HDR_LEADSPC) it was as follows:

  ```
  "From", "sender <f@example.com>"
  "To", "user <t@example.com>"
  "Subject", "no"
  ```

- Old filter make header changes or additions to the new filters.
- For more details about the header format, see RFC 822 and RFC 2822.

Arguments

Table 41. Arguments

Item	Description
ctx	The opaque context structure is maintained in the **libmilter** parameter.
headerf	The header field name.
headerv	The header field value. The content of the header might include folded white space, that is, multiple lines with following white space where lines are separated by LF (not CR or LF). The trailing line terminator (CR or LF) is removed.

Related information:

RFC 2822

RFC 822

xxfi_eoh callback Function:
Purpose

The **xxfi_eoh** callback function handles the end of message headers.

Syntax

```
#include <libmilter/mfapi.h>
sfsistat (*xxfi_eoh)(
 SMFICTX *ctx
);
```

Description

The **xxfi_eoh** callback function is called once after all the headers have been sent and processed, and returns the SMFIS_CONTINUE flag.

Arguments

Table 42. Arguments

Item	Description
ctx	The opaque context structure is maintained in the **libmilter** parameter.

xxfi_body callback Function:
Purpose

The **xxfi_body** callback function handles a piece of a messages body.

Syntax

```
#include <libmilter/mfapi.h>
sfsistat (*xxfi_body)(
 SMFICTX *ctx,
 unsigned char *bodyp,
 size_t len
);
```

Description

The **xxfi_body** callback function is not called or is called many times between the **xxfi_eoh** and **xxfi_eom** callback function and returns the SMFIS_CONTINUE flag.

Note:

- The **bodyp** points to a sequence of bytes. It is not a C string (a sequence of characters that is terminated by '\0'). Therefore, do not use the usual C string functions like **strlen(3)** on this byte block. The byte sequence might contain '\0' characters inside the block. Hence, even if a trailing '\0' is added, C string functions might still fail to work as expected.
- Because the message bodies can be large, defining the **xxfi_body** callback function can significantly affect the filter performance.
- End-of-lines are represented as received from SMTP (normally CR/LF).
- Old filters make body changes to the new filters.
- Message bodies might be sent in multiple chunks with one call to the **xxfi_body** callback function per chunk.
- This function returns the SMFIS_SKIP flag if a milter parameter has received sufficiently many body chunks to make a decision, but still wants to invoke message modification functions that are only allowed to be called from **xxfi_eom** callback function.
- The milter parameter must negotiate this behavior with the mail transfer agent (MTA), that is, it must check whether the protocol action SMFIP_SKIP flag is available and if so, the **milter** parameter must request it.

Arguments

Table 43. Arguments

Item	Description
ctx	The opaque context structure is maintained in the **libmilter** parameter.
bodyp	The pointer to the start of this block of body data. *bodyp* is not valid outside this call to the **xxfi_body** callback function.
len	The amount of data pointed to by *bodyp*.

Related information

xxfi_eoh

xxfi_eom

xxfi_eom callback Function:
Purpose

The **xxfi_eom** callback function handles the end of message.

Syntax

```
#include <libmilter/mfapi.h>
sfsistat (*xxfi_eom)(
 SMFICTX *ctx
);
```

Description

The **xxfi_eom** callback function is called once after all calls to the **xxfi_body** callback function for a given message and returns the SMFIS_CONTINUE flag.

Note: A filter is required to make all its modifications to the message headers, body, and envelope in the **xxfi_eom** callback function. Modifications are made via the **smfi_*** routines.

Arguments

Table 44. Arguments

Item	Description
ctx	The opaque context structure is maintained in the **libmilter** parameter.

Related information

xxfi_body

xxfi_abort callback Function:
Purpose

The **xxfi_abort** callback function handles the current messages that are being aborted.

Syntax

```
#include <libmilter/mfapi.h>
sfsistat (*xxfi_abort)(
 SMFICTX *ctx
);
```

Description

The **xxfi_abort** callback function is called at any time during message processing (that is between some message-oriented routine and the **xxfi_eom** callback function) and returns the SMFIS_CONTINUE flag.

Note:

- The **xxfi_abort** callback function must reclaim any resources allocated on a per-message basis, and must be tolerant of being called between any two message-oriented callbacks.
- Calls to the **xxfi_abort** and **xxfi_eom** callback function are mutually exclusive.
- The **xxfi_abort** callback function is not responsible for reclaiming connection-specific data, because the **xxfi_close** callback function is always called when a connection is closed.
- Because the current message is already being aborted, the return value is currently ignored.
- The **xxfi_abort** callback function is only called if the message is aborted outside the filter's control and the filter has not completed the message-oriented processing. For example, if a filter has already returned the SMFIS_ACCEPT, SMFIS_REJECT, or SMFIS_DISCARD flag from a message-oriented routine. The **xxfi_abort** callback function will not be called even if the message is later aborted outside the control.

Arguments

Table 45. Arguments

Item	Description
ctx	The opaque context structure is maintained in the **libmilter**.

Related information

xxfi_close

xxfi_eom

xxfi_close callback Function:

Purpose

The **xxfi_close** callback function closes the current connection.

Syntax

```
#include <libmilter/mfapi.h>
sfsistat (*xxfi_close)(
 SMFICTX *ctx
);
```

Description

The **xxfi_close** callback function is always called once at the end of each connection and returns the SMFIS_CONTINUE flag.

The **xxfi_close** callback function might be called out-of-order, that is, before even the **xxfi_connect** callback function is called. After a connection is established by the mail transfer agent (MTA) to the filter, if the MTA decides this connection traffic will be discarded (for example, via an access_db result), no data will be passed to the filter from the MTA until the client closes down. At that time, the **xxfi_close** callback function is called. It can therefore be the only callback ever used for a given connection, and you should anticipate this possibility when crafting the **xxfi_close** callback function code. In particular, it is incorrect to assume that the private context pointer will be a value other than NULL in this callback.

The **xxfi_close** callback function is called on close even if the previous mail transaction was aborted.

The **xxfi_close** callback function is responsible for freeing any resources allocated on a per-connection basis.

Because the connection is already closing, the return value is currently ignored.

Arguments

Table 46. Arguments

Item	Description
ctx	The opaque context structure is maintained in the **libmilter** parameter.

Related information

xxfi_connect

xxfi_negotiate callback Function:
Purpose

The **xxfi_negotiate** callback function handles negotiation.

Syntax

```
#include <libmilter/mfapi.h>
#include <libmilter/mfdef.h>
sfsistat (*xxfi_negotiate)(
        SMFICTX    "xxfi_negotiate callback Function",
 unsigned long f0,
 unsigned long f1,
 unsigned long f2,
 unsigned long f3,
```

```
unsigned long *pf0,
unsigned long *pf1,
unsigned long *pf2,
unsigned long *pf3);
```

Description

The **xxfi_negotiate** callback function is called at the start of each simple mail transfer protocol (SMTP) connection and returns `SMFIS_ALL_OPTS` flag.

With this function, a **milter** parameter could dynamically determine and request operations and actions during startup. In previous versions, the actions (`f0`) were fixed in the flags field of the **smfiDesc** structure and the protocol steps (`f1`) were implicitly derived by checking whether a callback was defined. Due to the extensions in the new **milter** version, such a static selection would not work if a **milter** parameter requires new actions that are not available when talking to an older mail transfer agent (MTA). Hence, in the negotiation a callback can determine which operations are available and dynamically select those callback function which it needs and which are offered. If some operations are not available, the **milter** parameter might either fall back to an older mode or stop the session and ask the user to upgrade.

Protocol steps
`(f1, *pf1)`

:

- SMFIP_RCPT_REJ: By setting this bit, a **milter** parameter can request that the MTA should also send RCPT commands that have been rejected because the user is unknown (or similar reasons), but not those function which have been rejected because of syntax errors. If a **milter** requests this protocol step, then it should check the macro **{rcpt_mailer}:** if that is set to error, then the recipient would be rejected by the MTA. Usually the macros **{rcpt_host}** and **{rcpt_addr}** would contain an enhanced status code and an error text in that case.
- SMFIP_SKIP indicates that the MTA understand the SMFIS_SKIP return code.
- SMFIP_NR_* indicates that the MTA understand the SMFIS_NOREPLY return code. There are flags for various protocol stages:
  - SMFIP_NR_CONN: “xxfi_connect callback Function” on page 81
  - SMFIP_NR_HELO: “xxfi_helo callback Function” on page 82
  - SMFIP_NR_MAIL: “xxfi_envfrom callback Function” on page 83
  - SMFIP_NR_RCPT: “xxfi_envrcpt callback Function” on page 84
  - SMFIP_NR_DATA: “xxfi_data callback Function” on page 84
  - SMFIP_NR_UNKN: “xxfi_unknown callback Function” on page 85
  - SMFIP_NR_EOH: “xxfi_eoh callback Function” on page 87
  - SMFIP_NR_BODY: “xxfi_body callback Function” on page 87
  - SMFIP_NR_HDR: “xxfi_header callback Function” on page 86
- The SMFIP_HDR_LEADSPC flag indicates that the MTA can send header values with leading space intact. If this protocol step is requested, the MTA would not add a leading space to headers when they are added, inserted, or changed.
- The MTA can be instructed not to send information about various SMTP stages, these flags start with: SMFIP_NO*.
  - SMFIP_NOCONNECT: “xxfi_connect callback Function” on page 81
  - SMFIP_NOHELO: “xxfi_header callback Function” on page 86
  - SMFIP_NOMAIL: “xxfi_envfrom callback Function” on page 83
  - SMFIP_NORCPT: “xxfi_envrcpt callback Function” on page 84
  - SMFIP_NOBODY: “xxfi_body callback Function” on page 87

– SMFIP_NOHDRS: “xxfi_header callback Function” on page 86
– SMFIP_NOEOH: “xxfi_eoh callback Function” on page 87
– SMFIP_NOUNKNOWN: “xxfi_unknown callback Function” on page 85
– SMFIP_NODATA: “xxfi_data callback Function” on page 84

For each of these **xxfi_* callbacks** that a `milter` parameter does not use, the corresponding flag should be set in
`*pf1.`

The available actions
`(f0, *pf0)`

are described in **(xxfi_flags)**.

If a `milter` returns the SMFIS_CONTINUE flag, the `milter` sets the desired actions and protocol steps via the (output) parameters `pf0` and `pf1` (which correspond to f0 and f1, respectively). The (output) parameters `pf2` and `pf3` should be set to 0 for compatibility with future versions.

Arguments

Table 47. Arguments

Item	Description
`ctx`	The opaque context structure maintained in **`libmilter`** parameter.
`f0`	The actions offered by the MTA.
`f1`	The protocol steps offered by the MTA.
`f2`	For future extensions.
`f3`	For future extensions.
`pf0`	The actions requested by the **`milter`**
`pf1`	The protocol steps requested by the **`milter`**.
`pf2`	For future extensions.
`pf3`	For future extensions.

Return values

Table 48. Return Values

Item	Description
`SMFIS_ALL_OPTS`	If a **`milter`** just wants to inspect the available protocol steps and actions, then it can return the SMFIS_ALL_OPTS flag and the MTA would make all protocol steps and actions available to the **`milter`**. In this case, no values should be assigned to the output parameters `pf0` -`pf3` as they would be ignored.
`SMFIS_REJECT`	The **`milter`** startup fails and it would not be contacted again (for the current connection).
`SMFIS_CONTINUE`	Continue processing. In this case the **`milter`** parameter must set all output parameters `pf0` - `pf3`. See the following for an explanation how to set those output parameters.

Miscellaneous and constant functions

The miscellaneous and constant functions fetch the **`libmilter`** parameters version information.

Table 49. Constant functions

Item	Description
smfi_version	The **smfi_version** function fetches the **libmilter** parameter (runtime) version info.
smfi_setsymlist	The **smfi_setsymlist** sets the list of macros that the **libmilter** parameter wants to receive from the mail transfer agent (MTA) for a protocol stage.

Table 50. Constant functions

Item	Description
SMFI_VERSION	The **SMFI_VERSION** gets the runtime version of **libmilter** parameter.

smfi_version Function:
Purpose

The **smfi_version** function provides the **libmilter** (runtime) version information.

Syntax

```
#include <libmilter/mfapi.h>
int smfi_version(
 unsigned int *pmajor,
 unsigned int *pminor,
 unsigned int *ppl
);
```

Description

The **smfi_version** callback function might be called at any time.

The compile-time version of the **libmilter** library is available in the SMFI_VERSION macro. To extract the major and minor version as well as the current patch level from this macro, the SM_LM_VRS_MAJOR(v), SM_LM_VRS_MINOR(v), and SM_LM_VRS_PLVL(v) macros might be used. A **milter** parameter can check the SMFI_VERSION macro to determine which functions to use (at compile time via C preprocessor statements). Using this macro and the **smfi_version** function, a **milter** parameter can determine at run time whether it has been (dynamically) linked against the expected **libmilter** version. Such a function must compare the major and minor version only, not the patch level, that is, the **libmilter** library will be compatible despite different patch levels.

Arguments

Table 51. Arguments

Item	Description
pmajor	A pointer to an unsigned int variable to store major version number.
pminor	A pointer to an unsigned int variable to store minor version number.
ppl	A pointer to an unsigned int variable to store patch level number.

Return values

The **smfi_version** function returns the MI_SUCCESS value.

smfi_setsymlist Function:

Purpose

The **smfi_setsymlist** function sets the list of macros that the **milter** parameter wants to receive from the mail transfer agent (MTA) for a protocol stage.

Syntax

```
#include <libmilter/mfapi.h>
int smfi_setsymlist(
        SMFICTX    *ctx,
 int        stage,
 char     *macros
);
```

Description

The **smfi_setsymlist** callback function must be called during the **xxfi_negotiate** function, and this function can be used to override the list of macros that the **milter** parameter wants to receive from the mail transfer agent (MTA).

Note: There is an internal limit on the number of macros that can be set (currently 5). However, this limit is not enforced by the **milter** parameter, and is enforced only by the MTA, but a possible violation of this restriction is not communicated back to the **milter** parameter.

Arguments

Table 52. Arguments

Item	Description
ctx	The opaque context structure maintained in **libmilter** parameter.
stage	The protocol stage during which the macro list should be used. See the `include/libmilter/mfapi.h` file for legal values, and look for the C macros with the SMFIM_ prefix. Available protocol stages are at least the initial connection, HELO or EHLO, MAIL, RCPT, DATA, end of header, and the end of a message.
macros	The list of macros (separated by space). For example: "**{rcpt_mailer} {rcpt_host}**".

Return values

The **smfi_setsymlist** function returns the MI_FAILURE value in the following cases. Otherwise the function returns MI_SUCCESS.

- There is no enough free memory to make a copy of the macro list.
- The *macros* are NULL or empty.
- The *stage* is not a valid protocol stage.
- The macro list for a stage has been set before.

Related information

xxfi_negotiate

Debug flags for sendmail

There are a large number of debug flags built into the **sendmail** command.

Each debug flag has a number and level, where higher levels print more information. The convention is levels greater than nine print out so much information that they are used only for debugging a particular piece of code. Debug flags are set using the **-d** flag as shown in the following example:

```
debug-flag:     -d debug-list
debug-list:     debug-flag[.debug-flag]*
debug-flag:     debug-range[.debug-level]
debug-range:    integer|integer-integer
debug-level:    integer
```

```
-d12            Set flag 12 to level 1
-d12.3          Set flag 12 to level 3
-d3-17          Set flags 3 through 17 to level 1
-d3-17.4        Set flags 3 through 17 to level 4
```

The available debug flags are:

Item	Description
-d0	General debugging.
-d1	Show send information.
-d2	End with *finis*( ).
-d3	Print the load average.
-d4	Enough disk space.
-d5	Show events.
-d6	Show failed mail.
-d7	The queue file name.
-d8	DNS name resolution.
-d9	Trace RFC1413 queries.
-d9.1	Make host name canonical.
-d10	Show recipient delivery.
-d11	Trace delivery.
-d12	Show mapping of relative host.
-d13	Show delivery.
-d14	Show header field commas.
-d15	Show network get request activity.
-d16	Outgoing connections.
-d17	List MX hosts.

Note: There are now almost 200 defined debug flags in **sendmail**.

Internet Message Access Protocol and Post Office Protocol

AIX provides two Internet-based mail protocol server implementations for accessing mail remotely.

- **Post Office Protocol** (**POP** or **POP3DS**)
- **Internet Message Access Protocol** (**IMAP** or **IMAPDS**)

Each type of server stores and provides access to electronic messages. Using these mail access protocols on a server eliminates the requirement that, to receive mail, a computer must always be up and running.

The **POP** or **POP3DS** server provides an offline mail system, whereby a client, using **POP** or **POP3DS** client software, can remotely access a mail server to retrieve mail messages. The client can either download the mail messages and immediately delete the messages from the server, or download the messages and leave the messages resident on the **POP** or **POP3DS** server. After the mail is downloaded to the client machine, all mail processing is local to the client machine. The **POP** server allows access to a user mailbox one client at a time. The **POP3DS** version uses the OpenSSL libraries, which require security certificates.

The **IMAP** or **IMAPDS** server provides a superset of **POP** functionality but has a different interface. The **IMAP** or **IMAPDS** server provides an offline service, as well as an online service and a disconnected service. The protocol is designed to permit manipulation of remote mailboxes as if they were local. For

example, clients can perform searches and mark messages with status flags such as **deleted** or **answered**. In addition, messages can remain in the server database until explicitly removed. The **IMAP** server also allows simultaneous interactive access to user mailboxes by multiple clients. The **IMAPDS** version uses the OpenSSL libraries, which require security certificates.

Each type of server is used for mail access only. These servers rely on the **Simple Mail Transfer Protocol (SMTP)** for sending mail.

Each protocol is an open protocol, based on standards described in RFCs. The **IMAP** servers are based on RFC 2060 and 2061, and the **POP** servers are based on RFC 1939. Both are connection-oriented using TCP sockets. The **IMAP** server listens on port 143, and the **IMAPDS** server listens on port 993. The **POP** server listens on port 110, and the **POP3DS** server listens on port 995. All servers are handled by the **inetd** daemon.

Requirement: To use the OpenSSL versions, you must install OpenSSL. OpenSSL is available on the *AIX Toolbox for Linux Applications* CD.

Configuring IMAP and POP servers

Use this procedure to configure **IMAP** and **POP** servers.

To perform this task, you must have root authority.

1. Uncomment the **imapd** or **imapds** and **pop3d** or **pop3ds** configuration entries in the `/etc/inetd.conf` file. The following are examples of the configuration entries:

```
#imap2  stream  tcp     nowait  root    /usr/sbin/imapd imapd
#pop3   stream  tcp     nowait  root    /usr/sbin/pop3d pop3d
#imaps  stream  tcp     nowait  root    /usr/sbin/imapds imapds
#pop3s   stream  tcp     nowait  root   /usr/sbin/pop3ds pop3ds
```

2. Set the configuration files for the **imapds** server in the `/etc/imapd.cf` file and for the **pop3ds** server in the `/etc/pop3d.cf` file. By default, the less secured security handshaking protocols Secure Sockets Layer version 2 (SSLv2) and SSLv3 are enabled for the **imapds** server and the **pop3ds** server. However, you can disable SSLv2 and SSLv3 by updating the configuration files as shown in the following example. You can also enable or disable any cipher by specifying the `SSL_CIPHER_LIST` string in the configuration file. This option overwrites the default ciphers string that is hardcoded in the applications.

   Configuration file for the imapds server (`/etc/imapd.cf`):

```
#============================================================
#
# Sample IMAP Server Configuration File
#
#============================================================
#============================================================
# Uncomment the line below to Disable SSL v2 for the imap server.
#
#   Disable SSL V2  --->  SSL_OP_NO_SSLv2         YES
#   Allow SSL V2    --->  SSL_OP_NO_SSLv2         NO
#
#
#SSL_OP_NO_SSLv2         YES  <-------------- uncomment this line to disable sslv2
#============================================================
# Uncomment the line below to Disable SSL v3 for the imap server.
#
#   Disable SSL V3  --->  SSL_OP_NO_SSLv3         YES
#   Allow SSL V3    --->  SSL_OP_NO_SSLv3         NO
#
#
#SSL_OP_NO_SSLv3         YES  <-------------- uncomment this line to disable sslv3
#============================================================
# Uncomment the line below to use the user provided cipher list
# for the imap server. Parser logic expect Cipher string within " ".
```

```
#
#
#SSL_CIPHER_LIST "ALL:!LOW" <--- uncomment this line to customized (enable/disabled) ciphers string
#====================================================================
```

Configuration file for the pop3ds server (/etc/pop3d.cf):

```
#====================================================================
#
# Sample POP3 Server Configuration File
#
#====================================================================
#====================================================================
# Uncomment the line below to Disable SSL v2 for the pop3d server.
#
#   Disable SSL V2  --->  SSL_OP_NO_SSLv2         YES
#   Allow SSL V2    --->  SSL_OP_NO_SSLv2         NO
#
#
#SSL_OP_NO_SSLv2         YES  <----------- uncomment this line to disable sslv2
#====================================================================
# Uncomment the line below to Disable SSL v3 for the pop3d server.
#
#   Disable SSL V3  --->  SSL_OP_NO_SSLv3         YES
#   Allow SSL V3    --->  SSL_OP_NO_SSLv3         NO
#
#
#SSL_OP_NO_SSLv3         YES  <----------- uncomment this line to disable sslv3
#====================================================================
# Uncomment the line below to use the user provided cipher list
# for the pop3d server. Parser logic expect Cipher string within " ".
#
#
#SSL_CIPHER_LIST "ALL:!LOW" <---- uncomment this line to customized (enable/disabled) ciphers string
#====================================================================
```

3. Refresh the **inetd** daemon by running the following command:

   ```
   refresh -s inetd
   ```

Running configuration tests:

Run a few tests to verify the servers are ready for operation.

1. First, verify the servers are listening on their ports. To do this, type the following commands at a command prompt, pressing the Enter key after each command:

   ```
   netstat -a | grep imap
   netstat -a | grep pop
   ```

   The following is the output from the **netstat** commands:

   ```
   tcp     0       0     *.imap2         *.*        LISTEN
   tcp     0       0     *.imaps         *.*        LISTEN
   tcp     0       0     *.pop3          *.*        LISTEN
   tcp     0       0     *.pop3s         *.*        LISTEN
   ```

2. If you do not receive similar output, recheck the entries in the `/etc/inetd.conf` file, and then rerun the **refresh -s inetd** command.
3. To test the configuration of the imapd server, use Telnet to access the imap2 server, port 143 (for IMAPDS, Telnet port 993). When you connect using Telnet, you get the imapd prompt. You can then enter the IMAP Version 4 commands as defined in RFC 1730. To run a command, type a period (.), followed by a space, then token, command name, and any parameters. The token is used to sequence the command name. For example:

   ```
   . token CommandName parameters
   ```

   Passwords are echoed when you Telnet into the imapd server.

   In the following Telnet example, you must provide your own password where *id_password* is indicated in the **login** command.

Tip: For IMAPDS, the command and output varies slightly.

```
telnet e-xbelize 143
Trying...
Connected to e-xbelize.austin.ibm.com.
Escape character is '^]'.
* OK e-xbelize.austin.ibm.com IMAP4 server ready
. 1 login id id_password
. OK
. 2 examine /usr/spool/mail/root
* FLAGS (\Answered \Flagged \Draft \Deleted \Seen)
* OK [PERMANENTFLAGS (\Answered \Flagged \Draft \Deleted \Seen \*)]
* 0 EXISTS
* 0 RECENT
* OK [UIDVALIDITY 823888143]
. OK [READ-ONLY] Examine completed
. 3 logout
* BYE Server terminating connection
. OK Logout completed
Connection closed.
```

4. To test the configuration of the pop3d server, use Telnet to access the POP3 port, 110 (for POP3DS, Telnet port 995). When you connect using Telnet, you get the pop3d prompt. You can enter the POP commands that are defined in RFC 1725. To run one of the commands, type a period (.), followed by a space, and then the command name. For example:

   ```
   . CommandName
   ```

   Passwords are echoed when you Telnet into the pop3d server.

   In the following Telnet example, you must provide your own password where *id_password* is indicated in the **pass** command.

   Tip: For POP3DS, the command and output varies slightly.

   ```
   telnet e-xbelize 110
   Trying...
   Connected to e-xbelize.austin.ibm.com.
   Escape character is '^]'.
   +OK e-xbelize.austin.ibm.com POP3 server ready
   user id
   +OK Name is a valid mailbox
   pass id_password
   +OK Maildrop locked and ready
   list
   +OK scan listing follows
   .
   stat
   +OK 0 0
   quit
   +OK
   Connection closed.
   ```

Logging in with the SYSLOG facility

The **IMAP** (and **IMAPDS**) and **POP** (and **POP3DS**) server software sends log messages to the **SYSLOG** facility.

1. To configure the system for **IMAP** and **POP** logging through the **SYSLOG** facility, you must be the root user. Edit the /etc/syslog.conf file, and add an entry for ***.debug** as follows:

   ```
   *.debug /usr/adm/imapd.log
   ```

2. The usr/adm/imapd.log file must exist before the **syslogd** daemon reads the /etc/syslog.conf configuration file. To create this file, type the following at a command-line prompt and press Enter:

   ```
   touch /usr/adm/imapd.log
   ```

3. Refresh the **syslogd** daemon to read its configuration file again. Type the following at a command line prompt and press Enter:

   ```
   refresh -s syslogd
   ```

Mail management commands

The mail management commands are summarized here.

Item	Description
bugfiler	Stores bug reports in specific mail directories.
comsat	Notifies users of incoming mail (daemon).
mailq	Prints the contents of the mail queue.
mailstats	Displays statistics about mail traffic.
newaliases	Builds a new copy of the alias database from the `/etc/mail/aliases` file.
rmail	Handles remote mail received through the **uucp** command of the Basic Networking Utilities (BNU).
sendbug	Mails a system bug report to a specific address.
sendmail	Routes mail for local or network delivery.
smdemon.cleanu	Cleans up the **sendmail** queue for periodic housekeeping.

Mail files and directories

Mail files and directories can be arranged by function.

Item	Description
`/usr/share/lib/Mail.rc`	Sets local system defaults for all users of the mail program. A text file you can modify to set the default characteristics of the **mail** command.
`$HOME/.mailrc`	Enables the user to change the local system defaults for the mail facility.
`$HOME/mbox`	Stores processed mail for the individual user.
`/usr/bin/Mail`, `/usr/bin/mail`, or `/usr/bin/mailx`	Specifies three names linked to the same program. The mail program is *one* of the user interfaces to the mail system.
`/var/spool/mail`	Specifies the default mail drop directory. By default, all mail is delivered to the `/var/spool/mail/UserName` file.
`/usr/bin/bellmail`	Performs local mail delivery.
`/usr/bin/rmail`	Performs remote mail interface for BNU.
`/var/spool/mqueue`	Contains the log file and temporary files associated with the messages in the mail queue.

Item	Description
`/usr/sbin/sendmail`	The **sendmail** command.
`/usr/ucb/mailq`	Links to the `/usr/sbin/sendmail`. Using `mailq` is equivalent to using the `/usr/sbin/sendmail -bp` command.
`/usr/ucb/newaliases`	Links to the `/usr/sbin/sendmail` file. Using `newaliases` is equivalent to using the `/usr/sbin/sendmail -bi` command.
`/etc/netsvc.conf`	Specifies the ordering of certain name resolution services.
`/usr/sbin/mailstats`	Formats and prints the **sendmail** statistics as found in the `/etc/sendmail.st` file if it exists. The `/etc/sendmail.st` file is the default, but you can specify an alternative file.
`/etc/mail/aliases`	Describes a text version of the aliases file for the **sendmail** command. You can edit this file to create, modify, or delete aliases for your system.
`/etc/aliasesDB`	Describes a directory containing the aliases database files, `DB.dir` and `DB.pag`, that are created from the `/etc/mail/aliases` file when you run the **sendmail -bi** command.
`/etc/mail/sendmail.cf`	Contains the **sendmail** configuration information in text form. Edit the file to change this information.
`/usr/lib/smdemon.cleanu`	Specifies a shell file that runs the mail queue and maintains the **sendmail** log files in the `/var/spool/mqueue` directory.
`/etc/mail/statistics`	Collects statistics about mail traffic. This file does not grow. Use the `/usr/sbin/mailstats` command to display the contents of this file. Delete this file if you do not want to collect this information.
`/var/spool/mqueue`	Describes a directory containing the temporary files associated with each message in the queue. The directory can contain the log file.

Item	Description
/var/spool/cron/crontabs	Describes a directory containing files that the **cron** daemon reads to determine which jobs to start. The root file contains a line to start the smdemon.cleanu shell script.

IMAP and POP commands

The **imapd** and **pop3d** mail commands are used for IMAP and POP.

Item	Description
/usr/sbin/**imapd**	The Internet Message Access Protocol (IMAP) server process.
/usr/sbin/**pop3d**	The Post Office Protocol Version 3 (POP3) server process.

Transmission Control Protocol/Internet Protocol

When computers communicate with one another, certain rules, or *protocols*, allow them to transmit and receive data in an orderly fashion. Throughout the world, one of the most routinely used sets of protocols is the **Transmission Control Protocol/Internet Protocol (TCP/IP)**. (Much of Europe, however, uses the **X.25** protocol.) Some common functions for using **TCP/IP** are electronic mail, computer-to-computer file transfer, and remote login.

The **mail** user command, the Message Handling (MH) user commands, and the **sendmail** server command can use **TCP/IP** for sending and receiving mail between systems, and the Basic Networking Utilities (BNU) can use **TCP/IP** for sending and receiving files and commands between systems.

TCP/IP is a suite of protocols that specify communications standards between computers and detail conventions for routing and interconnecting networks. It is used extensively on the Internet and consequently allows research institutions, colleges and universities, government, and industry to communicate with each other.

TCP/IP allows communication between a number of computers (called hosts) connected on a network. Each network can be connected to another network to communicate with hosts on that network. Although there are many types of network technologies, many of which operate with packet-switching and stream transport, **TCP/IP** offers one major advantage: hardware independence.

Because Internet protocols define the unit of transmission and specify how to send it, **TCP/IP** can hide the details of network hardware, allowing many types of network technologies to connect and exchange information. Internet addresses allow any machine on the network to communicate with any other machine on the network. **TCP/IP** also provides standards for many of the communications services that users need.

TCP/IP provides facilities that make the computer system an Internet host, which can attach to a network and communicate with other Internet hosts. **TCP/IP** includes commands and facilities that allow you to:

- Transfer files between systems
- Log in to remote systems
- Run commands on remote systems
- Print files on remote systems
- Send electronic mail to remote users
- Converse interactively with remote users
- Manage a network

> **Note: TCP/IP** provides basic network management capability. The **Simple Network Management Protocol (SNMP)** provides more network management commands and functions.

TCP/IP terminology

You might find it useful to become familiar with the following Internet terms as they are used in relation to TCP/IP.

Item	Description
client	A computer or process that accesses the data, services, or resources of another computer or process on the network.
host	A computer that is attached to an Internet network and can communicate with other Internet hosts. The *local host* for a particular user is the computer at which that user is working. A *foreign host* is any other host name on the network. From the point of view of the communications network, hosts are both the source and destination of packets. Any host can be a client, a server, or both. On an Internet network, a host is identified by its Internet name and address.
network	The combination of two or more hosts and the connecting links between them. A *physical network* is the hardware that makes up the network. A *logical network* is the abstract organization overlaid on all or part of one or more physical networks. The Internet network is an example of a logical network. The interface program handles translation of logical network operations into physical network operations.
packet	The block of control information and data for one transaction between a host and its network. Packets are the exchange medium used by processes to send and receive data through Internet networks. A packet is sent from a *source* to a *destination*.
port	A logical connecting point for a process. Data is transmitted between processes through ports (or *sockets*). Each port provides queues for sending and receiving data. In an interface program network, each port has an Internet *port number* based on how it is being used. A particular port is identified with an Internet *socket address*, which is the combination of an Internet host address and a port number.
process	A program that is running. A process is the active element in a computer. Terminals, files, and other I/O devices communicate with each other through processes. Thus, network communications is *interprocess communications* (that is, communication between processes).
protocol	A set of rules for handling communications at the physical or logical level. Protocols often use other protocols to provide services. For example, a *connection-level protocol* uses a *transport-level protocol* to transport packets that maintain a connection between two hosts.
server	A computer or process that provides data, services, or resources that can be accessed by other computers or processes on the network.

Planning your TCP/IP network

Because **TCP/IP** is such a flexible networking tool, you can customize it to fit the specific needs of your organization. Consider the major issues in this topic when planning your network. The details of these issues are discussed in other topics. This list is intended only to introduce you to the issues.

1. Decide which type of network hardware you want to use: token-ring, Ethernet Version 2, IEEE 802.3, Fiber Distributed Data Interface (FDDI), Serial Optical Channel (SOC), or Serial Line Interface Protocol (SLIP).
2. Plan the physical layout of the network. Consider which functions each host machine will serve. For example, you must decide which machine or machines will serve as gateways before you cable the network.
3. Decide whether a *flat* network or a *hierarchical* network organization best fits your needs.

   If your network is fairly small, at a single site, and consists of one physical network, then a flat network probably suits your needs. If your network is very large or complex with multiple sites or multiple physical networks, a hierarchical network might be a more efficient network organization for you.
4. If your network is to be connected to other networks, you must plan how your gateways should be set up and configured. Things to consider are:
   a. Decide which machine or machines will serve as gateways.
   b. Decide whether you need to use static or dynamic routing, or a combination of the two. If you choose dynamic routing, decide which routing daemons each gateway will use in light of the types of communications protocols you need to support.
5. Decide on an addressing scheme.

If your network will not be part of a larger internetwork, choose the addressing scheme that best fits your needs. If you want your network to be connected to a larger internetwork such as the Internet, you will have to obtain an official set of addresses from your internet service provider (ISP).

6. Decide whether your system needs to be divided into subnets. If so, decide how you will assign subnet masks.
7. Decide on a naming scheme. Each machine on the network needs its own unique host name.
8. Decide whether your network needs a name server for name resolution or if using the `/etc/hosts` file will be sufficient.

   If you choose to use name servers, consider the type of name servers you need and how many you need to serve your network efficiently.
9. Decide the types of services you want your network to provide to remote users; for example, mail services, print services, file sharing, remote login, remote command execution, and others.

Installation of TCP/IP

This section will discuss the Installation of **Transmission Control Protocol/Internet Protocol (TCP/IP)**.

For information on installing **Transmission Control Protocol/Internet Protocol (TCP/IP)**, see Installation and Migration.

Configuration of TCP/IP

After the **TCP/IP** software is installed on your system, you are ready to begin configuring your system.

Many **TCP/IP** configuration tasks can be performed in more than one way, either by:

- Using the System Management Interface Tool (SMIT)
- Editing a file format
- Issuing a command at the shell prompt.

For example, the `rc.net` shell script performs required minimum host configuration for **TCP/IP** during the system startup process (the `rc.net` script is run by the configuration manager program during the second boot phase). By using SMIT to perform the host configuration, the `rc.net` file is configured automatically.

Alternatively, you can configure the `/etc/rc.bsdnet` file using a standard text editor. With this method, you can specify the traditional UNIX **TCP/IP** configuration commands such as **ifconfig**, **hostname**, and **route**. If using the file edit method, you must enter `smit configtcp` fast path and then select **BSD Style rc Configuration**. See List of TCP/IP Programming References in *Communications Programming Concepts* for information about **TCP/IP** files and file formats.

A few tasks, such as configuring a name server, cannot be done using SMIT.

Host configuration

Each host machine on your network must be configured to function according to the needs of the end users and the network as a whole.

For each host on the network, you must configure the network interface, set the Internet address, and set the host name. You also must set up static routes to gateways or other hosts, specify daemons to be started by default, and set up the `/etc/hosts` file for name resolution (or set up the host to use a name server for name resolution).

Host as server configuration

If the host machine will have a specific function like serve as a gateway, file server, or name server, you must perform the necessary configuration tasks after the basic configuration is complete.

For example, if your network is organized hierarchically and you want to use the **Domain Name** protocol to resolve names into Internet addresses, you will need to configure at least one name server to provide this function for your network.

Remember, a server host does not have to be a dedicated machine, it can be used for other things as well. If the name server function for your network is fairly small, the machine might also be used as a workstation or as a file server for your network.

Note: If your system has NIS installed, these services can also provide name resolution.

Gateway configuration

If your network is going to communicate with other networks, you will need to configure at least one gateway host.

You must consider which communications protocols you want to support, and then use whichever routing daemon (the **routed** or **gated** daemon) that supports those protocols.

TCP/IP configuration and management commands

You can use a variety of commands to configure and manage a **TCP/IP** network. They are described in this table.

Item	Description
arp	Displays or changes the Internet address to hardware address translation tables used by the **Address Resolution** protocol.
finger	Returns information about users on a specified host.
host	Shows the Internet address of a specified host or the host name of a specified Internet address.
hostname	Shows or sets the Internet name and address of the local host.
ifconfig	Configures network interfaces and their characteristics.
netstat	Shows local and foreign addresses, routing tables, hardware statistics, and a summary of packets transferred.
no	Sets or shows current network kernel options.
ping	Determines whether a host is reachable.
route	Permits you to manipulate the routing tables manually.
ruptime	Shows status information on hosts that are connected to local physical networks and are running the **rwhod** server.
rwho	Shows status information for users on hosts that are connected to local physical networks and running the **rwhod** server.
setclock	Reads the network time service and sets the time and date of the local host accordingly.
timedc	Returns information about the **timed** daemon.
trpt	Reports protocol tracing on TCP sockets.
whois	Provides the Internet name directory service.

Configuring a TCP/IP network

Use this procedure as a guide for configuring your network. Ensure that you have read and understood the appropriate material.

Before beginning this procedure, make sure that the following prerequisites are true:

1. Network hardware is installed and cabled. For more information about installing and cabling your hardware, see "TCP/IP local area network adapter cards" on page 155.
2. **TCP/IP** software is installed. For more information about installing the TCP/IP software, see the *Installation and migration*.

After you bring your network up and it is running properly, you might find it useful to refer to this checklist for the purpose of debugging.

To configure your **TCP/IP** network, use the following steps:

1. Read "TCP/IP protocols" on page 118 for the basic organization of **TCP/IP**. You should understand:
   - the layered nature of **TCP/IP** (that is, different protocols reside at different layers)
   - how data flows through the layers
2. Minimally configure each host machine on the network. This means adding a network adapter, assigning an IP address, and assigning a host name to each host, as well as defining a default route to your network. For background information on these tasks, refer to "TCP/IP network interfaces" on page 158, "TCP/IP addressing" on page 164, and "Naming hosts on your network" on page 172.

   Note: Each machine on the network needs this basic configuration whether it will be an end-user host, a file server, a gateway, or a name server.
3. Configure and start the **inetd** daemon on each host machine on the network. Read "TCP/IP daemons" on page 332 and then follow the instructions in "Configuring the inetd daemon" on page 333.
4. Configure each host machine to perform either local name resolution or to use a name server. If you are setting up a hierarchical Domain Name network, configure at least one host to function as a name server. Read and follow the instructions in "Name resolution" on page 174.
5. If your network will communicate with any remote networks, configure at least one host to function as a gateway. The gateway can use static routes or a routing daemon to perform internetwork routing. Read and follow the instructions in "TCP/IP routing" on page 334.
6. Decide which services each host machine on the network will use. By default, all services are available. Follow the instructions in "Client network services" on page 333 if you wish to make a particular service unavailable.
7. Decide which hosts on the network will be servers, and which services a particular server will provide. Follow the instructions in "Server network services" on page 333 to start the server daemons you wish to run.
8. Configure any remote print servers you will need. See Printing administration in *Printers and printing* for more information.
9. **Optional:** If desired, configure a host to use or to serve as the master time server for the network. See the **timed** daemon in the *Commands Reference, Volume 5* for more information.

Authentication and the secure rcmds

These commands have been enhanced to provide additional authentication methods to those used today.

The secure rcmds are **rlogin**, **rcp**, **rsh**, **telnet**, and **ftp**. By default these commands use the, *Standard AIX* method of authentication. The two additional methods are Kerberos V.5 and Kerberos V.4.

When using the Kerberos V.5 authentication method, the client gets a Kerberos V.5 ticket from the DCE security server or Native Kerberos server. The ticket is a portion of the user's current DCE or Native credentials encrypted for the **TCP/IP** server with which they wish to connect. The daemon on the **TCP/IP** server decrypts the ticket. This allows the **TCP/IP** server to absolutely identify the user. If the DCE or Native principal described in the ticket is allowed access to the operating system user's account, the connection proceeds.

> **Note:** Beginning with DCE version 2.2, the DCE security server can return Kerberos V.5 tickets. The secure rcmds in the AIX operating system uses the Kerberos V.5 library and the GSSAPI library provided by NAS (Network Authentication Service) version 1.3.

In addition to authenticating the client, Kerberos V.5 forwards the current user's credentials to the **TCP/IP** server. If the credentials are marked forwardable, the client sends them to the server as a Kerberos TGT (Ticket Granting Ticket). On the **TCP/IP** server side, if one is communicating with a DCE security server, the daemon upgrades the TGT into full DCE credentials using the **k5dcecreds** command.

The **ftp** command uses a different authentication method than the other commands. It uses the GSSAPI security mechanism to pass the authentication between the **ftp** command and the **ftpd** daemon. Using the **clear**/**safe**/**private** subcommands, the **ftp** client supports data encryption.

Between operating system clients and servers, **ftp** has been enhanced to allow multiple-byte transfers for encrypted data connections. The standards define only single-byte transfers for encrypted data connections. When connected to third-party machines and using data encryption, **ftp** follows the single-byte transfer limit.

Note: The **rlogin**, **rsh**, and **telnet** secure rcmds commands, along with the **klogin** and **kshell** Kerberos V.5 authentication methods, allow three attempts before the connection to the remote host is closed.

System configuration for the secure rcmds

For all of the secure rcmds, there is a system-level configuration mechanism to determine which authentication methods are allowed for that system. The configuration controls both outgoing and incoming connections.

The authentication configuration consists of a library, `libauthm.a`, and two commands, **lsauthent** and **chauthent**, that provide command-line access to the library's two routines: **get_auth_methods** and **set_auth_methods**.

The system supports three different authentication methods: Kerberos V.5, Kerberos V.4, and *Standard AIX*. The authentication method defines which method is used to authenticate a user across a network.

- Kerberos V.5 is the most common method, as it is the basis for the Distributed Computing Environment (DCE). The operating system either upgrades incoming Kerberos V.5 tickets to full DCE credentials or uses incoming Native Kerberos V.5 tickets.
- Kerberos V.4 is used by only two of the secure rcmds: **rsh** and **rcp**. It is provided to support compatibility with an earlier version on SP systems and is functional only on one. A Kerberos V.4 ticket is not upgraded to DCE credentials.
- The term, *Standard AIX* authentication method, refers to the authentication method that is used by the AIX operating system.

There is a fallback implementation when more than one authentication method is configured. If the first method fails to connect, the client attempts to authenticate by using the next authentication method that is configured.

Authentication methods can be configured in any order. The only exception is that *Standard AIX* must be the final authentication method that is configured because there is no fallback option from it. If *Standard AIX* is not a configured authentication method, password authentication is not attempted, and any connection attempt that uses this method is rejected.

It is possible to configure the system without any authentication methods. In this case, the system refuses all connections from and to any terminal that uses secure rcmds. Also, because Kerberos V.4 is supported only with the **rsh** and **rcp** commands, a system that is configured to use only Kerberos V.4 does not allow connections that use **telnet**, **ftp**, or **rlogin**.

Related information:

get_auth_method subroutine

set_auth_method subroutine

lsauthent command

chauthent command

Kerberos V.5 user validation for the secure rcmds

When using the Kerberos V.5 authentication method, the **TCP/IP** client gets a service ticket encrypted for the **TCP/IP** server. When the server decrypts the ticket, it has a secure method of identifying the user (by DCE or Native principal).

However, it still needs to determine if this DCE or Native principal is allowed access to the local account. Mapping the DCE or Native principal to the local operating system account is handled by a shared library, `libvaliduser.a`, which has a single subroutine, **kvalid_user**. If a different method of mapping is preferred, the system administrator must provide an alternative for the `libvaliduser.a` library.

DCE configuration for the secure rcmds

To use the secure rcmds, two DCE principals must exist for every network interface to which they can be connected.

They are:

```
host/FullInterfaceName
ftp/FullInterfaceName
```

where *FullInterfaceName* is the interface name and domain name for the primary *HostName.DomainName*.

Native configuration for the secure rcmds

To use the secure rcmds, two principles must exist for every network interface to which they can be connected.

They are:

```
host/FullInterfaceName@Realmname
ftp/FullInterfaceName@Realmname
```

where *FullInterfaceName* is the interface name and the domain name for the primary `HostName.DomainName`. *Realmname* is the name of the Native Kerberos V realm.

TCP/IP customization

To customize **TCP/IP**, create a `.netrc` file.

The `.netrc` file specifies automatic login information for the **ftp** and **rexec** commands. You can also write new **ftp** macros, which are defined in the `$HOME/.netrc` file. To customize key functions or sequences, create and edit the `$HOME/.3270keys` file. In addition, the `.k5login` file specifies which DCE principals on which cells are allowed access to the user's account.

Creating the .netrc file

These steps describe how to create and edit the `$HOME/.netrc` file:

1. You must have a copy of the `/usr/samples/tcpip/netrc` file.
2. The **securetcpip** command must not be running on your system.

To create the `.netrc` file:

1. Copy the `/usr/samples/tcpip/netrc` file to your `$HOME` directory by typing the following command:
   ```
   cp /usr/samples/tcpip/netrc $HOME
   ```
2. Edit the `$HOME/netrc` file to supply the appropriate *HostName*, *LoginName*, and *Password* variables. For example:
   ```
   machine host1.austin.century.com login fred password bluebonnet
   ```
3. To set the permissions on the `$HOME/netrc` file to 600 by using the **chmod** command at the command line prompt ($), type:
   ```
   chmod 600 $HOME/netrc
   ```
4. Rename the `$HOME/netrc` file to `$HOME/.netrc` file. The initial period (.) causes the file to be hidden.

```
mv $HOME/netrc $HOME/.netrc
```

The $HOME/.netrc file can contain multiple login definitions and up to 16 macros per login definition.

Writing ftp macros

These steps describe how to create an **ftp** macro.

You must have created the $HOME/.netrc file.

To write an **ftp** macro:

1. Edit the $HOME/.netrc file to include the following instructions:

   ```
   macdef init
   put schedule
   ```

   Be sure to insert a blank line at the end of your **ftp** macro. The blank line terminates the **ftp** macro. In the above example, the **macdef** subcommand defines the subcommand macro `init`. The line following is the command the macro specifies, in this case `put` *schedule*, where *schedule* is the name of a file.
2. After you create the **ftp** macro, at the command line prompt, type:

   ```
   ftp hostname
   ```

   Where *hostname* is the name of the host to which you are connecting. **ftp** scans the $HOME/.netrc file for a login definition matching your host name and uses that login definition to log you in.
3. After you log in, at the command line prompt, type:

   ```
   ftp init
   ```

   In this example, **ftp** scans for the macro named `init` and executes the command or commands the macro specifies.

   An **ftp** macro is associated with the login entry immediately preceding it. **ftp** macros are not global to the $HOME/.netrc file. The macro `init` is executed automatically upon login. Other macros can be executed from the **ftp** prompt (`ftp>`) by typing the following:

   ```
   $getit
   ```

   In this example, the $ executes the **ftp** macro **getit**.

Changing the assignment of a key set

When customizing **TCP/IP**, you can use this procedure to change key functions and sequences.

1. You must have working knowledge of the **vi** editor.
2. The vi editor must be on your system.

The following steps describe how to create and edit the $HOME/.3270keys file to customize key functions or sequences:

1. Copy the /etc/3270.keys file to the $HOME directory and rename it .3270keys using the following command:

   ```
   cp /etc/3270.keys $HOME/.3270keys
   ```
2. Change the bind statements in the $HOME/.3270keys file to change the assignment of a key set using the following steps:
   a. Start the vi editor on a new file and enter insert mode.
   b. Press the Ctrl-V key sequence and then the key that you want to map. This displays a value for the pressed key.
   c. Place the displayed value on the appropriate line in the Sequence column of the $HOME/.3270keys file.

   For example, after you have invoked the vi editor and entered insert mode, press Ctrl-V and then Alt-Insert. This displays [[141q. The first [is replaced with \e in the Sequence column so that the configured line looks like the following:

   ```
   3270 Function Sequence  Key
   bind  pa1      "\e[141q" #a_insert
   ```

.k5login file:

The `.k5login` file is used when Kerberos V.5 authentication is used for the secure rcmds. This file specifies which DCE principals on which cells are allowed access to the user's account.

The file is located at `$HOME/.k5login`. It should be owned by the local user and the owner should have read permission on this file. The minimum permission setting for this file is 400.

The `.k5login` file contains a list of the DCE principal/cell pairs allowed to access the account. The principal/cell pairs are kept in Kerberos format (as opposed to DCE format). For example, if the file contains

```
UserA@Cell1
```

then the DCE principal `UserA` on the DCE cell `Cell1` can access the account.

If the DCE principal is the same as the user's account name, and if there is no`$HOME/.k5login` file for the user's account, the DCE principal gains access to the account (provided Kerberos V.5 authentication is configured).

For more information about Kerberos V.5 authentication, see “Authentication and the secure rcmds” on page 104.

Methods for communicating with other systems and users

There are several methods of communicating with other systems and users. Two methods are discussed here. The first method is by connecting a local host to a remote host. The second method is by conversing with a remote user.

Local host connections to a remote host

These **TCP/IP** host connection commands are for remote login and command execution.

There are several reasons why you might need to access a computer other than your own. For example, your system administrator might need to reassign permissions to a sensitive file you have been working on, or you might need to access a personal file from someone else's workstation. You can even connect to your own computer from someone else's computer station. Remote login functions, such as the **rlogin**, **rexec**, and **telnet** commands, enable the local host to perform as an input/output terminal host. Key strokes are sent to the remote host, and the results are displayed on the local monitor. When you end the remote login session, all functions return to your local host.

TCP/IP contains the following commands for remote login and command execution:

Item	Description
rexec	The **rexec** command makes it possible to execute commands interactively on different foreign hosts when you log in to a remote host with the **rlogin** command. This command is disabled by the system manager if extra security is needed for your network. When you issue the **rexec** command, your local host searches the `$HOME/.netrc` file of the remote host for your user name and a password from your local host. If these are found, the command you requested to be run on the local host will then be run. Otherwise, you will be required to supply a login name and password before the request can be honored.

Item	Description
rlogin	The **rlogin** command makes it possible to log in to similar foreign hosts. Unlike **telnet**, which can be used with different remote hosts, the **rlogin** command can be used on UNIX hosts only. This command is disabled by the system manager if extra security is needed for your network. The **rlogin** command is similar to the **telnet** command in that both allow a local host to connect to a remote host. The only difference is that the **rlogin** command is not a trusted command and can be disabled if your system needs extra security. The **rlogin** command is not a trusted command because both the `$HOME/.rhosts` file, which is owned by the local user, and the `/etc/hosts.equiv` file, which is owned by your system manager, keep a listing of remote hosts that have access to the local host. Therefore, if you leave your terminal on while unattended, an unauthorized user could examine the names and passwords contained in these files, or worse, could damage a remote host in some way. Ideally, remote users should be required to type a password after issuing the **rlogin** command, but it is quite possible to bypass this recommended feature. If neither the `$HOME/.rhosts` file nor the `/etc/hosts.equiv` file contains the name of a remote host that is trying to log in, the local host prompts for a password. The remote password file is first checked to verify the password entered; the login prompt is again displayed if the password is not correct. Pressing tilde and period (~.) at the login prompt ends the remote login attempt. The **rlogin** command can also be configured to use Kerberos V.5 to authenticate the user. This option allows the user to be identified without the use of a `$HOME/.rhosts` file or passing the password across the network. For more information about this use of the **rlogin** command, see "Authentication and the secure rcmds" on page 104.
rsh and **remsh**	The **rsh** and **remsh** commands make it possible to execute commands on similar foreign hosts. All required input must be performed by the remote host. The **rsh** and **remsh** commands are disabled by the system manager if extra security is needed for your network. The **rsh** command can be used in two ways: • To execute a single command on a remote host when a command name is specified • To execute the **rlogin** command when no command name is specified When the **rsh** command is issued, your local host searches the `/etc/hosts.equiv` file on the remote host for permission to log in. If that is unsuccessful, the `$HOME/.rhosts` file is searched. Both of these files are lists of remote hosts having login permission. Remote users should be required to type a password after issuing the **rsh** command. It is also possible to eliminate the need to issue the **rlogin** command. The **rsh** command permits the execution of commands on a remote host, but does not provide a means of bypassing the password requirement. If a password is needed to access a remote host, then a password is needed to use the **rsh** command as well because both commands access the `$HOME/.rhosts` and `/etc/hosts.equiv` files. The **rsh** command can also be configured to use Kerberos V.5 to authenticate the user. This option allows the user to be identified without either the use of a `$HOME/.rhosts` file or passing the password across the network. For more information about this use of the **rsh** command, see "Authentication and the secure rcmds" on page 104.

Item	Description
telnet, **tn**, and **tn3270**	The **telnet** command is a terminal emulation program that implements the TELNET protocol and allows you to log in on a similar or dissimilar foreign host. It uses TCP/IP to communicate with other hosts in the network. **Note:** For convenience, **telnet** hereafter refers to the **telnet**, **tn**, and **tn3270** commands. The **telnet** command is one way a user can log in to a remote host. The most important feature of the **telnet** command is that it is a *trusted* command. In contrast, the **rlogin** command, which also allows for remote login, is not considered a trusted command. A system may need extra security to prevent unauthorized users from gaining access to its files, stealing sensitive data, deleting files, or placing viruses or worms on the system. The security features of TCP/IP are designed to help prevent these occurrences. A user who wishes to log in to a remote host with the **telnet** command must provide the user name and password of an approved user for that computer. This is similar to the procedure used for logging in to a local host. When successfully logged in to a remote host, the user's terminal runs as if directly connected to the host. The **telnet** command supports an option called *terminal negotiation*. If the remote host supports terminal negotiation, the **telnet** command sends the local terminal type to the remote host. If the remote host does not accept the local terminal type, the **telnet** command attempts to emulate a 3270 terminal and a DEC VT100 terminal. If you specify a terminal to emulate, the **telnet** command does not negotiate for terminal type. If the local and remote hosts cannot agree on a terminal type, the local host defaults to **none**. The **telnet** command supports these 3270 terminal types: 3277-1, 3278-1, 3278-2, 3278-3, 3278-4, and 3278-5. If you are using the **telnet** command in 3270 mode on a color display, the colors and fields are displayed just as those on a 3279 display, by default. You can select other colors by editing one of the keyboard mapping files in the preceding list of terminal types. When the **telnet** session has ended, the display is reset to the colors in use before the session began. The **telnet** command can also be configured to use Kerberos V.5 to authenticate the user. This option allows the user to be identified without either the use of a `$HOME/.rhosts` file or passing the password across the network. For more information about this use of the **telnet** command, see “Authentication and the secure rcmds” on page 104.

Note: The **rsh** and **rexec** commands can be used to execute commands on a remote host, but neither is a trusted command, so they may not meet all security levels configured into your computer. As a result, these commands can be disabled if your system requires extra security.

Remote host logins

You can log in to a remote host by using the **telnet** command.

In order to do this, you must have a valid user ID and password for the remote host.

To log in to a remote host (`host1` in this example), type:

```
telnet host1
```

Information similar to the following displays on your screen:

```
Trying . . .
Connected to host1
Escape character is '^T'.

AIX telnet (host1)

AIX Operating System
Version 7.1
(/dev/pts0)
login:_
```

After you have logged in, you can issue commands. To log out of the system and close the connection, press the Ctrl-D key sequence.

If you cannot log in, cancel the connection by pressing the Ctrl-T key sequence.

Conversing with a remote user

Use the **talk** command to have a real-time conversation with another user on a remote host.

1. The **talkd** daemon must be active on both the local and remote host.
2. The user on the remote host must be logged in.

The **talk** command requires a valid address on which to bind. The host name of the remote terminal must be bound to a working network interface that is usable by other network commands, such as the **ping** command. If a machine has no network interface that is a standalone terminal, it must bind its host name to the loopback address (127.0.0.1) in order for the **talk** command to work.

Using electronic mail, you can send text messages to other users on a local network and receive mail from them as well. If a computer system is configured appropriately and you know the appropriate electronic address, you can send electronic mail messages across the world to someone on a remote system.

TCP/IP contains the following commands for remote communications:

Item	Description
mail	Sends and receives electronic memos and letters
talk	Lets you have an interactive conversation with a user on a remote host

1. To talk to the remote user `dale@host2` logged in on a remote host, `jane@host1` types:

   ```
   talk dale@host2
   ```

   A message similar to the following displays on the screen of `dale@host2`:

   ```
   Message from TalkDaemon@host1 at 15:16...
   talk: connection requested by jane@host1.
   talk: respond with: talk jane@host1
   ```

   This message informs `dale@host2` that `jane@host1` is trying to converse with her.
2. To accept the invitation, `dale@host2` types:

   ```
   talk jane@host1
   ```

   Users `dale@host2` and `jane@host1` are now able to have an interactive conversation.
3. To end a conversation at any time, either user can press the Ctrl-C key sequence. This returns them to the command line prompt.

File transfers

Although it is possible to send relatively short files using electronic mail, there are more efficient ways of transferring large files.

Electronic mail programs are usually designed to transmit relatively small amounts of text; therefore, other means are needed to transfer large files effectively. The **ftp**, **rcp**, and **tftp** commands rely on **TCP/IP** to establish direct connections from your local host to a remote host. Basic Network Utilities (BNU) can also use **TCP/IP** to provide direct connections with foreign hosts.

File transfers using the ftp and rcp commands

Use the **ftp** command to copy a file from a remote host. The **ftp** command does not preserve file attributes or copy subdirectories. If either of these conditions are necessary, use the **rcp** command.

Item	Description
ftp	Uses the **File Transfer Protocol (FTP)** to transfer files between hosts that use different file systems or character representations, such as EBCDIC and ASCII. It provides for security by sending passwords to the remote host and also permits automatic login, file transfers, and log off.
rcp	Copies one or more files between a local host and a remote host, between two separate remote hosts, or between files at the same remote host. This command is similar to the **cp** command except that it works only for remote file operations. If extra security is needed for your network, this command is disabled by the system manager.

Before attempting file transfer using the **ftp** and **rcp** commands, make sure the following conditions are true:

1. You must have remote login permission specified in the remote host's `$HOME/.netrc` file if the automatic login feature is to be used. Otherwise, you must know a login name and password for the remote host. For more information about the `.netrc` file, see "Creating the .netrc file" on page 106.

   Alternatively, the system can be configured to use Kerberos V.5 authentication. This is used instead of the `.netrc` or `$HOME/.rhosts` files. See "Authentication and the secure rcmds" on page 104.
2. If you wish to copy a file from a remote host, you must have read permission for that file.

   Note: The read and write permissions for files and directories on a remote host are determined by the login name used.
3. If you wish to copy a file from your local host to a remote host, you must have write permission for the directory that is to contain the copied file. Also, if the directory on the remote host contains a file with the same name as the file that you wish to place in it, you must have write permission to append the file on the remote host.

Logging in to a remote host directly:

When using **TCP/IP** to transfer files, you can use this procedure to log in to a remote host directly.

1. Use the **cd** command to change to the directory that contains the file you want to send (sending a file) or to the directory where you want the transferred file to reside (receiving a file).
2. Log in to the remote host directly by typing:

   ```
   ftp HostName
   ```

   If you have automatic login permission, information similar to the following is displayed on your local host:

   ```
   Connected to canopus.austin.century.com.
   220 canopus.austin.century.com FTP server
   (Version 4.1 Sat Nov 23 12:52:09 CST 1995) ready.
   331 Password required for dee.
   230 User dee logged in.
   ftp>
   ```

   Otherwise, information similar to the following is displayed on your local host:

   ```
   Connected to canopus.austin.century.com.
   220 canopus.austin.century.com FTP server
   (Version 4.1 Sat Nov 23 12:52:09 CST 1995) ready.
   Name (canopus:eric): dee
   331 Password required for dee.
   Password:
   230 User dee logged in.
   ftp>
   ```
3. Enter your login name and password when prompted by the system.

You are now ready to copy a file between two hosts.

Logging in to a remote host indirectly:

When using **TCP/IP** to transfer files, you can use this procedure to log in to a remote host indirectly.

1. Use the **cd** command to change to the directory that contains the file you want to send (sending a file) or to the directory where you want the transferred file to reside (receiving a file).
2. Log in to the remote host indirectly by typing:

   ```
   ftp
   ```
3. When the ftp> prompt is displayed, type:

   open *HostName*

   If you have automatic login permission, information similar to the following is displayed on your local host:

   ```
   Connected to canopus.austin.century.com.
   220 canopus.austin.century.com FTP server
   (Version 4.1 Sat Nov 23 12:52:09 CST 1995) ready.
   331 Password required for dee.
   230 User dee logged in.
   ftp>
   ```

   Otherwise, information similar to the following is displayed on your local host:

   ```
   Connected to canopus.austin.century.com.
   220 canopus.austin.century.com FTP server
   (Version 4.1 Sat Nov 23 12:52:09 CST 1995) ready.
   Name (canopus:eric): dee
   331 Password required for dee.
   Password:
   230 User dee logged in.
   ftp>
   ```
4. Type in your name and password when prompted by the system.

Copying a file from a remote host to a local host:

Use the **ftp** command to copy a file from a remote host to a local host.

To copy a file from a remote host to a local host using the **ftp** command, you must first log in to the remote system either directly or indirectly. See Logging in to a remote host directly or Logging in to a remote host indirectly for instructions.

Note: The **ftp** command uses the ASCII default transfer type to copy files.

To copy a file from a remote host to a local host:

1. Determine if the file that you want to copy is in the current directory by running the **dir** subcommand. (The **dir** subcommand for the **ftp** command works in the same way as the **ls -l** command.) If the file is not there, use the **cd** subcommand to move to the proper directory.
2. To copy your local file using binary image, type:

   ```
   binary
   ```
3. To copy a file to your host, type:

   ```
   get FileName
   ```

   The file is placed in the directory from which you issued the **ftp** command.
4. To end the session, press the Ctrl-D key sequence, or type `quit`.

Copying a file from a local host to a remote host:

Use the **ftp** command to copy a file from a local host to a remote host.

To copy a file from a local host to a remote host using the **ftp** command, you must first log in to the remote system either directly or indirectly. See Logging in to a remote host directly or Logging in to a remote host indirectly for instructions.

Note: The **ftp** command uses the ASCII default transfer type to copy files.

To copy a file from a local host to a remote host:

1. If you want to place the file in a directory other than the `$HOME` directory, use the **cd** subcommand to move to the desired directory.
2. To copy your local file using binary image, type:
   ```
   binary
   ```
3. To copy a file to the remote host, type:
   ```
   put FileName
   ```
4. To end the session, press the Ctrl-D key sequence, or type `quit`.

File transfers using the tftp and utftp commands

Use the **tftp** and **utftp** commands for the **Trivial File Transfer Protocol** (**TFTP**) to transfer files to and from hosts.

Because **TFTP** is a single-file transfer protocol, the **tftp** and **utftp** commands do not provide all of the features of the **ftp** command. If extra security is required for your network, the system manager can disable this command.

> **Note:** The **tftp** command is not available when your host is operating at a high level of security.

Before attempting a file transfer using the **tftp** and **utftp** commands, make sure the following conditions are true:

1. If you wish to copy a file *from* a remote host, you must have *read* permission for the directory that contains the desired file.
2. If you wish to copy a file *to* a remote host, you must have *write* permission for the directory in which the file is to be placed.

Copying a file from a remote host:

When using **TCP/IP** to copy files, you can use this procedure to copy files from a remote host.

1. To establish a connection to a remote host, type:
   ```
   tftp host1
   ```
   In this example, `host1` is the name of the host to which you wish to connect.

   The `tftp>` prompt is displayed.
2. To determine that a connection has been established, type:
   ```
   status
   ```
   A message similar to the following is displayed:
   ```
   Connected to host1
   Mode: netascii Verbose: off Tracing: off
   Remxt-interval: 5 seconds, Max-timeout: 25 seconds
   tftp>
   ```
3. Enter the **get** subcommand, the name of the file to be transferred, and the name to be assigned to the file on the remote system:
   ```
   get /home/alice/update update
   ```

   The `/home/alice` directory on the remote host must have read permission set for other users. In this example, the `/home/alice/update` file is transferred from `host1` to the `update` file in the current directory on the local system.
4. To end the session, type:
   ```
   quit
   ```
   or press the Ctrl-D key sequence.

Copying a file to a remote host:

When using **TCP/IP** to copy files, you can use this procedure to copy a file to a remote host.

1. To establish a connection to a remote host, type:

   ```
   tftp host1
   ```

   In this example, `host1` is the name of the host to which you wish to connect.

   The `tftp>` prompt is displayed.
2. To determine that a connection has been established, type:

   ```
   status
   ```

   A message similar to the following is displayed:

   ```
   Connected to host1
   Mode: netascii Verbose: off Tracing: off
   Remxt-interval: 5 seconds, Max-timeout: 25 seconds
   tftp>
   ```
3. Enter the **put** subcommand, the name of the file to be transferred from the local host, and the path and file name for the file on the remote host:

   ```
   put myfile /home/alice/yourfile
   ```

   The `/home/alice` directory on the remote host must have write permission set for others. The `myfile` file, located in the user's current working directory, is transferred to `host1`. The path name must be specified unless a default has been established. The `myfile` file appears on the remote host as `yourfile`.
4. To end the session, type:

   ```
   quit
   ```

   or use the Ctrl-D key sequence.

Printing files to a remote system

If you have a local printer attached to your host, then the following procedures refer to printing to a remote printer. If you have no local printer, then the following procedures refers to printing to a nondefault remote printer.

1. Your host name must appear in the `/etc/hosts.lpd` file of the remote host.

   Note: The queuing system does not support multibyte host names.

   To implement changes to the `/etc/hosts.lpd` file without restarting the system, use the System Resource Controller (SRC) **refresh** command.
2. You must be able to determine the queue name and the remote printer name in your local `/usr/lib/lpd/qconfig` file

You can use either the **enq** command or the System Management Interface Tool (SMIT) to complete this task.

> **Note:** This section explains how to print to a remote host at the simplest level possible. For more information and ideas about remote printing, see **enq** command.

Placing a print job in a remote print queue

When using **TCP/IP** to print files, you can use this procedure to place a job in a remote print queue.

In order for you to place a job in a remote printing queue, your host name must appear in the `/etc/hosts.lpd` file of the remote host (the queuing system does not support multibyte host names). To implement changes to the `/etc/hosts.lpd` file without restarting the system, use the System Resource Controller (SRC) **refresh** command. Also, you must be able to determine the queue name and the remote printer name in your local `/usr/lib/lpd/qconfig` file.

1. Find the appropriate queue name and remote device name. The queue name usually begins with the letters `rp` followed by a numeral or set of numerals. The remote printer name usually begins with the letters `drp` followed by a numeral or set of numerals.
2. Enter the following command:

   ```
   enq -P QueueName:PrinterName FileName
   ```

   where *QueueName* is the name of the queue (such as `rp1`), and *PrinterName* is the name of the printer (such as `drp1`) as found in the `/usr/lib/lpd/qconfig` file. Do not omit the colon (`:`) between the *QueueName* and the *PrinterName*. *FileName* is the name of the file that you wish to print.

The following are examples of how the **enq** command can be used:

- To print the file `memo` on the default printer, type:

  ```
  enq memo
  ```

- To print the file `prog.c` with page numbers, type:

  ```
  pr prog.c | enq
  ```

  The **pr** command puts a heading at the top of each page that includes the date the file was last modified, the name of the file, and the page number. The **enq** command then prints the file.

- To print the file `report` on the next available printer configured for the `fred` queue, type:

  ```
  enq -P fred report
  ```

- To print several files beginning with the prefix `sam` on the next available printer configured for the `fred` queue, type:

  ```
  enq -P fred sam*
  ```

  All files beginning with the prefix `sam` are included in one print job. Normal status commands show only the title of the print job, which in this case is the name of the first file in the queue unless a different value was specified with the **-T** flag. To list the names of all the files in the print job, use the **enq -A -L** long status command .

Enqueuing a job using SMIT

When using **TCP/IP** to enqueue files, you can use the **smit** command.

1. To enqueue a job using SMIT, type the following command:

   ```
   smit
   ```

2. Select the **Spooler**, and start a print job menu.
3. Select the **File to Print** option, and type the name of the file you wish to print.
4. Select the **Print Queue** option, and select the name of the remote printer to which you wish to print.

You are now ready to print to a remote printer.

Printing files from a remote system

Occasionally, you might need to print a file that is located on a remote host. The location of the printout depends upon which remote printers are available to the remote host.

1. You must be able to log in to the remote system using the **rlogin** or **telnet** command.
2. You must have read permission for the remote file that you wish to print on your local printer.

> **Note:** This procedure explains how to print to a remote host at the simplest level possible. For more information and ideas about remote printing, read about the **enq** command.

To print from a remote system:

1. Log in to the remote system using the **rlogin** or **telnet** command.
2. Find the appropriate queue name and remote device name. The queue name usually begins with the letters `rp` followed by a numeral or set of numerals. The remote printer name usually begins with the letters `drp` followed by a numeral or set of numerals.

3. Type the following command:

   ```
   enq -P QueueName:PrinterName FileName
   ```

   where *QueueName* is the name of the queue (such as `rp1`), and *PrinterName* is the name of the printer (such as `drp1`) as found in the `/usr/lib/lpd/qconfig` file. Do not omit the : (colon) between the *QueueName* and the *PrinterName*. *FileName* is the name of the file that you want to print.
4. End the connection to the remote host by pressing the Ctrl-D sequence or by typing `quit`.

Displaying status information

You can use **TCP/IP** commands to determine the status of a network, display information about a user, and resolve host information needed to communicate to another host or user.

TCP/IP status commands

TCP/IP contains status commands to determine the status of local and remote hosts and their networks.

Item	Description
finger or **f**	Displays information about the current users on a specified host. This information can include the user's login name, full name, and terminal name, as well as the date and time of login.
host	Resolves a host name into an Internet address or an Internet address into a host name.
ping	Helps determine the status of a network or host. It is most commonly used to verify that a network or host is currently running.
rwho	Shows which users are logged in to hosts on a local network. This command displays the user name, host name, and date and time of login for everyone on the local network.
whois	Identifies to whom a user ID or nickname belongs. This command can only be used if your local network is connected to the Internet.

Displaying information about all users logged in to a host

Use this procedure to view information about *all* users logged in to a remote host.

To display information about all users logged in to a remote host:

1. Log in to the remote host with which you want to communicate.
2. To display information about all users logged in to host `alcatraz`, type:

   ```
   finger @alcatraz
   ```

   Information similar to the following is displayed:

   ```
   brown    console  Mar 15 13:19
   smith    pts0     Mar 15 13:01
   jones    tty0    Mar 15 13:01
   ```

User `brown` is logged in at the console, user `smith` is logged in from a pseudo teletype line pts0, and user `jones` is logged in from a tty0. Your system administrator can set up your system so that the **finger** command works differently. If you encounter any problems using the **finger** command, contact your system administrator.

Displaying information about a user logged in to a host

Use this procedure to view information about a *specific* user logged in to a remote host.

To display information about a single user logged in to a remote host:

1. Log in to the remote host with which you want to communicate.
2. To display information about user `brown` on host `alcatraz`, type:

   ```
   finger brown@alcatraz
   ```

   Information similar to the following is displayed:

```
Login name:  brown
Directory:  /home/brown     Shell:  /home/bin/xinit -L -n Startup
On since May 8 07:13:49 on console
No Plan.
```

Your system administrator can set up your system so that the **finger** command works differently. If you encounter any problems using the **finger** command, contact your system administrator.

TCP/IP protocols

Protocols are sets of rules for message formats and procedures that allow machines and application programs to exchange information. These rules must be followed by each machine involved in the communication in order for the receiving host to be able to understand the message. The TCP/IP *suite* of protocols can be understood in terms of layers (or levels).

This figure depicts the layers of the **TCP/IP** protocol. From the top they are, Application Layer, Transport Layer, Network Layer, Network Interface Layer, and Hardware.

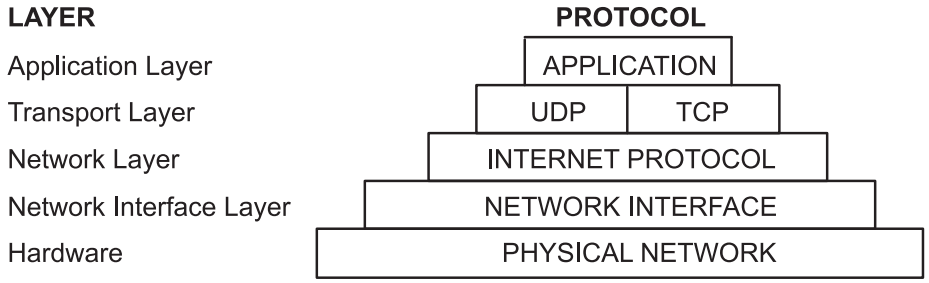


Figure 4. TCP/IP suite of protocols

TCP/IP carefully defines how information moves from sender to receiver. First, application programs send messages or streams of data to one of the Internet Transport Layer Protocols, either the **User Datagram Protocol** (**UDP**) or the **Transmission Control Protocol** (**TCP**). These protocols receive the data from the application, divide it into smaller pieces called *packets*, add a destination address, and then pass the packets along to the next protocol layer, the Internet Network layer.

The Internet Network layer encloses the packet in an **Internet Protocol** (**IP**) datagram, puts in the datagram header and trailer, decides where to send the datagram (either directly to a destination or else to a gateway), and passes the datagram on to the Network Interface layer.

The Network Interface layer accepts **IP** datagrams and transmits them as *frames* over a specific network hardware, such as Ethernet or Token-Ring networks.

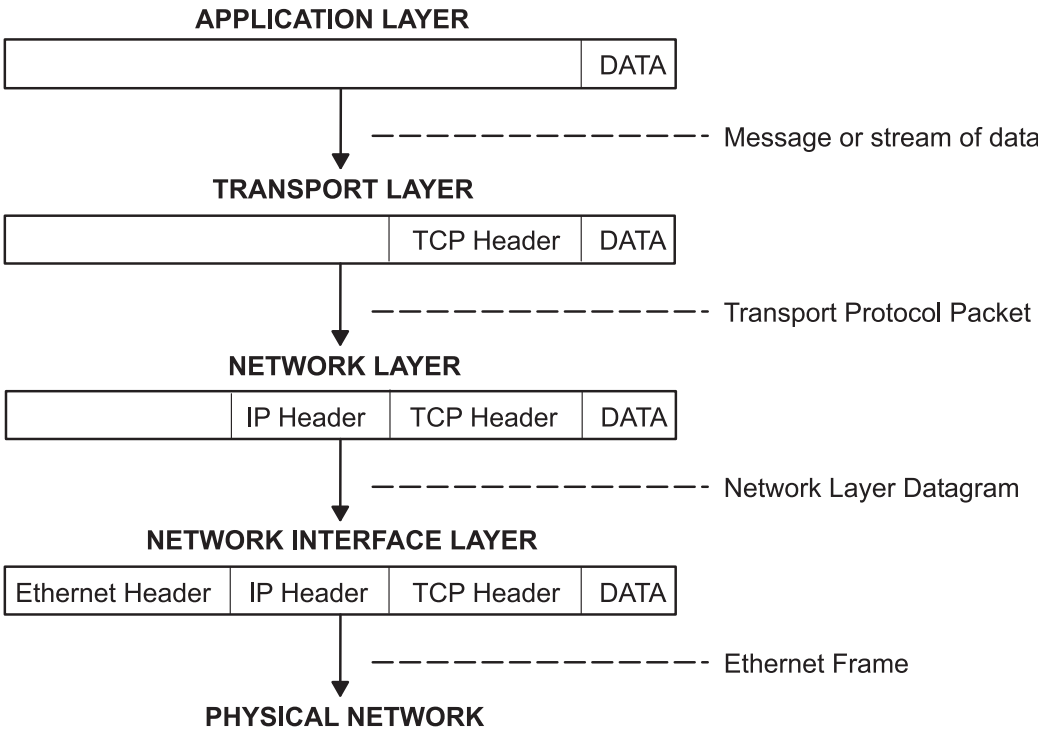


Figure 5. Movement of information from sender application to receiver host

This figure shows the flow of information down the TCP/IP protocol layers from the Sender to the Host.

Frames received by a host go through the protocol layers in reverse. Each layer strips off the corresponding header information, until the data is back at the application layer.

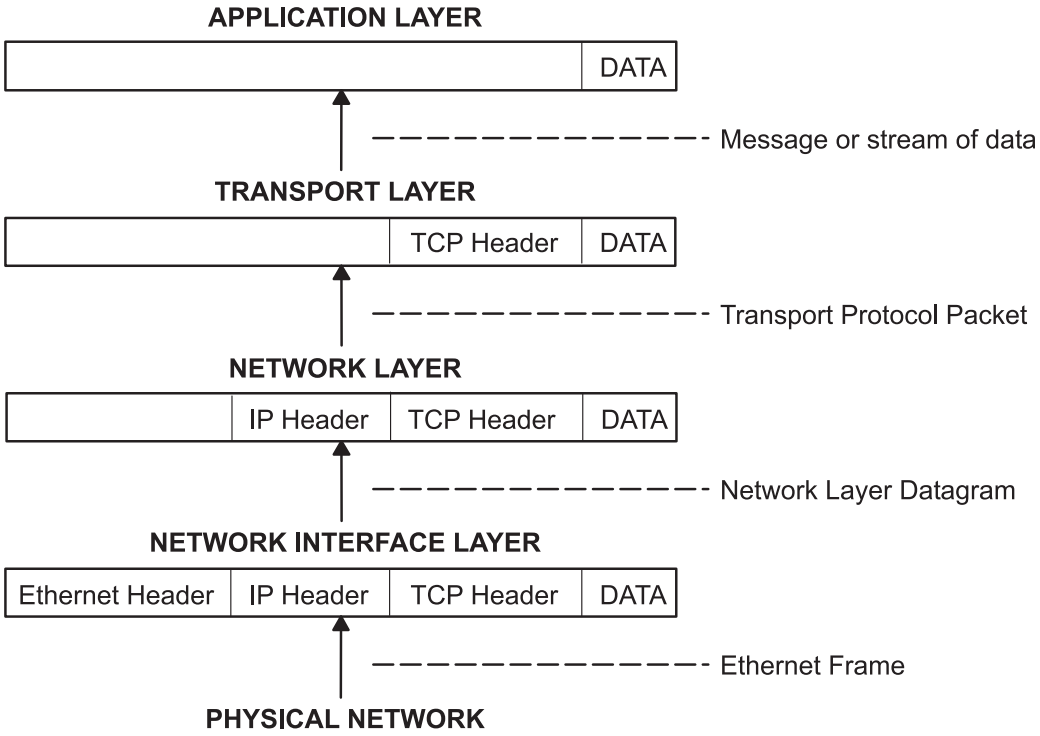
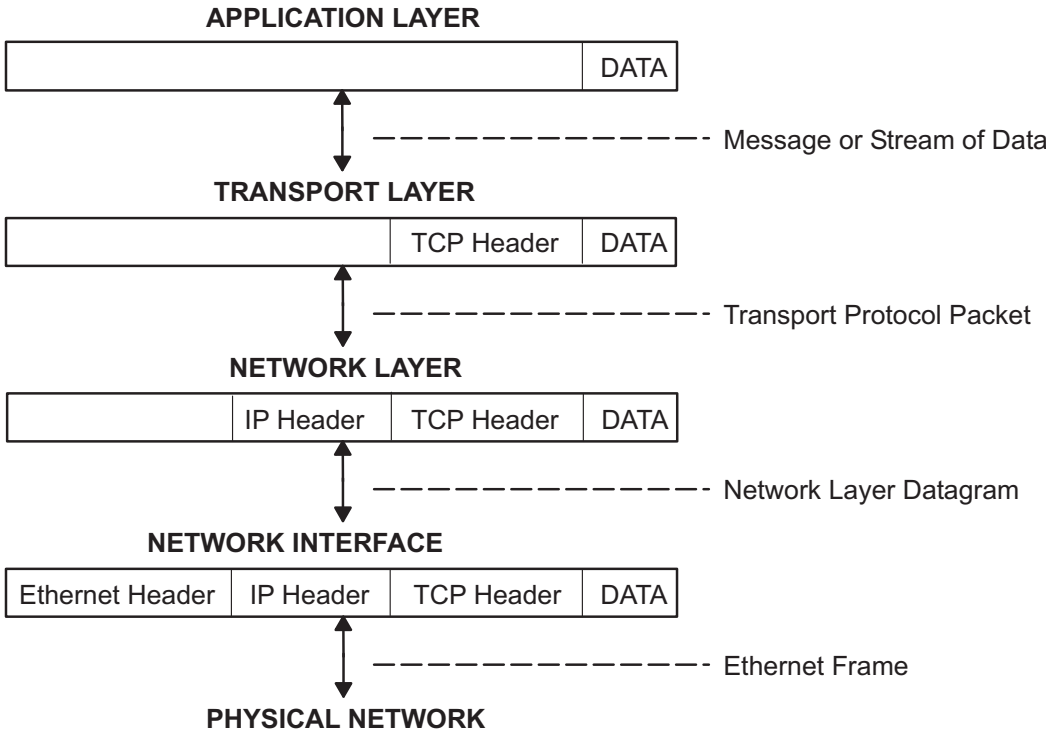


Figure 6. Movement of information from host to application

This figure shows the flow of information up the **TCP/IP** protocol layers from the Host to the Sender.

Frames are received by the Network Interface layer (in this case, an Ethernet adapter). The Network Interface layer strips off the Ethernet header, and sends the datagram up to the Network layer. In the Network layer, the Internet Protocol strips off the IP header and sends the packet up to the Transport layer. In the Transport layer, the **TCP** (in this case) strips off the **TCP** header and sends the data up to the Application layer.

Hosts on a network send and receive information simultaneously. Figure 7 on page 121 more accurately represents a host as it communicates.



Note: Headers are added and stripped in each protocol layer as data is transmitted and received by a host

Figure 7. Host data transmissions and receptions

This figure shows data flowing both ways through the **TCP/IP** layers.

Internet Protocol (IP) version 6

Internet Protocol (**IP**) version 6 (**IPv6** or IP*ng*) is the next generation of **IP** and has been designed to be an evolutionary step from **IP** version 4 (**IPv4**).

While **IPv4** has allowed the development of a global Internet, it is not capable of carrying much farther into the future because of two fundamental factors: limited address space and routing complexity. The **IPv4** 32-bit addresses do not provide enough flexibility for global Internet routing. The deployment of Classless InterDomain Routing (CIDR) has extended the lifetime of **IPv4** routing by a number of years, but the effort to better manage the routing will continue. Even if **IPv4** routing could be scaled up, the Internet will eventually run out of network numbers.

The Internet Engineering Task Force (IETF) recognized that **IPv4** would not be able to support the phenomenal growth of the Internet, so the IETF IP*ng* working group was formed. Of the proposals that were made, **Simple Internet Protocol Plus** (**SIPP**) was chosen as an evolutionary step in the development of IP. This was renamed to IP*ng*, and RFC1883 was finalized in December of 1995.

IPv6 extends the maximum number of Internet addresses to handle the ever increasing Internet user population. As an evolutionary change from **IPv4**, **IPv6** has the advantage of allowing 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.

This overview is intended to give the reader a general understanding of the IP*ng* protocol. For detailed information, please see RFCs 2460, 2373, 2465, 1886, 2461, 2462, and 2553.

Security provides security information on the **TCP/IP** suite of protocols, including **IPv6**. For details about IP Security, versions 4 and 6, see Internet Protocol security.

IPv6 expanded routing and addressing:

IPv6 increases the **IP** address size from 32 bits to 128 bits, thereby supporting more levels of addressing hierarchy, a much greater number of addressable nodes, and simpler autoconfiguration of addresses.

IPv6 has three types of addresses:

Item	Description
unicast	A packet sent to a unicast address is delivered to the interface identified by that address. A unicast address has a particular scope: link-local, site-local, global. There are also two special unicast addresses: • ::/128 (unspecified address) • ::1/128 (loopback address)
multicast	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: node-local, link-local, site-local, and organization-local.
anycast	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). For example, several web servers listening on an anycast address. When a request is sent to the anycast address, only one responds. An anycast address is indistinguishable from a unicast address. 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 has been superseded by the multicast address.

IPv6 autoconfiguration:

The primary mechanisms available that enable a node to start up and communicate with other nodes over an **IPv4** network are hard-coding, **BOOTP**, and **DHCP**.

IPv6 introduces the concept of *scope* to **IP** addresses, one of which is link-local. This allows a host to construct a valid address from the predefined link-local prefix and its local identifier. This local identifier is typically derived from the medium access control (MAC) address of the interface to be configured. Using this address, the node can communicate with other hosts on the same subnet and, for a fully-isolated subnet, might not need any other address configuration.

IPv6 meaningful addresses:

With **IPv4**, the only generally recognizable meaning in addresses are broadcast (typically all 1s or all 0s), and classes (for example, a class D is multicast). With **IPv6**, the prefix can be quickly examined to determine *scope* (for example, link-local), multicast versus unicast, and a mechanism of assignment (provider-based or geography-based).

Routing information might be explicitly loaded into the upper bits of addresses as well, but this has not yet been finalized by the IETF (for provider-based addresses, routing information is implicitly present in the address).

IPv6 duplicate address detection:

When an interface is initialized or reinitialized, it uses autoconfiguration to tentatively associate a link-local address with that interface (the address is not yet assigned to that interface in the traditional sense). At this point, the interface joins the all-nodes and solicited-nodes multicast groups, and sends a neighbor discovery message to these groups. By using the multicast address, the node can determine whether that particular link-local address has been previously assigned, and choose an alternate address.

This eliminates accidentally assigning the same address to two different interfaces on the same link. (It is still possible to create duplicate global-scope addresses for nodes that are not on the same link.)

Neighbor discovery/stateless address autoconfiguration:

Neighbor Discovery Protocol (**NDP**) 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. **IPv6** defines both a stateful and a stateless address autoconfiguration mechanism. *Stateless autoconfiguration* requires no manual configuration of hosts; minimal, if any, configuration of routers; and no additional servers.

Hosts also use **NDP** to find neighboring routers that are willing to forward packets on their behalf and detect changed link-layer addresses. **NDP** uses the **Internet Control Message Protocol** (**ICMP**) Version 6 with its own unique message types. In general terms, the **IPv6** Neighbor Discovery protocol corresponds to a combination of the **IPv4 Address Resolution Protocol** (**ARP**), ICMP Router Discovery (RDISC), and **ICMP** Redirect (ICMPv4), but with many improvements over these **IPv4** protocols.

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 subnets 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.

Routing simplification:

To simplify routing issues, **IPv6** addresses are considered in two parts: a prefix and an ID. This might seem the same as the **IPv4** net-host address breakdown, but it has two advantages.

Item	Description
no class	No fixed number of bits for prefix or ID, which allows for a reduction in loss due to over-allocation
nesting	An arbitrary number of divisions can be employed by considering different numbers of bits as the prefix.

Case 1

128 bits
node address

Case 2

Item	Description
n bits	128-*n* bits
Subnet prefix	Interface ID

Case 3:

Item	Description	
n bits	80-*n* bits	48 bits
Subscriber prefix	Subnet ID	Interface ID

Case 4:

Item	Description		
s bits	*n* bits	*m* bits	128-*s*-*n*-*m* bits
Subscribe prefix	Area ID	Subnet ID	Interface ID

Generally, IPv4 cannot go beyond Case 3, even with Variable Length Subnet Mask (VLSM is a means of allocating **IP** addressing resources to subnets according to their individual need rather than some general network-wide rule). This is as much an artifact of the shorter address length as the definition of variable length prefixes, but is worth noting nonetheless.

Header format simplification:

IPv6 simplifies the **IP** header by removing entirely or by moving to an extension header some of the fields found in the **IPv4** header. It defines a more flexible format for optional information (the extension headers).

Specifically, note the absence of:
- header length (length is constant)
- identification
- flags
- fragment offset (moved into fragmentation extension headers)
- header checksum (upper-layer protocol or security extension header handles data integrity).

Table 53. IPv4 header

Item	Description	Description	Description	Description
Version	IHL	Type of Service	Total Length	
Identification	Identification	Identification	Flags	Fragment Offset
Time to Live	Time to Live	Protocol	Header Checksum	Header Checksum
Source Address	Source Address	Source Address	Source Address	Source Address
Destination Address	Destination Address	Destination Address	Destination Address	Destination Address
Options	Options	Options	Options	Padding

Table 54. IPv6 header

Item	Description	Description	Description	Description
Version	Prio		Flow Label	
Payload Length	Payload Length	Payload Length	Next Header	Hop Limit
Source Address	Source Address	Source Address	Source Address	Source Address
Destination Address	Destination Address	Destination Address	Destination Address	Destination Address

IP*ng* includes an improved options mechanism over **IPv4**. **IPv6** options are placed in separate extension headers that are located between the **IPv6** header and the transport-layer header in a packet. Most extension headers are not examined or processed by any router along a packet delivery path until it arrives at its final destination. This mechanism facilitates a major improvement in router performance for packets containing options. In **IPv4** the presence of any options requires the router to examine all options.

Another improvement is that, unlike **IPv4** options, **IPv6** extension headers can be of arbitrary length and the total amount of options carried in a packet is not limited to 40 bytes. This feature, plus the manner in which it is processed, permits **IPv6** options to be used for functions that were not practical in **IPv4**, such as the **IPv6** Authentication and Security Encapsulation options.

To improve the performance when handling subsequent option headers and the transport protocol that follows, IPv6 options are always an integer multiple of eight octets long to retain this alignment for subsequent headers.

By using extension headers instead of a protocol specifier and options fields, newly defined extensions can be integrated more easily.

Current specifications define extension headers in the following ways:

- Hop-by-hop options that apply to each hop (router) along the path
- Routing header for loose/strict source routing (used infrequently)
- A fragment defines the packet as a fragment and contains information about the fragment (**IPv6** routers do not fragment)
- Authentication (See **TCP/IP** security in *Security*
- Encryption (See **TCP/IP** security in *Security*
- Destination options for the destination node (ignored by routers).

Improved quality-of-service/traffic control:

While quality of service can be controlled by use of a control protocol such as **RSVP**, **IPv6** provides for explicit priority definition for packets by using the priority field in the **IP** header.

A node can set this value to indicate the relative priority of a particular packet or set of packets, which can then be used by the node, one or more routers, or the destination to make choices concerning the packet (that is, dropping it or not).

IPv6 specifies two types of priorities, those for congestion-controlled traffic, and those for non-congestion-controlled traffic. No relative ordering is implied between the two types.

Congestion-controlled traffic is defined as traffic that responds to congestion through some sort of "back-off" or other limiting algorithm. Priorities for congestion-controlled traffic are:

Item	Description
0	uncharacterized traffic
1	"filler" traffic (for example, netnews)
2	unattended data transfer (for example, mail)
3	(reserved)
4	attended bulk transfer (for example, **FTP**)
5	(reserved)
6	interactive traffic (for example, Telnet)
7	control traffic (for example, routing protocols)

Non-congestion-controlled traffic is defined as traffic that responds to congestion by dropping (or simply not resending) packets, such as video, audio, or other real-time traffic. Explicit levels are not defined with examples, but the ordering is similar to that for congestion-controlled traffic:

- The lowest value that the source is most willing to have discarded should be used for traffic.
- The highest value that the source is least willing to have discarded should be used for traffic.

This priority control is only applicable to traffic from a particular source address. Control traffic from one address is not an explicitly higher priority than attended bulk transfer from another address.

Flow labeling:

Outside of basic prioritization of traffic, **IPv6** defines a mechanism for specifying a particular flow of packets. In **IPv6** terms, a *flow* is defined as a sequence of packets sent from a particular source to a particular (unicast or multicast) destination for which the source desires special handling by the intervening routers.

This flow identification can be used for priority control, but might also be used for any number of other controls.

The flow label is chosen randomly, and does not identify any characteristic of the traffic other than the flow to which it belongs. This means that a router cannot determine that a packet is a particular type by examining the flow label. It can, however, determine that it is part of the same sequence of packets as the last packet containing that label.

Note: Until **IPv6** is in general use, the flow label is mostly experimental. Uses and controls involving flow labels have not yet been defined nor standardized.

IPv6 tunneling:

Tunneling provides a way to use an existing **IPv4** routing infrastructure to carry **IPv6** traffic.

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**. While the **IPv6** infrastructure is being deployed, the existing **IPv4** routing infrastructure can remain functional, and can be used to carry **IPv6** traffic.

IPv6 or **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:

Item	Description
Router-to-Router	**IPv6** or **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** or **IPv4** hosts can tunnel **IPv6** packets to an intermediary **IPv6** or **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** or **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** or **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 router-to-router or host-to-router methods, the **IPv6** packet is being tunneled to a router. In host-to-host or router-to-host methods, the **IPv6** packet is tunneled all the way to its final destination.

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. However, the encapsulating node needs to maintain soft state information for each tunnel, such as the maximum transmission unit (MTU) of the tunnel, to process **IPv6** packets forwarded into the tunnel.

There are two types of tunnels in **IPv6**:

automatic tunnels
Automatic tunnels are configured by using **IPv4** address information embedded in an **IPv6** address – the **IPv6** address of the destination host includes information about which **IPv4** address the packet should be tunneled to.

configured tunnels
Configured tunnels must be configured manually. These tunnels are used when using **IPv6** addresses that do not have any embedded **IPv4** information. The **IPv6** and **IPv4** addresses of the endpoints of the tunnel must be specified.

For information on configuring automatic and configured tunnels, see "Setting up tunneling in IPv6" on page 134.

IPv6 multihomed link-local and site-local support:

A host can have more than one interface defined. A host with two or more active interfaces is called multihomed. Each interface has a link-local address associated with it.

Link-local addresses are sufficient for allowing communication among nodes attached to the same link.

A multihomed host has two or more associated link-local addresses. The AIX **IPv6** implementation has 4 options to handle how link-layer address resolution is resolved on multihomed hosts. Option 1 is the default.

Item	Description
Option 0	No multihomed actions are taken. Transmissions will go out on the first link-local interface. When the **Neighbor Discovery Protocol** (**NDP**) must perform address resolution, it multicasts a Neighbor Solicitation message out on each interface with a link local address defined. NDP queues the data packet until the first Neighbor Advertisement message is received. The data packet is then sent out on this link.
Option 1	When the **NDP** must perform address resolution, that is, when sending a data packet to a destination and the link-layer information for the next hop is not in the Neighbor Cache, it multicasts a Neighbor Solicitation message out on each interface with a link-local address defined. **NDP** then queues the data packet until it gets the link-layer information. **NDP** then waits until a response is received for each interface. This guarantees that the data packets are sent on the appropriate outgoing interfaces. If **NDP** did not wait, but responded to the first Neighbor Advertisement received, it would be possible for a data packet to be sent out on a link not associated with the packet source address. Because **NDP** must wait, a delay in the first packet being sent occurs. However, the delay occurs anyway in waiting for the first response.
Option 2	Multihomed operation is allowed, but dispatching of a data packet is limited to the interface specified by main_if6. When the **NDP** must perform address resolution, it multicasts a Neighbor Solicitation message out on each interface with a link-local address defined. It then waits for a Neighbor Advertisement message from the interface specified by main_if6 (see the **no** command). Upon receiving a response from this interface, the data packet is sent out on this link.
Option 3	Multihomed operation is allowed, but dispatching of a data packet is limited to the interface specified by main_if6 and site-local addresses are only routed for the interface specified by main_site6 (see the **no** command). The NDP operates just as it does for Option 2. For applications that route data packets using site-local addresses on a multihomed host, only the site-local address specified by main_site6 are used.

Upgrading to IPv6 with IPv4 configured:

This scenario leads you through a manual upgrade from **IPv4** to **IPv6**.

The network used in this example consists of a router and two subnets. There are two hosts on each subnet: the router, and another host. You will upgrade each machine on this network to **IPv6**. By the end of the scenario, the router will advertise prefix `3ffe:0:0:aaaa::/64` on network interface `en0` and prefix

3ffe:0:0:bbbb::/64 on network interface en1. You will first configure the machines to temporarily support **IPv6** so that you can test them. You will then configure the machines so they will be **IPv6**-ready at boot time.

If you are running the AIX operating system and do not have your **IPv4** settings configured, see "Upgrading to IPv6 with IPv4 not configured" on page 129.

Things to Consider

- The information in this how-to scenario was tested using specific versions of AIX. The results you obtain might vary significantly depending on your version and level of AIX.

Step 1: Set up the hosts for IPv6

On the hosts on both subnets, do the following:

1. Make sure **IPv4** is configured by typing the following command:

   ```
   netstat -ni
   ```

   Your results should look similar to the following:

   ```
   Name  Mtu   Network     Address            Ipkts Ierrs    Opkts Oerrs  Coll
   en0   1500  link#2      0.6.29.4.55.ec      279393     0     2510     0     0
   en0   1500  9.3.230.64  9.3.230.117         279393     0     2510     0     0
   lo0   16896 link#1                             913     0      919     0     0
   lo0   16896 127         127.0.0.1               913     0      919     0     0
   lo0   16896 ::1                               913     0      919     0     0
   ```

2. With root authority, configure your **IPv6** settings by typing the following command:

   ```
   autoconf6
   ```

3. Rerun the following command:

   ```
   netstat -ni
   ```

   Your results should look similar to the following:

   ```
   Name  Mtu   Network     Address            Ipkts Ierrs    Opkts Oerrs  Coll
   en0   1500  link#2      0.6.29.4.55.ec      279679     0     2658     0     0
   en0   1500  9.3.230.64  9.3.230.117         279679     0     2658     0     0
   en0   1500  fe80::206:29ff:fe04:55ec        279679     0     2658     0     0
   sit0  1480  link#3      9.3.230.117              0     0        0     0     0
   sit0  1480  ::9.3.230.117                        0     0        0     0     0
   lo0   16896 link#1                            2343     0     2350     0     0
   lo0   16896 127         127.0.0.1              2343     0     2350     0     0
   lo0   16896 ::1                              2343     0     2350     0     0
   ```

4. Start the **ndpd-host** daemon by typing the following command:

   ```
   startsrc -s ndpd-host
   ```

Step 2: Set up the router for IPv6

1. Make sure that the **IPv4** settings are configured by typing the following command:

   ```
   netstat -ni
   ```

2. With root authority, type the following command:

   ```
   autoconf6
   ```

3. Manually configure global addresses on the router's interfaces belonging to each of the two subnets by typing the following commands:

   ```
   # ifconfig en0 inet6 3ffe:0:0:aaaa::/64 eui64 alias
   # ifconfig en1 inet6 3ffe:0:0:bbbb::/64 eui64 alias
   ```

   You will need to do this for every subnet that your router is sending packets to.

4. To activate IPv6 forwarding, type the following:

   ```
   no -o ip6forwarding=1
   ```

5. To start the **ndpd-router** daemon, type the following:

```
startsrc -s ndpd-router
```

The **ndpd-router** daemon will advertise prefixes corresponding to the global addresses that you configured on the router. In this case, the ndpd-router will advertise prefix `3ffe:0:0:aaaa::/64` on `en0` and prefix `3ffe:0:0:bbbb::/64` on `en1`

Step 3. Set up IPv6 to be configured on the hosts at boot time

Your newly configured **IPv6** will be deleted when you reboot the machine. To enable **IPv6** host functionality every time you reboot, do the following:

1. Open the `/etc/rc.tcpip` file using your favorite text editor.
2. Uncomment the following lines in that file:

```
# Start up autoconf6 process
start /usr/sbin/autoconf6 ""

# Start up ndpd-host daemon
start /usr/sbin/ndpd-host "$src_running"
```

3. Add the **-A** flag to `start /usr/sbin/autoconf6 ""`:

```
start /usr/sbin/autoconf6 "" -A
```

When you reboot, your **IPv6** configuration will be set. Repeat this process for each host.

Step 4: Set up IPv6 to be configured on the router at boot time

Your newly configured **IPv6** will be deleted when you reboot. To enable **IPv6** router functionality every time you reboot, do the following:

1. Open the `/etc/rc.tcpip` file in your favorite text editor.
2. Uncomment the following line in that file:

```
# Start up autoconf6 process
start /usr/sbin/autoconf6 ""
```

3. Add the following lines immediately after the line that you just uncommented in the previous step:

```
# Configure global addresses for router
ifconfig en0 inet6 3ffe:0:0:aaaa::/64 eui64 alias
ifconfig en1 inet6 3ffe:0:0:bbbb::/64 eui64 alias
```

In this scenario, our network has only two subnets, `en0` and `en1`. You will need to add a line to this file for every subnet that your router is sending packets to.

4. Uncomment the following line in the file:

```
# Start up ndpd-router daemon
start /usr/sbin/ndpd- router "$src_running"
```

When you reboot, **IPv6** will be automatically started.

Upgrading to IPv6 with IPv4 not configured:

This scenario shows how to set up hosts and a router for **IPv6** without **IPv4** settings configured.

The network used in this example consists of a router and two subnets. There are two hosts on each subnet: the router, and another host. By the end of the scenario, the router will advertise prefix `3ffe:0:0:aaaa::/64` on network interface `en0` and prefix `3ffe:0:0:bbbb::/64` on network interface `en1`. You will first configure the machines to temporarily support **IPv6** so that you can test them. You will then configure the machines so they will be IPv6-ready at boot time.

This scenario assumes that the `bos.net.tcp.client` fileset is installed.

To upgrade to **IPv6** with **IPv4** already configured, see "Upgrading to IPv6 with IPv4 configured" on page 127.

Things to Consider

- The information in this how-to scenario was tested using specific versions of AIX. The results you obtain might vary significantly depending on your version and level of AIX.

Step 1: Set up the hosts for IPv6

1. With root authority, type the following command on each host on the subnet:

   ```
   autoconf6 -A
   ```

   This will bring up all **IPv6**-capable interfaces on the system.

   Note: To bring up a subset of interfaces, use the **-i** flag. For example, `autoconf6 -i en0 en1` will bring up interfaces `en0` and `en1`.
2. Type the following command to view your interfaces:

   ```
   netstat -ni
   ```

   Your results should look similar to the following:

   ```
   Name  Mtu   Network     Address            Ipkts Ierrs    Opkts Oerrs  Coll
   en0   1500  link#3      0.4.ac.17.b4.11        7     0       17     0     0
   en0   1500  fe80::204:acff:fe17:b411            7     0       17     0     0
   lo0   16896 link#1                           436     0      481     0     0
   lo0   16896 127         127.0.0.1            436     0      481     0     0
   lo0   16896 ::1                             436     0      481     0     0
   ```
3. Start the **ndpd-host** daemon by typing the following command:

   ```
   startsrc -s ndpd-host
   ```

Step 2: Set up the router for IPv6

1. With root authority, type the following command on the router host:

   ```
   autoconf6 -A
   ```

   This will bring up all **IPv6**-capable interfaces on the system.

   Note: To bring up a subset of interfaces, use the `-i` flag. For example, `autoconf6 -i en0 en1` will bring up interfaces `en0` and `en1`.
   Your results should look similar to the following:

   ```
   Name  Mtu   Network     Address            Ipkts Ierrs    Opkts Oerrs  Coll
   en1   1500  link#2      0.6.29.dc.15.45        0     0        7     0     0
   en1   1500  fe80::206:29ff:fedc:1545            0     0        7     0     0
   en0   1500  link#3      0.4.ac.17.b4.11        7     0       17     0     0
   en0   1500  fe80::204:acff:fe17:b411            7     0       17     0     0
   lo0   16896 link#1                           436     0      481     0     0
   lo0   16896 127         127.0.0.1            436     0      481     0     0
   lo0   16896 ::1                             436     0      481     0     0
   ```
2. Manually configure global addresses on the router's interfaces belonging to each of the two subnets by typing the following commands:

   ```
   # ifconfig en0 inet6 3ffe:0:0:aaaa::/64 eui64 alias
   # ifconfig en1 inet6 3ffe:0:0:bbbb::/64 eui64 alias
   ```

   Note: You will need to do this for every subnet that your router is sending packets to.
3. To activate **IPv6** forwarding, type the following:

   ```
   no -o ip6forwarding=1
   ```
4. To start the **ndpd-router** daemon, type the following:

   ```
   startsrc -s ndpd-router
   ```

The **ndpd-router** daemon will advertise prefixes corresponding to the global addresses that you configured on the router. In this case, the ndpd-router will advertise prefix `3ffe:0:0:aaaa::/64` on `en0` and prefix `3ffe:0:0:bbbb::/64` on `en1`.

5. Press Enter to continue.
6. Press Enter a second time to confirm your decision and begin the installation of your software bundle.

Step 3. Set up IPv6 to be configured on the hosts at boot time

After completing Step 1 for each host, **IPv6** will be deleted when you reboot the machine. To enable **IPv6** host functionality every time you reboot, do the following:

1. Open the `/etc/rc.tcpip` file using your favorite text editor.
2. Uncomment the following lines in that file:

```
# Start up autoconf6 process
start /usr/sbin/autoconf6 ""

# Start up ndpd-host daemon
start /usr/sbin/ndpd-host "$src_running"
```

3. Add the **-A** flag to `start /usr/sbin/autoconf6 ""`:

```
start /usr/sbin/autoconf6 "" -A
```

4. Repeat this process for each host.

When you reboot, **IPv6** will be automatically started.

Step 4: Set up IPv6 to be configured on the router at boot time

After completing Step 2 for your router, **IPv6** will be deleted when you reboot. To enable **IPv6** router functionality every time you reboot, do the following:

1. Open the `/etc/rc.tcpip` file in your favorite text editor.
2. Uncomment the following line in that file:

```
# Start up autoconf6 process
start /usr/sbin/autoconf6 ""
```

3. Add the **-A** flag to that line:

```
start /usr/sbin/autoconf6 "" -A
```

4. Add the following lines immediately after the line that you just uncommented in the previous step:

```
# Configure global addresses for router
ifconfig en0 inet6 3ffe:0:0:aaaa::/64 eui64 alias
ifconfig en1 inet6 3ffe:0:0:bbbb::/64 eui64 alias
```

   In this scenario, our network has only two subnets, `en0` and `en1`. You will need to add a line to this file for every subnet that your router is sending packets to.

5. Uncomment the following line in the file:

```
# Start up ndpd-router daemon
start /usr/sbin/ndpd-router "$src_running"
```

6. Run the following command to enable IP forwarding at boot time:

```
no -r -o ip6forwarding=1
```

When you reboot, **IPv6** will be automatically started.

Run-time Static Configuration:

This scenario leads you through run-time configuration of a node by using static IPs and routes.

The network that is used in this example consists of a host and a router. By the end of the scenario, an IPv6 interface is set up on the host. You first configure the machines to temporarily support IPv6 so that you can test them. Then you configure the machines so that they are IPv6-ready at boot time.

Things to consider

- The information in this how-to scenario was tested by using specific versions of AIX. The results that you obtain might vary significantly depending on your version and level of AIX.
- The example assumes **2001:1:2::/48** is the Aggregate Global Unicast Address for the IPv6 interface assigned by the Internet Assigned Numbers Authority (IANA) to the provider. And **2001:1:2:3:4::/64** is the subnet that uses bits 49 - 64 assigned by the network administrator.
- You must refer RFC 3587 to understand the IPv6 Global Unicast Address Format.

Related information:

Run-Time Configuration Commands

autoconf6 Command

Step 1. Setting up hosts for IPv6:

Follow this procedure for setting up hosts for IPv6.

1. With root authority, configure your IPv6 settings by entering the following command:

   ```
   # autoconf6
   ```

2. Rerun the following command:

   ```
   # netstat -ni
   ```

   Your results must look similar to the following output:

   ```
   Name  Mtu   Network     Address            Ipkts Ierrs    Opkts Oerrs  Coll
   en0   1500  link#2      0.6.29.4.55.ec      279679     0     2658     0     0
   en0   1500  9.3.230.64  9.3.230.117         279679     0     2658     0     0
   en0   1500  fe80::206:29ff:fe04:55ec        279679     0     2658     0     0
   sit0  1480  link#3      9.3.230.117              0     0        0     0     0
   sit0  1480  ::9.3.230.117                        0     0        0     0     0
   lo0   16896 link#1                           2343     0     2350     0     0
   lo0   16896 127         127.0.0.1           2343     0     2350     0     0
   lo0   16896 ::1                             2343     0     2350     0     0
   ```

3. Use the **chdev** command to add the IPv6 address to the host interface. For this example, the low-order 64 bits are taken from low-order 64 bits of Link-Local IP generated by **autoconf6** on interface **en0**.

   ```
   # chdev –l en0 –a netaddr6='2001:2:3:4:206:29ff:fe04:55ec' –a prefixlen=64
   ```

4. Delete any existing prefix link routes for the following prefix:

   ```
   # route delete –inet6 2001:2:3:4::/64
   ```

5. Configure prefix static route on the host to add reachability to the router, where **fe80::206:29ff:fe04:66e** is the router or a gateway that has connectivity to the router.

   ```
   # route add –inet6 –net 2001:2:3:4::/64 fe80::206:29ff:fe04:66e –static
   ```

   Note: If a change is needed for the default route, make sure **autoconf6** is run with the **–R** option that prevents it from adding or overwriting any default routes on the node. Then, repeat steps 3-5.

Step 2. Setting up the router for IPv6:

Follow this procedure for setting up the router for IPv6.

1. Check to ensure that the IPv4 settings are configured, by entering the following command:

   ```
   # netstat -ni
   ```

2. With root authority, enter the following command:

   ```
   # autoconf6
   ```

3. To activate IPv6 forwarding, enter the following command:

   ```
   # no -o ip6forwarding=1
   ```

4. Configure Global IP on the router interface, by entering the following command:

   ```
   # chdev –l en0 –a netaddr6='2001:4:5:6:207:30ff:fe05:66ec' –a prefixlen=64
   ```

5. Manually configure routes on the router to enable accurate delivery of packets. For example, if **fe80::3ca6:70ff:fe00:3004/64** is the gateway for prefix **2001:2:3:4::/64**, add a prefix route as follows:
   ```
   # route add –inet6 –net 2001:2:3:4::/64 fe80::3ca6:70ff:fe00:3004 –static
   ```

Step 3. Setting up IPv6 to be configured on hosts at every restart:

The IPv6 hosts settings configured in **Step 1. Setting up hosts for IPv6** are deleted when you restart the machine. To enable IPv6 hosts functionality every time you restart the machine, follow this procedure.

1. Open the **/etc/rc.tcpip** file in a text editor.
2. Uncomment the following line in the **/etc/rc.tcpip** file:
   ```
   # Start up autoconf6 process
   start /usr/sbin/autoconf6 ""
   ```

   Note: If the preceding line is not present in the **/etc/rc.tcpip** file, add it in the file.
3. Add the **-A** flag to **start /usr/sbin/autoconf6 ""**.
   ```
   start /usr/sbin/autoconf6 "" –A
   ```
4. Add the following line in the **/etc/rc.tcpip** file after the line you uncommented (or added):
   ```
   chdev –l en0 –a netaddr6='2001:2:3:4:206:29ff:fe04:55ec' –a prefixlen=64
   ```
5. Delete any previously existing prefix routes, by entering the following command:
   ```
   chdev -l inet0 –a delrout6='–net, 2001:2:3:4::/64'
   ```
6. Set up a route, by entering the following command:
   ```
   chdev -l inet0 –a rout6='–net, 2001:2:3:4::/64 ,fe80::206:29ff:fe04:66e,–static'
   ```

When you restart the machine, your IPv6 configuration is set.

Note: You must repeat this procedure for each host.

Step 4. Setting up IPv6 to be configured on the router at every restart:

The IPv6 router settings configured in **Step 2. Setting up the router for IPv6** are deleted when you restart the machine. To enable IPv6 router functionality every time you restart the machine, follow this procedure.

1. Open the **/etc/rc.tcpip** file in a text editor.
2. Uncomment the following line in **/etc/rc.tcpip** file:
   ```
   # Start up autoconf6 process
   start /usr/sbin/autoconf6 ""
   ```

   Note: If the preceding line is not present in the **/etc/rc.tcpip** file, add it in the file.
3. Add the **-A** flag to **start /usr/sbin/autoconf6 ""**.
   ```
   start /usr/sbin/autoconf6 "" –A
   ```
4. Add the following lines after the line that you uncommented (or added) in step 2 to configure Global IP on the router and to configure the prefix route.
   ```
   chdev –l en0 –a netaddr6='2001:4:5:6:207:30ff:fe05:66ec' –a prefixlen=64
   chdev -l inet0 –a rout6='–net,2001:2:3:4::/64,fe80::3ca6:70ff:fe00:3004,–static'
   ```

   In this scenario, the network has only one subnet, **en0**. You must add a line to this file for every subnet to which the router sends packets.

When you restart the machine, IPv6 is automatically started on the machine.

Note: When you use static configurations simultaneously with **ndpd-host** make sure various flags in **ndpd-host** are explored to retain static IPs and routes, if needed.

Setting up tunneling in IPv6:

You can use either of two methods to set up tunneling in IPv6. The first method sets up an automatic tunnel. The second method sets up a configured tunnel.

Things to Consider

- The information in this how-to scenario was tested using specific versions of AIX. The results you obtain might vary significantly depending on your version and level of AIX.

Set up an automatic tunnel in IPv6

In this scenario, the **autoconf6** command will be used to configure IPv6 and set up an automatic tunnel through the primary interface, `en2`. The **autoconf6** command will then be used to configure a tunnel through the secondary interface, `en0`.

Following is the result of the **netstat -ni** command, which displays the current network configuration of the system:

```
en0   1500  link#2      MAC address here         0     0       33     0     0
en0   1500  1.1         1.1.1.3                  0     0       33     0     0
en2   1500  link#3      MAC address here     79428     0      409     0     0
en2   1500  10.1        10.1.1.1             79428     0      409     0     0
```

- To enable IPv6 and one automatic tunnel, type the following command:

  ```
  autoconf6
  ```

  Running the **netstat -ni** command now produces the following results:

  ```
  # netstat -in
  en0   1500  link#2      MAC address here         0     0       33     0     0
  en0   1500  1.1         1.1.1.3                  0     0       33     0     0
  en0   1500  fe80::204:acff:fe49:4910              0     0       33     0     0
  en2   1500  link#3      MAC address here     79428     0      409     0     0
  en2   1500  10.1        10.1.1.1             79428     0      409     0     0
  en2   1500  fe80::220:35ff:fe12:3ae8
  sit0  1480  link#7      10.1.1.1                 0     0        0     0     0
  sit0  1480  ::10.1.1.1
  ```

  If `en2` (IP address 10.1.1.1) is the primary interface, address ::10.1.1.1 is now available for automatic tunneling over the `en2` interface.

- To enable an automatic tunnel through interface `en0`, type the following command:

  ```
  autoconf6 -s -i en0
  ```

  Running the **netstat -ni** command now produces the following results:

  ```
  # netstat -in
  en0   1500  link#2      MAC address here         0     0       33     0     0
  en0   1500  1.1         1.1.1.3                  0     0       33     0     0
  en0   1500  fe80::204:acff:fe49:4910              0     0       33     0     0
  en2   1500  link#3      MAC address here     79428     0      409     0     0
  en2   1500  10.1        10.1.1.1             79428     0      409     0     0
  en2   1500  fe80::220:35ff:fe12:3ae8
  sit0  1480  link#7      1.1.1.3                  0     0        3     0     0
  sit0  1480  ::10.1.1.1                           0     0        3     0     0
  sit0  1480  ::1.1.1.3                            0     0        3     0     0
  ```

  This action causes an IPv4-compatible IPv6 address to be added to the existing SIT interface, `sit0`. Tunneling is now also enabled for interface `en0` using address ::1.1.1.3. The same interface, `sit0`, will be used for both tunnels.

  Note: The automatic tunnels are deleted when the system is restarted. To have the automatic tunnel present at boot time, add the required arguments to the **autoconf6** command in the `/etc/rc.tcpip` file.

Set up configured tunnels

In this scenario, SMIT will be used to set up a configured tunnel. This tunnel will be available when the system restarts because it will be stored in the ODM. A tunnel will be configured between systems `alpha` and `beta`. The IPv4 address of `alpha` is 10.1.1.1, and the IPv4 address of `beta` is 10.1.1.2.

To set up configured tunnels, follow these steps:

1. To configure a tunnel between `alpha` and `beta`, type the following on both systems:

   ```
   smit ctinet6
   ```

2. Select **Add an IPV6 in IPV4 Tunnel Interface** on both systems.

   ```
   autoconf6
   ```

3. In this scenario, we filled in the values as follows on `alpha`, based on the IPv4 addresses:

   ```
   * IPV4 SOURCE ADDRESS (dotted decimal)              [10.1.1.1]
   * IPV4 DESTINATION ADDRESS (dotted decimal)         [10.1.1.2]
     IPV6 SOURCE ADDRESS (colon separated)             []
     IPV6 DESTINATION ADDRESS (colon separated)        []
   ```

   On `beta`, the following values were entered:

   ```
   * IPV4 SOURCE ADDRESS (dotted decimal)              [10.1.1.2]
   * IPV4 DESTINATION ADDRESS (dotted decimal)         [10.1.1.1]
     IPV6 SOURCE ADDRESS (colon separated)             []
     IPV6 DESTINATION ADDRESS (colon separated)        []
   ```

4. To view the configured interfaces, type the following command:

   ```
   ifconfig ctiX
   ```

   where *X* is the number of the interface. In this scenario, the following results were returned. On `alpha`:

   ```
   cti0: flags=8080051<UP,POINTOPOINT,RUNNING,MULTICAST>
           inet6 fe80::a01:101/128 --> fe80::a01:102
   ```

   On `beta`:

   ```
   cti0: flags=8080051 <UP,POINTOPOINT,RUNNING,MULTICAST>
           inet6 fe80::a01:102/128 --> fe80::a01:101
   ```

SMIT automatically creates IPv6 addresses for both ends of the tunnel using the following method:

- The lower 32 bits contain the IPv4 address
- The upper 96 bits contain the prefix `fe80::/96`

You can fill in specific IPv6 addresses if desired.

Packet tracing

Packet tracing is the process by which you can verify the path of a packet through the layers to its destination.

The **iptrace** command performs network interface level packet tracing. The **ipreport** command issues output on the packet trace in both hexadecimal and ASCII format. The **trpt** command performs transport protocol level packet tracking for the **TCP**. The **trpt** command output is more detailed, including information on time, **TCP** state, and packet sequencing.

Network Interface packet headers

At the Network Interface layer, packet headers are attached to outgoing data.

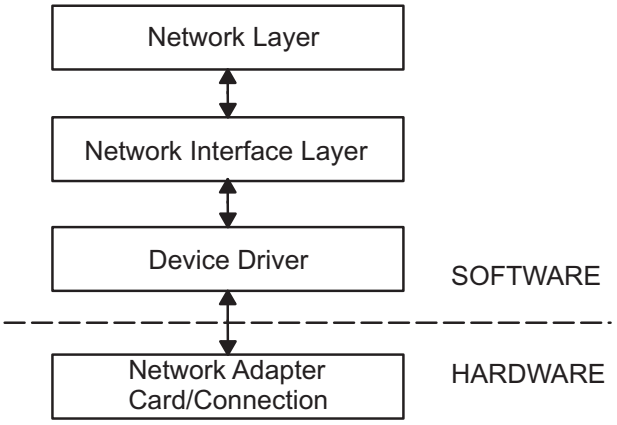


Figure 8. Packet flow through Network Interface Structure

This illustration shows bi-directional data flow through the layers of the Network Interface Structure. From the top (software) they are the Network Layer, Network Interface Layer, Device Driver, and the (hardware) Network Adapter Card or Connection.

Packets are then sent through the network adapter to the appropriate network. Packets can pass through many gateways before reaching their destinations. At the destination network, the headers are stripped from the packets and the data is sent to the appropriate host.

The following section contains packet header information for several of the more common network interfaces.

Ethernet adapter frame headers:

An **Internet Protocol** (**IP**) or **Address Resolution Protocol** (**ARP**) frame header for the Ethernet adapter is composed of these three fields.

Table 55. Ethernet adapter frame header

Field	Length	Definition
DA	6 bytes	Destination address.
SA	6 bytes	Source address. If bit 0 of this field is set to 1, it indicates that routing information (RI) is present.
Type	2 bytes	Specifies whether the packet is **IP** or **ARP**. The type number values are listed below.

Type field numbers:

Item	Description
IP	0800
ARP	0806

Token-Ring frame headers:

There are five fields that comprise the medium access control (MAC) header for the token-ring adapter.

Table 56. Token-ring MAC header

Field	Length	Definition
AC	1 byte	Access control. The value in this field x`00' gives the header priority 0.
FC	1 byte	Field control. The value in this field x`40' specifies the Logical Link Control frame.
DA	6 bytes	Destination address.
SA	6 bytes	Source address. If bit 0 of this field is set to 1, it indicates that routing information (RI) is present.
RI	18 bytes	Routing information. The valid fields are discussed below.

The MAC header consists of two routing information fields of two bytes each: routing control (RC) and segment numbers. A maximum of eight segment numbers can be used to specify recipients of a limited broadcast. RC information is contained in bytes 0 and 1 of the RI field. The settings of the first two bits of the RC field have the following meanings:

Item	Description
bit (0) = 0	Use the nonbroadcast route specified in the RI field.
bit (0) = 1	Create the RI field and broadcast to all rings.
bit (1) = 0	Broadcast through all bridges.
bit (1) = 1	Broadcast through limited bridges.

The logical link control (LLC) header is composed of five fields, as shown in the following LLC header table.

Table 57. 802.3 LLC header

Field	Length	Definition
DSAP	1 byte	Destination service access point. The value in this field is x`aa'.
SSAP	1 byte	Source service access point. The value in this field is x`aa'.
CONTROL	1 byte	Determines the LLC commands and responses. The three possible values for this field are discussed below.
PROT_ID	3 bytes	Protocol ID. This field is reserved. It has a value of x`0'.
TYPE	2 bytes	Specifies whether the packet is **IP** or **ARP**.

Control field values:

The token-ring control fields include an unnmubered information frame, an exchange identification frame, and a test frame. Their values are described here.

Item	Description
x`03'	Unnumbered Information (UI) frame. This is the normal, or unsequenced, way in which token-ring adapter data is transmitted through the network. **TCP/IP** sequences the data.
x`AF'	Exchange identification (XID) frame. This frame conveys the characteristics of the sending host.
x`E3'	Test frame. This frame supports testing of the transmission path, echoing back the data that is received.

802.3 frame headers:

The MAC header for the 802.3 adapter is composed of two fields, as shown in this MAC header table.

Table 58. 802.3 MAC header

Field	Length	Definition
DA	6 bytes	Destination address.
SA	6 bytes	Source address. If bit 0 of this field is set to 1, it indicates that routing information (RI) is present.

The LLC header for 802.3 is the same as for Token-Ring MAC header.

Internet network-level protocols

The Internet network-level protocols handle machine-to-machine communication.

In other words, this layer implements **TCP/IP** routing. These protocols accept requests to send packets (along with the network address of the destination machine) from the Transport layer, convert the packets to datagram format, and send them down to the Network Interface layer for further processing.

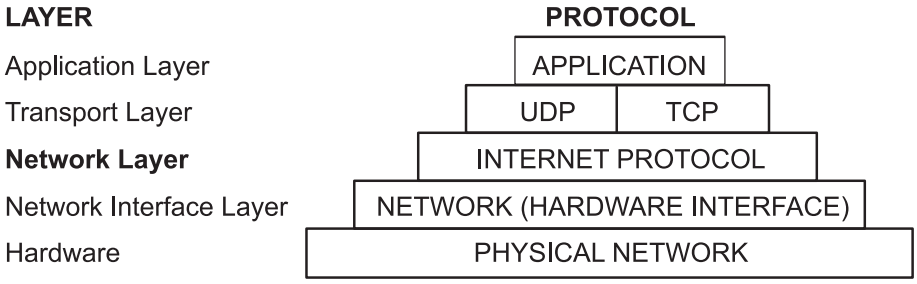


Figure 9. Network layer of the TCP/IP Suite of Protocols

This illustration shows the various layers of the **TCP/IP** Suite of Protocols. From the top, the application layer consists of the application. The transport layer contains **UDP** and **TCP**. The network layer contains the network (hardware) interface. And finally, the hardware layer contains the physical network.

TCP/IP provides the protocols that are required to comply with RFC 1100, *Official Internet Protocols*, as well as other protocols commonly used by hosts in the Internet community.

Note: The use of Internet network, version, socket, service, and protocol numbers in **TCP/IP** also complies with RFC 1010, *Assigned Numbers*.

Address Resolution Protocol:

The first network-level protocol is the **Address Resolution Protocol** (**ARP**). **ARP** dynamically translates Internet addresses into the unique hardware addresses on local area networks.

To illustrate how **ARP** works, consider two nodes, `X` and `Y`. If node `X` wishes to communicate with `Y`, and `X` and `Y` are on different local area networks (LANs), `X` and `Y` communicate through *bridges*, *routers*, or *gateways*, using IP addresses. Within a LAN, nodes communicate using low-level hardware addresses.

Nodes on the same segment of the same LAN use **ARP** to determine the hardware address of other nodes. First, node `X` broadcasts an **ARP** request for node `Y`'s hardware address. The **ARP** request contains `X`'s **IP** and hardware addresses, and `Y`'s **IP** address. When `Y` receives the **ARP** request, it places an entry for `X` in its **ARP** cache (which is used to map quickly from IP address to hardware address), then responds directly to `X` with an **ARP** response containing `Y`'s **IP** and hardware addresses. When node `X` receives `Y`'s **ARP** response, it places an entry for `Y` in its **ARP** cache.

Once an **ARP** cache entry exists at `X` for `Y`, node `X` is able to send packets directly to `Y` without resorting again to **ARP** (unless the **ARP** cache entry for `Y` is deleted, in which case **ARP** is reused to contact `Y`).

Unlike most protocols, **ARP** packets do not have fixed-format headers. Instead, the message is designed to be useful with a variety of network technologies, such as:

- Ethernet LAN adapter (supports both Ethernet and 802.3 protocols)
- Token-ring network adapter
- Fiber Distributed Data Interface (FDDI) network adapter

However, ARP does not translate addresses for **Serial Line Interface Protocol** (**SLIP**) or **Serial Optical Channel Converter** (**SOC**), since these are point-to-point connections.

The kernel maintains the translation tables, and the **ARP** is not directly available to users or applications. When an application sends an Internet packet to one of the interface drivers, the driver requests the appropriate address mapping. If the mapping is not in the table, an **ARP** broadcast packet is sent through the requesting interface driver to the hosts on the local area network.

Entries in the **ARP** mapping table are deleted after 20 minutes; incomplete entries are deleted after 3 minutes. To make a permanent entry in the **ARP** mapping tables, use the **arp** command with the *pub* parameter:

```
arp -s 802.3 host2 0:dd:0:a:8s:0 pub
```

When any host that supports **ARP** receives an **ARP** request packet, the host notes the **IP** and hardware addresses of the requesting system and updates its mapping table, if necessary. If the receiving host **IP** address does not match the requested address, the host discards the request packet. If the **IP** address does match, the receiving host sends a response packet to the requesting system. The requesting system stores the new mapping and uses it to transmit any similar pending Internet packets.

Internet Control Message Protocol:

The second network-level protocol is the **Internet Control Message Protocol** (**ICMP**). **ICMP** is a required part of every **IP** implementation. **ICMP** handles error and control messages for **IP**.

This protocol allows gateways and hosts to send problem reports to the machine sending a packet. **ICMP** does the following:

- Tests whether a destination is alive and reachable
- Reports parameter problems with a datagram header
- Performs clock synchronization and transit time estimations
- Obtains Internet addresses and subnet masks

Note: **ICMP** uses the basic support of **IP** as if it were a higher-level protocol. However, **ICMP** is actually an integral part of **IP** and must be implemented by every **IP** module.

ICMP provides feedback about problems in the communications environment, but does not make **IP** reliable. That is, **ICMP** does not guarantee that an **IP** packet is delivered reliably or that an **ICMP** message is returned to the source host when an **IP** packet is not delivered or is incorrectly delivered.

ICMP messages might be sent in any of the following situations:

- When a packet cannot reach its destination
- When a gateway host does not have the buffering capacity to forward a packet
- When a gateway can direct a host to send traffic on a shorter route

TCP/IP sends and receives several ICMP message types (see "Internet Control Message Protocol message types"). **ICMP** is embedded in the kernel, and no application programming interface (API) is provided to this protocol.

Internet Control Message Protocol message types:

ICMP sends and receives these message types.

Item	Description
echo request	Sent by hosts and gateways to test whether a destination is alive and reachable.
information request	Sent by hosts and gateways to obtain an Internet address for a network to which they are attached. This message type is sent with the network portion of **IP** destination address set to a value of 0.
timestamp request	Sent to request that the destination machine return its current value for time of day.
address mask request	Sent by host to learn its subnet mask. The host can either send to a gateway, if it knows the gateway address, or send a broadcast message.
destination unreachable	Sent when a gateway cannot deliver an **IP** datagram.
source quench	Sent by discarding machine when datagrams arrive too quickly for a gateway or host to process, in order to request that the original source slow down its rate of sending datagrams.
redirect message	Sent when a gateway detects that some host is using a nonoptimum route.
echo reply	Sent by any machine that receives an echo request in reply to the machine which sent the request.
information reply	Sent by gateways in response to requests for network addresses, with both the source and destination fields of the **IP** datagram specified.
timestamp reply	Sent with current value of time of day.
address mask reply	Sent to machines requesting subnet masks.
parameter problem	Sent when a host or gateway finds a problem with a datagram header.
time exceeded	Sent when the following are true: • Each **IP** datagram contains a time-to-live counter (hop count), which is decremented by each gateway. • A gateway discards a datagram because its hop count has reached a value of 0.
Internet Timestamp	Used to record the time stamps through the route.

Internet Protocol:

The third network-level protocol is the **Internet Protocol** (**IP**), which provides unreliable, connectionless packet delivery for the Internet.

IP is connectionless because it treats each packet of information independently. It is unreliable because it does not guarantee delivery, meaning, it does not require acknowledgments from the sending host, the receiving host, or intermediate hosts.

IP provides the interface to the network interface level protocols. The physical connections of a network transfer information in a frame with a header and data. The header contains the source address and the

destination address. **IP** uses an Internet datagram that contains information similar to the physical frame. The datagram also has a header containing Internet Protocol addresses of both source and destination of the data.

IP defines the format of all the data sent over the Internet.

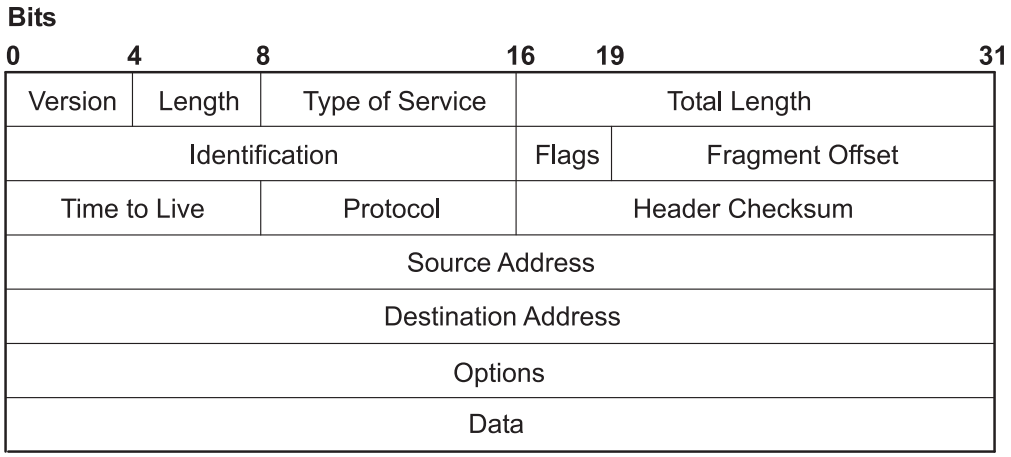


Figure 10. Internet Protocol packet header

This illustration shows the first 32 bits of a typical IP packet header. Table below lists the various entities.

IP header field definitions

Item	Description
Version	Specifies the version of the **IP** used. The current version of the **IP** protocol is 4.
Length	Specifies the datagram header length, measured in 32-bit words.
Type of Service	Contains five subfields that specify the type of precedence, delay, throughput, and reliability desired for that packet. (The Internet does not guarantee this request.) The default settings for these five subfields are routine precedence, normal delay, normal throughput, and normal reliability. This field is not generally used by the Internet at this time. This implementation of **IP** complies with the requirements of the **IP** specification, RFC 791, *Internet Protocol*.
Total Length	Specifies the length of the datagram including both the header and the data measured in octets. Packet fragmentation at gateways, with reassembly at destinations, is provided. The total length of the **IP** packet can be configured on an interface-by-interface basis with the **ifconfig** command, or the System Management Interface Tool (SMIT) fast path, `smit chinet`. Use SMIT to set the values permanently in the configuration database; use the **ifconfig** command to set or change the values in the running system.
Identification	Contains a unique integer that identifies the datagram.
Flags	Controls datagram fragmentation, along with the Identification field. The Fragment Flags specify whether the datagram can be fragmented and whether the current fragment is the last one.
Fragment Offset	Specifies the offset of this fragment in the original datagram measured in units of 8 octets.
Time to Live	Specifies how long the datagram can remain on the Internet. This keeps misrouted datagrams from remaining on the Internet indefinitely. The default time to live is 255 seconds.
Protocol	Specifies the high-level protocol type.
Header Checksum	Indicates a number computed to ensure the integrity of header values.
Source Address	Specifies the Internet address of the sending host.
Destination Address	Specifies the Internet address of the receiving host.

Item	Description
Options	Provides network testing and debugging. This field is not required for every datagram. **End of Option List** Indicates the end of the option list. It is used at the end of the final option, not at the end of each option individually. This option should be used only if the end of the options would not otherwise coincide with the end of the **IP** header. End of Option List is used if options exceed the length of the datagram. **No Operation** Provides alignment between other options; for example, to align the beginning of a subsequent option on a 32-bit boundary. **Loose Source and Record Route** Provides a means for the source of an Internet datagram to supply routing information used by the gateways in forwarding the datagram to a destination and in recording the route information. This is a *loose* source route: the gateway or host **IP** is allowed to use any route of any number of other intermediate gateways in order to reach the next address in the route. **Strict Source and Record Route** Provides a means for the source of an Internet datagram to supply routing information used by the gateways in forwarding the datagram to a destination and in recording the route information. This is a *strict* source route: In order to reach the next gateway or host specified in the route, the gateway or host **IP** must send the datagram directly to the next address in the source route and only to the directly connected network that is indicated in the next address. **Record Route** Provides a means to record the route of an Internet datagram. **Stream Identifier** Provides a way for a stream identifier to be carried through networks that do not support the stream concept. **Internet Timestamp** Provides a record of the time stamps through the route.

Outgoing packets automatically have an **IP** header prefixed to them. Incoming packets have their **IP** header removed before being sent to the higher-level protocols. The **IP** protocol provides for the universal addressing of hosts in the Internet network.

Internet Transport-Level Protocols

The **TCP/IP** transport-level protocols allow application programs to communicate with other application programs.

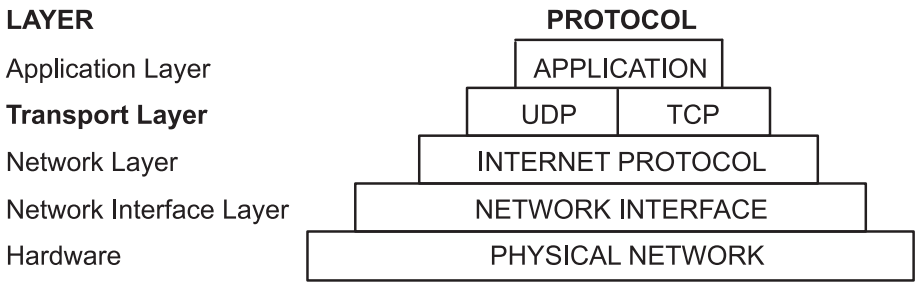


Figure 11. Transport layer of the TCP/IP Suite of Protocols.

This illustration shows the various layers of the **TCP/IP** Suite of Protocols. From the top, the application layer consists of the application. The transport layer contains **UDP** and **TCP**. The network layer contains the network (hardware) interface. And finally, the hardware layer contains the physical network.

User Datagram Protocol (**UDP**) and the **TCP** are the basic transport-level protocols for making connections between Internet hosts. Both **TCP** and **UDP** allow programs to send messages to and receive messages from applications on other hosts. When an application sends a request to the Transport layer to send a message, **UDP** and **TCP** break the information into packets, add a packet header including the destination address, and send the information to the Network layer for further processing. Both **TCP** and **UDP** use protocol ports on the host to identify the specific destination of the message.

Higher-level protocols and applications use **UDP** to make datagram connections and **TCP** to make stream connections. The operating system sockets interface implements these protocols.

User Datagram Protocol:

Sometimes an application on a network needs to send messages to a specific application or process on another network. The **UDP** provides a datagram means of communication between applications on Internet hosts.

Because senders do not know which processes are active at any given moment, **UDP** uses destination protocol ports (or abstract destination points within a machine), identified by positive integers, to send messages to one of multiple destinations on a host. The protocol ports receive and hold messages in queues until applications on the receiving network can retrieve them.

Because **UDP** relies on the underlying **IP** to send its datagrams, **UDP** provides the same connectionless message delivery as **IP**. It offers no assurance of datagram delivery or duplication protection. However, **UDP** does allow the sender to specify source and destination port numbers for the message and calculates a checksum of both the data and header. These two features allow the sending and receiving applications to ensure the correct delivery of a message.

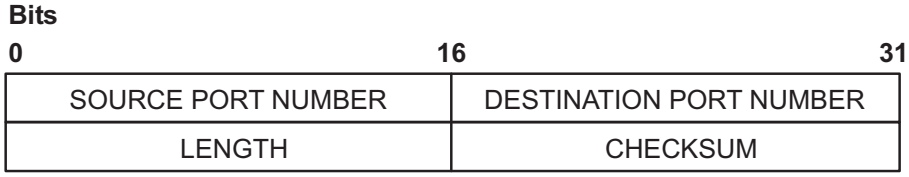


Figure 12. User Datagram Protocol (UDP) packet header

This illustration shows the first 32 bits of the **UDP** packet header. The first 16 bits contain the source port number and the length. The second 16 bits contain the destination port number and the checksum.

Applications that require reliable delivery of datagrams must implement their own reliability checks when using **UDP**. Applications that require reliable delivery of streams of data should use **TCP**.

UDP Header Field Definitions

Item	Description
Source Port Number	Address of the protocol port sending the information.
Destination Port Number	Address of the protocol port receiving the information.
Length	Length in octets of the **UDP** datagram.
Checksum	Provides a check on the **UDP** datagram using the same algorithm as the **IP**.

The applications programming interface (API) to **UDP** is a set of library subroutines provided by the sockets interface.

Reliable Datagram Sockets over InfiniBand and RoCE:

Reliable Datagram Sockets (RDS) is a connectionless and record-oriented protocol that provides an in-order and no-duplicate service over InfiniBand and RDMA over Converged Ethernet (RoCE). RDS exposes the User Datagram Protocol (UDP) subset of the socket API.

The RDS is part of the **AF_BYPASS** domain that is used for protocols that bypass the kernel TCP/IP stack.

The AIX operating system provides two versions of RDS: RDSv2 and RDSv3. RDSv3 is the latest version and includes support for Remote Direct Memory Access (RDMA). RDSv3 on AIX 7.2, and later, supports Open Fabrics Enterprise Distribution (OFED) based RDMA over Converged Ethernet (RoCE).

Creating an RDS socket: To create an RDS socket, invoke the **socket()** system call by adding the following lines to the application program:

```
#include <sys/bypass.h>
include <net/rds_rdma.h>           /* for RDSv3 only */
sock = socket (AF_BYPASS, SOCK_SEQPACKET,BYPASSPROTO_RDS);
```

If the **BYPASSPROTO_RDS** protocol is the only reliable datagram protocol that is supported in the **AF_BYPASS** family, you can also call the **socket()** system call as follows:

```
sock = socket (AF_BYPASS, SOCK_SEQPACKET,0);
```

System calls

The RDS also supports the following system calls:

- blind()
- close()
- getsockopt()
- recvform()
- recvmsg()
- sendmsg()
- sendto()
- setsockopt()

In addition, RDSv3 also supports the following system calls:

- connect()
- read()
- recv()
- send()
- write()

Note: Although RDS sockets are connectionless, the **connect()** system call is supported by RDSv3. However, in this case, **connect()** does not create a socket-level connection entity between two RDS endpoints. It merely associates a default destination endpoint with the socket. For this reason, the **listen()**, **accept()**, and **shutdown()** system calls are not supported for the RDS sockets.

The rdsctrl utility for RDSv2: Use the **rdsctrl** utility (`/usr/sbin/rdsctrl`) to change the tunables and the diagnostics for RDS statistics. For RDSv2, the utility can be used after RDS is loaded (**bypassctrl load rds**). For more information for this utility, run the rdsctrl command with no arguments.

Statistics

To display various RDS statistics, run the `# rdsctrl stats` command.

To reset the statistics, run the `# rdsctrl stats reset` command.

Tuning Parameters

The following RDS parameters can be tuned after RDS is loaded, but before an RDS application is run:

rds_sendspace
Specifies the high-water mark of the per-flow send buffer. Each socket might have multiple flow. The default value is 524288 bytes (512 KB). The value is set by using the following command: `# rdsctrl set rds_sendspace= <value in bytes>`.

rds_recvspace
Specifies the per-flow high-water mark of the per-socket receive buffer. For every additional flow to this socket, the **receive high-water** mark is increased by this value. The default value is 524288 bytes (512 KB). The value is set by using the following command: `# rdsctrl set rds_recvspace= <value in bytes>`.

Note: For increased RDS streaming performance, the values of the **rds_sendspace** parameter and the **rds_recvspace** parameter must be at least the value of the largest RDS **sendmsg()** size, multiplied by four. RDS sends an ACK for each set of four messages that are received. If the **rds_recvspace** is not at least four times larger than the message size, the throughput is very low.

rds_mclustsize
Specifies the size of the individual memory cluster, which is also the message fragment size. The default size is 16384 bytes (16 KB). The value, always a multiple of 4096, is set by using the following command: `# rdsctrl set rds_mclustsize= <multiple of 4096, in bytes>`.

Attention: The **rds_mclustsize** value must be the same on all systems (nodes) in the cluster. Changing this value also has performance implications.

The current values for the preceding parameters can be retrieved by using the `# rdsctrl get <parameter>` command.

To get the list of all tunables and their values, run the `# rdsctrl get` command.

The rdsctrl utility for RDSv3: For RDSv3, the **rdsctrl** command supports its options. These options are listed here:

Item	Description
help [<*tunable name*>]	The **help** option displays a descriptive message of the specified RDSv3 tunable. If no tunable is specified, this option displays the list of all the tunables that are supported for RDSv3, along with the description of each tunable.
set [-p] {<*tunable name*> = <*value*>}	The **set** option sets the value of the specified RDSv3 tunable. It verifies that the user has the required privileges to prevent unauthorized users to change the RDS tunables. It also does range validation for the new tunable values. The **-p** flag makes the assignment permanent across reboot operations.
get [<*tunable name*>]	The **get** option gets the current value of the queried tunable. When no name field is specified to this command, it returns the current value of all the available RDS tunables.
default [-p] [<*tunable name*>]	The **default** option is used to reset a tunable to its default value. When the name field is specified, only that tunable is reset. If no name field is specified, this command resets all the tunables to their default values. This option also provides a way to make the change permanent across reboots by using the **-p** flag.

Item	Description
load [`ofed` | `aixib`]	The **load** option loads the RDSv3 kernel extension (if it is not already loaded). The `ofed` argument loads the kernel extension in RDSv3 on OFED verbs in RoCE mode. The `aixib` argument loads the kernel extension in RDSv3 in InfiniBand mode. Specifying an argument for the **load** option is optional. The **load** option defaults to the `aixib` argument when the argument is not specified. By default, the **rdsctrl** utility loads the InfiniBand device unless the new attribute (`ofed`) is specified at the command line.
unload	The **unload** option is used to unload the RDSv3 kernel extension.
ras [-p] <*minimal* | *normal* | *detail* | *maximal*>	The **ras** option sets the AIX operating system RAS tracing and error checking settings for RDSv3 to the specified level. Internally, this command calls the **errctrl** and **ctctrl** AIX operating system commands. The **-p** flag makes the settings persistent across reboot operations.
ras extract	The **ras extract** option dumps the contents of the RAS error and non-error trace buffers for RDS to standard output.
info [<*flags*>]	The **info** option is an alias for the **rds-info** command.
ping [<*IP v4 address*>]	The **ping** option is an alias for the **rds-ping** command.
conn <*restart* | *kill*> <*source IP address*> <*destination IP address*>	The **conn** option restarts the specified RDS connection (**restart** suboption) or permanently ends the specified RDS connection (**kill** suboption). The RDS connection to be restarted or ended is specified by giving the IP addresses of the local and remote nodes for the connection. Restarting a connection drops the underlying InfiniBand connection and attempts to establish the connection again. In contrast, ending a connection (kill suboption) drops the underlying InfiniBand connection and deallocates all resources that are associated with the corresponding RDS connection.
trace start <*trace file path*> <*maximum data captured per RDS fragment*>	The **trace start** option initiates a tracing session to capture over-the-wire traffic for the RDSv3 protocol. The RDSv3 messages are transmitted in fragments. Each RDS fragment that is transmitted or received is captured as a trace packet in the specified trace file. For each RDS fragment, its payload is captured up to <*maximum data captured per RDS fragment*> bytes. Only privileged users can trace RDS traffic and only one tracing session can be active at a time.
trace stop	The **trace stop** option ends a tracing session that was previously initiated by a **trace start** command. It closes the trace file that is associated with the tracing session. After this command, the **trace report** command can be used to generate a text report of the trace file.
trace report <*trace file path*>	The **trace report** option prints a text report to standard output, from a previously captured RDS protocol trace file.
version	The **version** option prints the RDS protocol version that is currently loaded in the system.

RDSv3 tunables: To see the list of tunables that are supported for RDSv3, run the command **rdsctrl help** with no arguments.

RDMA API (RDSv3 only): The programming model for working on RDMA with RDS sockets is based on the client/server model. The RDMA client is the application that initiates an RDMA read or write operation from a specified RDMA server. The RDMA server is the application that processes the RDMA data transfer. An RDMA read operation is a data transfer from the client's address space to the server's address space, whereas an RDMA write operation is a data transfer from the server's address space to the client's address space. In either case, data is transferred directly between user-space memory on both sides, without being copied to kernel-space memory on either side.

An RDMA client application can initiate an RDMA read or write operation by sending an application-level request, along with an RDMA cookie, to an RDMA server application. The application-level request must specify whether the operation is an RDMA read or write operation as well as the address and length of the area of the client's memory to be remotely read or written by the RDMA server.

There are two methods for sending an RDMA request from the RDMA client to the RDMA server.

The first method is to send an **RDS_CMSG_RDMA_MAP** control message (carrying an **rds_get_mr_args** structure) along with the application-level RDMA request by using the **sendmsg()** system call on an RDS socket. The AIX operating system kernel at the client side processes the **RDS_CMSG_RDMA_MAP** control message by mapping the specified area of local memory (from the client application's address space), for DMA access, and generating an RDMA cookie. Then, the application-level request is sent to the server along with the RDMA cookie.

The second method consists of two steps. The first step is to call the **setsockopt()** system call with the **RDS_GET_MR** socket option, passing an **rds_get_mr_args** structure. This call maps the specified area of local memory for DMA access, and returns an RDMA cookie. The second step is to send an **RDS_CMSG_RDMA_DEST** control message (carrying the RDMA cookie that is obtained from the first step) along with the application-level RDMA request by using the **sendmsg()** system call.

The first method, which requires one system call, is preferred over the second method, which requires two system calls.

When the RDMA server application receives the application-level **RDMA read** request from the client, it also receives an **RDS_CMSG_RDMA_DEST** control message (carrying the RDMA cookie from the client). Then, the server initiates the **RDMA read** operation, by sending an application-level reply to the client along with an **RDS_CMSG_RDMA_ARGS** control message (carrying a **rds_rdma_args** structure). The AIX operating system kernel at the server side processes the **RDS_CMSG_RDMA_ARGS** control message, by mapping the specified area of local memory (from the server application's address space), for DMA access, and by physically starting the RDMA read operation. The RDMA read operation is performed by the server-side InfiniBand adapter, which interacts with the client-side InfiniBand adapter, to do the data transfer directly from the client application's memory to the server application's memory, without further software intervention. After the RDMA read operation is completed, the server-side adapter sends the application-level reply to the client. This is how the client application knows that its RDMA read operation has been completed.

Note: An RDMA operation is requested by the client by using an **RDS_CMSG_RDMA_MAP** control in which the **RDS_RDMA_USE_ONCE** flag is set. For this request, the area of memory that is mapped for DMA in the client's address space of memory is automatically unmapped for DMA, when the client receives the application-level reply from the server.

Although this implicit DMA mapping or unmapping mechanism makes it simpler to write RDMA applications, developers must be aware that registering memory for DMA on the AIX operating system is an expensive operation. Thus, if the same area of memory is going to be accessed by using RDMA multiple times, it is more efficient to do the DMA registration only the first time. To do this activity, a client application needs to use an **RDS_CMSG_RDMA_MAP** control message without the **RDS_RDMA_USE_ONCE** flag set when sending the RDMA request to the server. Then, subsequent RDMA transfers to the same area of the client's memory can be initiated by the RDMA server application without the client needing to send another request to the server. At the end, the client application would need to explicitly unmap the DMA-mapped memory by using the **setsockopt()** system call with the **RDS_FREE_MR** socket option.

RDS-specific socket options are specified by using **SOL_RDS** as the level parameter for the **setsockopt()** or **getsockopt()** system call.

Transmission Control Protocol:

TCP provides reliable stream delivery of data between Internet hosts.

Like **UDP**, **TCP** uses Internet Protocol, the underlying protocol, to transport datagrams, and supports the block transmission of a continuous stream of datagrams between process ports. Unlike **UDP**, **TCP**

provides reliable message delivery. **TCP** ensures that data is not damaged, lost, duplicated, or delivered out of order to a receiving process. This assurance of transport reliability keeps applications programmers from having to build communications safeguards into their software.

The following are operational characteristics of **TCP**:

Item	Description
Basic Data Transfer	**TCP** can transfer a continuous stream of 8-bit octets in each direction between its users by packaging some number of bytes into segments for transmission through the Internet system. **TCP** implementation allows a segment size of at least 1024 bytes. In general, **TCP** decides when to block and forward packets at its own convenience.
Reliability	**TCP** must recover data that is damaged, lost, duplicated, or delivered out of order by the Internet. **TCP** achieves this reliability by assigning a sequence number to each octet it transmits and requiring a positive acknowledgment (ACK) from the receiving **TCP**. If the ACK is not received within the time-out interval, the data is retransmitted. The **TCP** retransmission time-out value is dynamically determined for each connection, based on round-trip time. At the receiver, the sequence numbers are used to correctly order segments that may be received out of order and to eliminate duplicates. Damage is handled by adding a checksum to each segment transmitted, checking it at the receiver, and discarding damaged segments.
Flow Control	**TCP** governs the amount of data sent by returning a window with every ACK to indicate a range of acceptable sequence numbers beyond the last segment successfully received. The window indicates an allowed number of octets that the sender may transmit before receiving further permission.
Multiplexing	**TCP** allows many processes within a single host to use **TCP** communications facilities simultaneously. **TCP** receives a set of addresses of ports within each host. **TCP** combines the port number with the network address and the host address to uniquely identify each socket. A pair of sockets uniquely identifies each connection.
Connections	**TCP** must initialize and maintain certain status information for each data stream. The combination of this information, including sockets, sequence numbers, and window sizes, is called a connection. Each connection is uniquely specified by a pair of sockets identifying its two sides.
Precedence and Security	Users of **TCP** may indicate the security and precedence of their communications. Default values are used when these features are not needed.

The **TCP Packet Header** figure illustrates these characteristics.

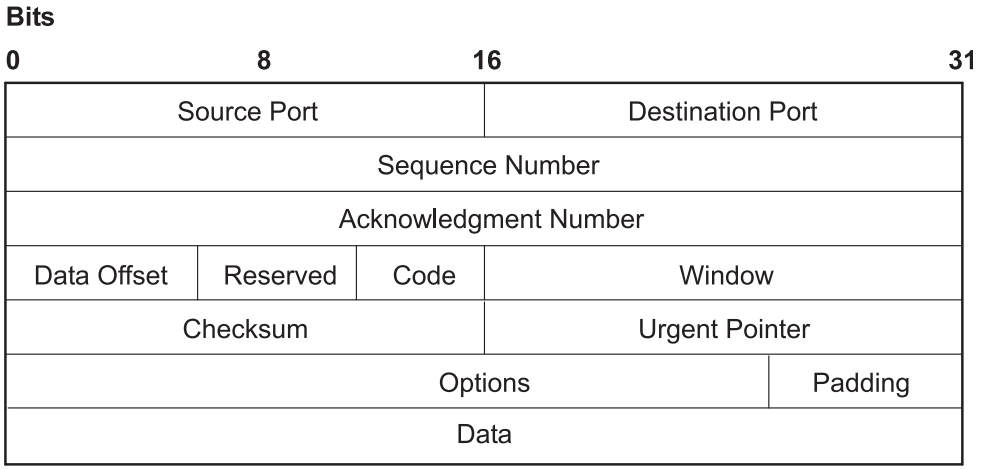


Figure 13. Transmission Control Protocol (TCP) packet header

This illustration shows what is contained in the **TCP** packet header. The individual entities are listed in the text below.

TCP header field definitions:

Short descriptions of each of the **Transmission Control Protocol** (**TCP**) fields follow.

Item	Description
Source Port	Identifies the port number of a source application program.
Destination Port	Identifies the port number of a destination application program.
Sequence Number	Specifies the sequence number of the first byte of data in this segment.
Acknowledgment Number	Identifies the position of the highest byte received.
Data Offset	Specifies the offset of data portion of the segment.
Reserved	Reserved for future use.
Code	Control bits to identify the purpose of the segment: **URG** Urgent pointer field is valid. **ACK** Acknowledgement field is valid. **PSH** Segment requests a PUSH. **RTS** Resets the connection. **SYN** Synchronizes the sequence numbers. **FIN** Sender has reached the end of its byte stream.
Window	Specifies the amount of data the destination is willing to accept.
Checksum	Verifies the integrity of the segment header and data.
Urgent Pointer	Indicates data that is to be delivered as quickly as possible. This pointer specifies the position where urgent data ends.
Options	**End of Option List** Indicates the end of the option list. It is used at the final option, not at the end of each option individually. This option needs to be used only if the end of the options would not otherwise coincide with the end of the **TCP** header. **No Operation** Indicates boundaries between options. Can be used between other options; for example, to align the beginning of a subsequent option on a word boundary. There is no guarantee that senders will use this option, so receivers must be prepared to process options even if they do not begin on a word boundary. **Maximum Segment Size** Indicates the maximum segment size **TCP** can receive. This is only sent in the initial connection request.

The applications programming interface to **TCP** consists of a set of library subroutines provided by the sockets interface.

Internet Application-Level Protocols

TCP/IP implements higher-level Internet protocols at the application program level.

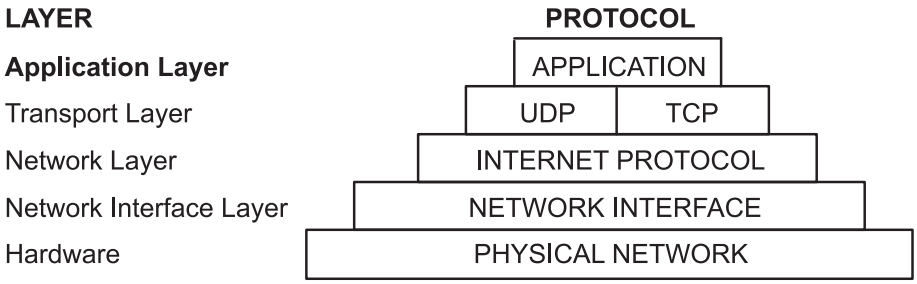


Figure 14. Application layer of the TCP/IP Suite of Protocols

This illustration shows the various layers of the **TCP/IP** Suite of Protocols. From the top, the application layer consists of the application. The transport layer contains **UDP** and **TCP**. The network layer contains the network (hardware) interface. And finally, the hardware layer contains the physical network.

When an application needs to send data to another application on another host, the applications send the information down to the transport level protocols to prepare the information for transmission.

The official Internet application-level protocols include:

- **Domain Name Protocol**
- **Exterior Gateway Protocol**
- **File Transfer Protocol**
- **Name/Finger Protocol**
- **Telnet Protocol**
- **Trivial File Transfer Protocol**

TCP/IP implements other higher-level protocols that are not official Internet protocols but are commonly used in the Internet community at the application program level. These protocols include:

- Distributed Computer Network (DCN) **Local-Network Protocol**
- **Remote Command Execution Protocol**
- **Remote Login Protocol**
- **Remote Shell Protocol**
- **Wake On LAN Protocol**
- **Routing Information Protocol**
- **Time Server Protocol**

TCP/IP does not provide APIs to any of these application-level protocols.

Domain Name Protocol:

The **Domain Name Protocol** (**DOMAIN**) allows a host in a domain to act as a *name server* for other hosts within the domain.

DOMAIN uses **UDP** or **TCP** as its underlying protocol and allows a local network to assign host names within its domain independently from other domains. Normally, the **DOMAIN** protocol uses **UDP**. However, if the **UDP** response is truncated, **TCP** can be used. The **DOMAIN** protocol in **TCP/IP** supports both.

In the **DOMAIN** hierarchical naming system, local resolver routines can resolve Internet names and addresses using a local name resolution database maintained by the **named** daemon. If the name

requested by the host is not in the local database, the resolver routine queries a remote **DOMAIN** name server. In either case, if the name resolution information is unavailable, the resolver routines attempt to use the `/etc/hosts` file for name resolution.

Note: **TCP/IP** configures local resolver routines for the **DOMAIN** protocol if the local file `/etc/resolv.conf` exists. If this file does not exist, the **TCP/IP** configures the local resolver routines to use the `/etc/hosts` database.

TCP/IP implements the **DOMAIN** protocol in the **named** daemon and in the resolver routines and does not provide an API to this protocol.

Exterior Gateway Protocol:

Exterior Gateway Protocol (**EGP**) is the mechanism that allows the exterior gateway of an *autonomous system* to share routing information with exterior gateways on other autonomous systems.

Autonomous systems:

An autonomous system is a group of networks and gateways for which one administrative authority has responsibility.

Gateways are *interior neighbors* if they reside on the same autonomous system and *exterior neighbors* if they reside on different autonomous systems. Gateways that exchange routing information using **EGP** are said to be **EGP** *peers* or *neighbors*. Autonomous system gateways use **EGP** to provide access information to their **EGP** neighbors.

EGP allows an exterior gateway to ask another exterior gateway to agree to exchange access information, continually checks to ensure that its **EGP** neighbors are responding, and helps **EGP** neighbors to exchange access information by passing routing update messages.

EGP restricts exterior gateways by allowing them to advertise only those destination networks reachable entirely within that gateway's autonomous system. Thus, an exterior gateway using **EGP** passes along information to its **EGP** neighbors but does not advertise access information about its **EGP** neighbors outside its autonomous system.

EGP does not interpret any of the distance metrics that appear in routing update messages from other protocols. **EGP** uses the distance field to specify whether a path exists (a value of 255 means that the network is unreachable). The value cannot be used to compute the shorter of two routes unless those routes are both contained within a single autonomous system. Therefore, **EGP** cannot be used as a routing algorithm. As a result, there will be only one path from the exterior gateway to any network.

In contrast to the **Routing Information Protocol** (**RIP**), which can be used within an autonomous system of Internet networks that dynamically reconfigure routes, **EGP** routes are predetermined in the **/etc/gated.conf** file. **EGP** assumes that **IP** is the underlying protocol.

EGP message types:

The various exterior gateway protocol (EGP) message types are defined here.

Item	Description
Neighbor Acquisition Request	Used by exterior gateways to request to become neighbors of each other.
Neighbor Acquisition Reply	Used by exterior gateways to accept the request to become neighbors.
Neighbor Acquisition Refusal	Used by exterior gateways to deny the request to become neighbors. The refusal message includes reasons for refusal, such as `out of table space`.
Neighbor Cease	Used by exterior gateways to cease the neighbor relationship. The cease message includes reasons for ceasing, such as `going down`.
Neighbor Cease Acknowledgment	Used by exterior gateways to acknowledge the request to cease the neighbor relationship.
Neighbor Hello	Used by exterior gateways to determine connectivity. A gateway issues a `Hello` message and another gateway issues an `I Heard You` message.
I Heard You	Used by exterior gateways to reply to a `Hello` message. The `I Heard You` message includes the access of the answering gateway and, if the gateway is unreachable, a reason for lack of access, such as `You are unreachable because of problems with my network interface`.
NR Poll	Used by exterior gateways to query neighbor gateways about their ability to reach other gateways.
Network Reachability	Used by exterior gateways to answer the `NR Poll` message. For each gateway in the message, the `Network Reachability` message contains information on the addresses that gateway can reach through its neighbors.
EGP Error	Used by exterior gateways to respond to EGP messages that contain bad checksums or have fields containing incorrect values.

TCP/IP implements the **EGP** protocol in the **gated** server command and does not provide an API to this protocol.

File Transfer Protocol:

File Transfer Protocol (**FTP**) allows hosts to transfer data among dissimilar hosts, as well as files between two foreign hosts indirectly.

FTP provides for such tasks as listing remote directories, changing the current remote directory, creating and removing remote directories, and transferring multiple files in a single request. **FTP** keeps the transport secure by passing user and account passwords to the foreign host. Although **FTP** is designed primarily to be used by applications, it also allows interactive user-oriented sessions.

FTP uses reliable stream delivery (**TCP/IP**) to send the files and uses a Telnet connection to transfer commands and replies. **FTP** also understands several basic file formats including NETASCII, IMAGE, and Local 8.

TCP/IP implements **FTP** in the **ftp** user command and the **ftpd** server command and does not provide an applications programming interface (API) to this protocol.

When creating anonymous ftp users and directories please be sure that the home directory for users ftp and anonymous (for example, `/u/ftp`) is owned by root and does not allow write permissions (for example, `dr-xr-xr-x`). The script `/usr/samples/tcpip/anon.ftp` can be used to create these accounts, files and directories.

Telnet Protocol:

The **Telnet Protocol** (**TELNET**) provides a standard method for terminal devices and terminal-oriented processes to interface.

TELNET is commonly used by terminal emulation programs that allow you to log into a remote host. However, **TELNET** can also be used for terminal-to-terminal communication and interprocess communication. **TELNET** is also used by other protocols (for example, **FTP**) for establishing a protocol control channel.

TCP/IP implements **TELNET** in the **tn**, **telnet**, or **tn3270** user commands. The **telnetd** daemon does not provide an API to **TELNET**.

TCP/IP supports the following **TELNET** options which are negotiated between the client and server:

Item	Description
BINARY TRANSMISSION (Used in **tn3270** sessions)	Transmits characters as binary data.
SUPPRESS GO_AHEAD (The operating system suppresses GO-AHEAD options.)	Indicates that when in effect on a connection between a sender of data and the receiver of the data, the sender need not transmit a GO_AHEAD option. If the GO_AHEAD option is not desired, the parties in the connection will probably suppress it in both directions. This action must take place in both directions independently.
TIMING MARK (Recognized, but has a negative response)	Makes sure that previously transmitted data has been completely processed.
EXTENDED OPTIONS LIST	Extends the **TELNET** option list for another 256 options. Without this option, the **TELNET** option allows only 256 options.
ECHO (User-changeable command)	Transmits echo data characters already received back to the original sender.
TERM TYPE	Enables the server to determine the type of terminal connected to a user **TELNET** program.
SAK (Secure Attention Key)	Establishes the environment necessary for secure communication between you and the system.
NAWS (Negotiate About Window Size)	Enables client and server to negotiate dynamically for the window size. This is used by applications that support changing the window size.

Note: TELNET must allow transmission of eight bit characters when not in binary mode in order to implement ISO 8859 Latin code page.

Trivial File Transfer Protocol:

The **Trivial File Transfer Protocol** (**TFTP**) can read and write files to and from a foreign host.

Because **TFTP** uses the unreliable **User Datagram Protocol** to transport files, it is generally quicker than **FTP**. Like **FTP**, **TFTP** can transfer files as either NETASCII characters or as 8-bit binary data. Unlike **FTP**, **TFTP** cannot be used to list or change directories at a foreign host and it has no provisions for security like password protection. Also, data can be written or retrieved only in public directories.

TCP/IP implements **TFTP** in the **tftp** and **utftp** user commands and in the **tftpd** server command. The **utftp** command is a form of the **tftp** command for use in a pipe. **TCP/IP** does not provide an API to this protocol.

For more information, see the **tftp** or **utftp** command description and the **tftpd** daemon description in *Commands Reference, Volume 5*.

Name/Finger Protocol:

The **Name/Finger Protocol** (**FINGER**) is an application-level Internet protocol that provides an interface between the **finger** command and the **fingerd** daemon.

The **fingerd** daemon returns information about the users currently logged in to a specified remote host. If you execute the finger command specifying a user at a particular host, you will obtain specific information about that user. The **FINGER** Protocol must be present at the remote host and at the requesting host. **FINGER** uses **Transmission Control Protocol** ("Transmission Control Protocol" on page 147) as its underlying protocol.

Note: **TCP/IP** does not provide an API to this protocol.

For more information, see the **finger** command description and the **fingerd** daemon description in *Commands Reference, Volume 2*.

Distributed Computer Network Local-Network Protocol:

The **gated** daemon provides the **Distributed Computer Network (DCN)** local network protocol.

Local-Network Protocol (**HELLO**) is an interior gateway protocol designed for use within autonomous systems. (For more information, see "Autonomous systems" on page 151.) **HELLO** maintains connectivity, routing, and time-keeping information. It allows each machine in the network to determine the shortest path to a destination based on time delay and then dynamically updates the routing information to that destination.

For more information, see the **gated** daemon description in *Commands Reference, Volume 2*.

Remote Command Execution Protocol:

The **rexec** user command and the **rexecd** daemon provide the remote command execution protocol, allowing users to run commands on a compatible remote host.

For more information, see the **rexec** command description and the **rexecd** daemon description in *Commands Reference, Volume 4*.

Remote Login Protocol:

The **rlogin** user command and the **rlogind** daemon provide the **remote login protocol**, allowing users to log in to a remote host and use their terminals as if they were directly connected to the remote host.

For more information, see the **rlogin** command description and the **rlogind** daemon description in *Commands Reference, Volume 4*.

Remote Shell Protocol:

The **rsh** user command and the **rshd** daemon provide the **remote command shell protocol**, allowing users to open a shell on a compatible foreign host for running commands.

For more information, see the **rsh** command description and the **rshd** daemon description in *Commands Reference, Volume 4*.

Wake On LAN Protocol:

Wake On LAN (WOL) allows you to wake up one or more hosts that are connected to a network in suspended mode by sending a Magic Packet to the specified address or addresses on the specified subnet.

For more information on using **WOL**, see the **wol** command description in the *Commands Reference, Volume 6*.

Routing Information Protocol:

Routing Information Protocol (**RIP**) and the **routed** and **gated** daemons that implement it keep track of routing information based on gateway hops and maintain kernel-routing table entries.

For more information, see the **routed** and **gated** daemons.

Time Server Protocol:

The **timed** daemon is used to synchronize one host with the time of other hosts.

It is based on the client/server concept. For more information, see the **timedc** command description and the **timed** daemon description in *Commands Reference, Volume 5*.

Assigned Numbers

For compatibility with the general network environment, well-known numbers are assigned for the Internet versions, networks, ports, protocols, and protocol options. Additionally, well-known names are also assigned to machines, networks, operating systems, protocols, services, and terminals.

TCP/IP complies with the assigned numbers and names defined in RFC 1010, *Assigned Numbers*.

The **Internet Protocol** (**IP**) defines a 4-bit field in the **IP** header that identifies the version of the general Internetwork protocol in use. For **IP**, this version number in decimal is 4. For details on the assigned numbers and names used by **TCP/IP**, see `/etc/protocols` and `/etc/services` files included with **TCP/IP**. For further details on the assigned numbers and names, refer to RFC 1010 and the `/etc/services` file.

TCP/IP local area network adapter cards

The network adapter card is the hardware that is physically attached to the network cabling. It is responsible for receiving and transmitting data at the physical level.

The network adapter card is controlled by the network adapter device driver.

A machine must have one network adapter card (or connection) for each network (not network type) to which it connects. For instance, if a host attaches to two token-ring networks, it must have two network adapter cards.

TCP/IP uses the following network adapter cards and connections:

- Standard Ethernet Version 2
- IEEE 802.3
- Token-ring
- Asynchronous adapters and native serial ports
- Fiber Distributed Data Interface (FDDI)
- Serial Optical Channel Converter (described in *Kernel Extensions and Device Support Programming Concepts*)
- Fibre Channel

The Ethernet and 802.3 network technologies use the same type of adapter.

Each machine provides a limited number of expansion slots, some or all of which you might wish to use for communications adapters. Additionally, each machine supports a limited number of communications adapters of a given type. Within these limits (software limitations), you can install any combination of adapters up to the total number of expansion slots available in your machine (hardware limitations).

Only one **Transmission Control Protocol/Internet Protocol (TCP/IP)** interface is configurable regardless of the number of Serial Optical Channel Converters supported by the system. The Serial Optical device driver makes use of both channel converters even though only one logical **TCP/IP** interface is configured.

Installing a network adapter

Use this procedure to install a network adapter.

To install a network adapter:

1. Shut down the computer. See the **shutdown** command for information on how to shut down a system.
2. Turn off the computer power.
3. Remove the computer cover.
4. Find a free slot and insert the network adapter. Be careful to seat the adapter properly in the slot.
5. Replace the computer cover.
6. Restart the computer.

Adapter management and configuration

To configure and manage token-ring or Ethernet adapters, use the tasks in the following table.

Table 59. Configuring and managing adapters tasks

Task	SMIT fast path	Command or file
Configure an Adapter	`smit chgtok` (token-ring) `smit chgenet` (Ethernet)	1. Determine adapter name:[1] `lsdev -C -c adapter -t tokenring -H` or `lsdev -C -c adapter -t ethernet -H` 2. Reset ring speed (token-ring) or connector type (Ethernet), if necessary. For example: `chdev -l tok0 -a ring_speed=16 -P` or `chdev -l ent0 -a bnc_select=dix -P`
Determining a Network Adapter Hardware Address	`smit chgtok` (token-ring) `smit chgenet` (Ethernet)	`lscfg -l tok0 -v` (token-ring)[2] `lscfg -l ent0 -v` (Ethernet)[2]

Setting an Alternate Hardware Address	`smit chgtok` (token-ring) `smit chgenet` (Ethernet)	1. Define the alternate hardware address. For example, for token-ring:[2,3]
		`chdev -l tok0 -a alt_addr=0X10005A4F1B7F`
		For Ethernet:[2,3]
		`chdev -l ent0 -a alt_addr=0X10005A4F1B7F -p`
		2. Begin using alternate address, for token-ring:[4]
		`chdev -l tok0 -a use_alt_addr=yes`
		For Ethernet:[4]
		`chdev -l ent0 -a use_alt_addr=yes`

Note:

1. The name of a network adapter can change if you move it from one slot to another or remove it from the system. If you ever move the adapter, issue the **diag -a** command to update the configuration database.
2. Substitute your adapter name for `tok0` and `ent0`.
3. Substitute your hardware address for `0X10005A4F1B7F`.
4. After performing this procedure, you might experience a disruption of communication with other hosts until they flush their Address Resolution Protocol (ARP) cache and obtain the new hardware address of this host.

Virtual Local Area Networks

VLANs (Virtual Local Area Networks) can be thought of as logical broadcast domains. A VLAN splits up groups of network users on a real physical network onto segments of logical networks.

This implementation supports the IEEE 802.1Q VLAN tagging standard with the capability to support multiple VLAN IDs running on Ethernet adapters. Each VLAN ID is associated with a separate Ethernet interface to the upper layers (IP, etc.) and creates unique logical Ethernet adapter instances per VLAN, for example `ent1`, `ent2` and so on.

The IEEE 802.1Q VLAN support can be configured over any supported Ethernet adapters. The adapters must be connected to a switch that supports IEEE 802.1Q VLAN.

You can configure multiple VLAN logical devices on a single system. Each VLAN logical devices constitutes an additional Ethernet adapter instance. These logical devices can be used to configure the same Ethernet IP interfaces as are used with physical Ethernet adapters. As such, the **no** option, *ifsize* (default 8), needs to be increased to include not only the Ethernet interfaces for each adapter, but also any VLAN logical devices that are configured. See the **no** command documentation.

Each VLAN can have a different maximum transmission unit (MTU) value even if sharing a single physical Ethernet adapter.

VLAN support is managed through SMIT. Type the `smit vlan` fast path from the command line and make your selection from the main VLAN menu. Online help is available.

After you configure VLAN, configure the IP interface, for example, `en1` for standard Ethernet or `et1` for IEEE 802.3, using SMIT, or commands.

AIX 5.3 and later supports virtual Ethernet using a virtual I/O switch as a method to perform in-memory communication between partitions in a POWER5 system. The switch also supports IEEE 802.1Q tagging, which allows the virtual Ethernet adapters to belong different VLANs on the switch. Virtual Ethernet adapters are created and configured on partitions using the Hardware Management Console (HMC). After it is created, the partition will see the virtual Ethernet adapter in the open firmware tree when it scans for devices. After it is detected, the virtual Ethernet adapter is configured and used just like a physical Ethernet adapter. For more information, see the hardware documentation for your POWER5 system.

Note:

1. If you try to configure a VLAN ID value that is already in use for the specified adapter, the configuration fails with the following error:

   ```
   Method error (/usr/lib/methods/chgvlan):
     0514-018 The values specified for the following attributes
              are not valid:
              vlan_tag_id    VLAN Tag ID
   ```

2. If a user (for example, IP interface) is currently using the VLAN logical device, any attempt to remove the VLAN logical device fails. A message similar to the following displays:

   ```
   Method error (/usr/lib/methods/ucfgcommo):
           0514-062 Cannot perform the requested function because the
                    specified device is busy.
   ```

   To remove the logical VLAN device, first detach the user. For example, if the user is IP interface `en1`, then you can use the following command:

   ```
   ifconfig en1 detach
   ```

   Then remove the network interface using the SMIT TCP/IP menus.

3. If a user (for example, IP interface) is currently using the VLAN logical device, any attempt to change the VLAN characteristic (VLAN tag ID or base adapter) fails. A message similar to the following displays:

   ```
   Method error (/usr/lib/methods/chgvlan):
           0514-062 Cannot perform the requested function because the
                    specified device is busy.
   ```

   To change the logical VLAN device, first detach the user. For example, if the user is the IP interface `en1`, you could use the following command:

   ```
   ifconfig en1 detach
   ```

   Then change the VLAN and add the network interface again using the SMIT TCP/IP menus.

VLAN troubleshooting:

tcpdump and **trace** can be used to troubleshoot the VLAN.

The trace hook ID for each type of transmit packet follows:

Item	Description
transmit packets	3FD
receive packets	3FE
other events	3FF

The **entstat** command gives the aggregate statistics of the physical adapter for which the VLAN is configured. It does *not* provide the individual statistics for that particular VLAN logical device.

VLAN restrictions:

Remote dump is not supported over a VLAN. Also, VLAN logical devices cannot be used to create a Cisco Systems' Etherchannel.

TCP/IP network interfaces

The **TCP/IP** Network Interface layer formats IP datagrams at the Network layer into packets that specific network technologies can understand and transmit.

A network interface is the network-specific software that communicates with the network-specific device driver and the IP layer in order to provide the IP layer with a consistent interface to all network adapters that might be present.

The IP layer selects the appropriate network interface based on the destination address of the packet to be transmitted. Each network interface has a network address. The Network Interface layer is responsible for adding or removing any link layer protocol header required to deliver a message to its destination. The **network adapter** device driver controls the network adapter card.

Although not required, a network interface is usually associated with a network adapter. For instance, the loopback interface has no network adapter associated with it. A machine must have one network adapter card for each network (not network type) to which it connects. However, a machine requires only one copy of the network interface software for each network adapter it uses. For instance, if a host attaches to two token-ring networks, it must have two network adapter cards. However, only one copy of the **token-ring** network interface software and one copy of the token-ring device driver is required.

TCP/IP supports types of network interfaces:

- Standard Ethernet Version 2 (en)
- IEEE 802.3 (et)
- Token-ring (tr)
- **Serial Line Internet Protocol (SLIP)**
- Loopback (lo)
- FDDI
- Serial Optical (so)
- **Point-to-Point Protocol (PPP)**
- Virtual IP Address (vi)

The Ethernet, 802.3, and token-ring interfaces are for use with local area networks (LANs). The **SLIP** interface is for use with serial connections. The loopback interface is used by a host to send messages back to itself. The Serial Optical interface is for use with optical point-to-point networks using the Serial Optical Link device handler. **Point to Point protocol** is most often used when connecting to another computer or network via a modem. The Virtual IP Address interface (also called *virtual interface*) is not associated with any particular network adapter. Multiple instances of a virtual interface can be configured on a host. When virtual interfaces are configured, the address of the first virtual interface becomes the

source address unless an application has chosen a different interface. Processes that use a virtual IP address as their source address can send packets through any network interface that provides the best route for that destination. Incoming packets destined for a virtual IP address are delivered to the process regardless of the interface through which they arrive.

Automatic configuration of network interfaces

When a new network adapter is physically installed in the system, the operating system automatically adds the appropriate network interface for that adapter.

For example, if you install a token-ring adapter in your system, the operating system assigns it the name `tok0` and add a token-ring network interface named `tr0`. If you install an Ethernet adapter in your system, the operating system assigns it the name `ent0` and add both an Ethernet Version 2 and an IEEE 802.3 interface, named `en0` and `et0` respectively.

In most cases, there is a one-to-one correspondence between adapter names and network interface names. For example, token-ring adapter `tok0` corresponds to interface `tr0`, adapter tok1 corresponds to interface `tr1`, and so on. Similarly, Ethernet adapter `ent0` corresponds to interface `en0` (for Ethernet Version 2) and `et0` (for IEEE 802.3), and adapter `ent1` corresponds to interface `en1` (for Ethernet Version 2) and `et1` (for IEEE 802.3).

Note: Under normal circumstances, you do not need to delete or add a network interface manually. However, some problem determination procedures might require you to do so. In this case, use the SMIT fast path, **smit inet**, to delete and re-add the appropriate interface.

TCP/IP default configuration values

At each system startup, the operating system automatically configures the network interface software based upon the information in the ODM database. Initially, the network interface is configured with default values.

In order to communicate through a given network interface, the Internet address must be set. This is the only attribute that you need to set. All other necessary attributes can use the default values. The default values for each network interface follow.

TCP/IP default Ethernet values:

Valid Ethernet network adapter attributes along can have their values changed using the Network Interface Selection menu in SMIT.

Attribute	Default value	Possible values
netaddr		
state	down	up, down, detach
arp	yes	yes, no
netmask		
broadcast		

The following valid Ethernet network device driver attribute is shown along with its default values, which can be changed using the Network Interface Drivers menu in SMIT.

Attribute	Default value	Possible values
mtu	1500	60 through 1500

TCP/IP default 802.3 values:

Valid 802.3 network adapter attributes can have their values changed using the Network Interface Selection menu in SMIT.

Attribute	Default value	Possible values
netaddr		
state	down	up, down, detach
arp	yes	yes, no
netmask		
broadcast		

The following valid 802.3 network device driver attribute is shown along with its default values, which can be changed using the Network Interface Drivers menu in SMIT.

Attribute	Default value	Possible values
mtu	1492	60 through 1492

TCP/IP default Token-Ring values:

Valid token-ring network adapter attributes can have their values changed using the Network Interface Selection menu in SMIT.

Attribute	Default value	Possible values
netaddr		
netmask		
state	down	up, down, detach
arp	yes	yes, no
hwloop	no	yes, no
netmask		
broadcast		
allcast	no	yes, no

The following valid token-ring network device driver attributes are shown along with its default values, which can be changed using the Network Interface Drivers menu in SMIT.

Attribute	Default value	Possible values
mtu (4Mbps)	1500	60 through 4056
mtu (16Mbps)	1500	60 through 17960

Note: When operating through a bridge, the default value of 1500 for the maximum transmission unit (MTU) should be changed to a value that is 8 less than the maximum information field (maximum I-frame) advertised by the bridge in the routing control field. For example, if the maximum I-frame value is 1500 in the routing control field, the MTU size should be set to 1492. This is for token-ring network interfaces only. For more information, see “TCP/IP problems with a Token-Ring/Token-Ring bridge” on page 401.

When using the IBM® 16/4 PowerPC token-ring adapter (ISA), the MTU is restricted to 2000.

TCP/IP default SLIP values:

Valid SLIP network adapter attributes can have their values changed using the Network Interface Selection menu in SMIT.

Attribute	Default value	Possible values
netaddr		
dest		
state	up	up, down, detach
netmask		

The following valid SLIP network device driver attribute is shown along with its default values as displayed under the Network Interface Drivers menu in SMIT.

Attribute	Default value	Possible values
mtu	1006	60 through 4096

TCP/IP default Serial Optical values:

Valid Serial Optical network channel converter can have their values changed using the Network Interface Selection menu in SMIT.

Attribute	Default value	Possible values
netaddr		
state	down	up, down, detach
netmask		

The following valid serial optical network device handler attribute is shown along with its default values as displayed under the Network Interface Drivers menu in SMIT.

Attribute	Default value	Possible values
mtu	61428	1 through 61428

Implications of multiple network interfaces on the same network

If multiple network interfaces are attached to a single network, each interface must have a unique IP address.

The Multipath Routing feature allows routes to be added to the IP routing table for multipath interfaces on the same subnet. This allows outgoing traffic to alternate between the interfaces rather than being sent through one interface only.

Network interface management

To manage network interfaces, use the WSM Network, FastPath (application) or the tasks in this table.

Table 60. Managing network interfaces tasks

Task	SMIT fast path	Command or file
List all network devices	`smit lsinet`	**lsdev -C -c if**
Configure a network device	`smit chinet`	See the **ifconfig** command and the `rc.net` file
Changing network interface info with remotely mounted `/usr`	`smit chdev`[1,2]	**chgif**[1,2]
Obtaining statistics for a network interface		**netstat -v**

Note:

1. Changes from a remotely mounted `/usr` affect only the Information Database (ODM) until the network is restarted or until the **ifconfig** command is used to make the changes take effect right away.
2. When using a remotely mounted `/usr`, be careful not to modify the interface being used, because that is the location of the libraries, commands, and kernel.

Interface-specific network options

TCP/IP interfaces must be specially tuned to achieve good, high-speed network performance (100 Mb or more). This effort is complicated by the fact that multiple network interfaces and a combination of traditional and high-speed **TCP/IP** interfaces can be used on a single system.

In the AIX operating system, Interface Specific Network Options (ISNO) allow system administrators to tune each **TCP/IP** interface individually for best performance.

There are five ISNO parameters for each supported interface: **rfc1323**, **tcp_nodelay**, **tcp_sendspace**, **tcp_recvspace**, and **tcp_mssdflt**. When set, the values for these parameters override the system-wide parameters of the same names that had been set with the **no** command. When ISNO options are not set for a particular interface, system-wide options are used. When options have been set by an application for a particular socket using the **setsockopt** subroutine, such options override the ISNOs.

The network option **use_isno**, set with the **no** command, must have a value of 1 for the ISNOs to take effect. The default value for **use_isno** is 1.

Some high-speed adapters have ISNO parameters set by default in the ODM database.

Gigabit Ethernet interfaces, when configured to use an MTU of 9000, use the following ISNO values by default:

Name	AIX 4.3.3 Value	AIX 4.3.3 (4330-08) Value	AIX 5.1 (and later) Value
tcp_sendspace	131072	262144	262144
tcp_recvspace	92160	131072	131072
rfc1323	1	1	1

Gigabit Ethernet interfaces, when configured to use an MTU of 1500, use the following ISNO values by default:

Name	AIX 4.3.3 Value	AIX 4.3.3 (4330-08) Value	AIX 5.1 (and later) Value
tcp_sendspace	65536	131072	131072
tcp_recvspace	16384	65536	65536
rfc1323	0	not set	not set

FDDI interfaces, when configured to use an MTU of 4352, use the following ISNO values by default:

Name	Value
tcp_sendspace	45046
tcp_recvspace	45046

The ISNO parameters cannot be displayed or changed using SMIT. They can be set using the **chdev** command or the **ifconfig** command. The **ifconfig** command changes the values only until the next reboot. The **chdev** command changes the values in the ODM database so they are used on subsequent reboots. The **lsattr** or **ifconfig** commands can be used to display the current values.

The following examples show commands that can be used first to verify system and interface support and then to set and verify the new values.

1. Verify general system and interface support using the **no** and **lsattr** commands.
   - Ensure the **use_isno** option is enabled using a command similar to the following:

     ```
     $ no -a | grep isno
               use_isno=1
     ```

   - Ensure the interface supports the five new ISNOs using the **lsattr -El** command, as shown in the following:

     ```
     $ lsattr -E -l en0 -H
               attribute   value  description
               rfc1323                     N/A
               tcp_nodelay                 N/A
               tcp_sendspace               N/A
               tcp_recvspace               N/A
               tcp_mssdflt                 N/A
     ```

2. Set the interface specific values, using either the **ifconfig** or **chdev** command. The **ifconfig** command sets values temporarily, which is recommended for testing. The **chdev** command alters the ODM, so customized values remain valid after reboot.
   - Set the **tcp_recvspace** and **tcp_sendspace** to 64K and enable **tcp_nodelay** by using one of the following:

     ```
     $ ifconfig en0 tcp_recvspace 65536 tcp_sendspace 65536 tcp_nodelay 1
     $ chdev -l en0 -a tcp_recvspace=65536 -a tcp_sendspace=65536 -a tcp_nodelay=1
     ```

   - Alternatively, assuming the **no** command reports an `rfc1323=1` global value, the root user can turn **rfc1323** off for all connections over en0 with the following commands:

     ```
     $ ifconfig en0 rfc1323 0
     $ chdev -l en0 -a rfc1323=0
     ```

3. Verify the settings using the **ifconfig** or **lsattr** command, as shown in the following example:

   ```
   $ ifconfig en0 <UP,BROADCAST,NOTRAILERS,RUNNING,SIMPLEX,MULTICAST,GROUPRT,64BIT>
             en0: flags=e080863
                 inet 9.19.161.100 netmask 0xffffff00 broadcast 9.19.161.255
                 tcp_sendspace 65536 tcp_recvspace 65536 tcp_nodelay 1 rfc1323 0
   $ lsattr -El en0
             rfc1323         0           N/A               True
             tcp_nodelay     1           N/A               True
             tcp_sendspace   65536       N/A               True
             tcp_recvspace   65536       N/A               True
             tcp_mssdflt                 N/A               True
   ```

TCP/IP addressing

TCP/IP includes an Internet addressing scheme that allows users and applications to identify a specific network or host with which to communicate.

An Internet address works like a postal address, allowing data to be routed to the chosen destination. **TCP/IP** provides standards for assigning addresses to networks, subnetworks, hosts, and sockets, and for using special addresses for broadcasts and local loopback.

Internet addresses are made up of a network address and a host (or local) address. This two-part address allows a sender to specify the network as well as a specific host on the network. A unique, official network address is assigned to each network when it connects to other Internet networks. However, if a local network is not going to connect to other Internet networks, it can be assigned any network address that is convenient for local use.

The Internet addressing scheme consists of Internet Protocol (IP) addresses and two special cases of IP addresses: broadcast addresses and loopback addresses.

Internet addresses

The Internet Protocol (IP) uses a 32-bit, two-part address field.

The 32 bits are divided into four *octets* as in the following:

```
01111101   00001101   01001001   00001111
```

These binary numbers translate into:

```
125              13              73              15
```

The two parts of an Internet address are the network address portion and the host address portion. This allows a remote host to specify both the remote network and the host on the remote network when sending information. By convention, a host number of 0 is used to refer to the network itself.

TCP/IP supports three classes of Internet addresses: Class A, Class B, and Class C. The different classes of Internet addresses are designated by how the 32 bits of the address are allocated. The particular address class a network is assigned depends on the size of the network.

Class A addresses:

A Class A address consists of an 8-bit network address and a 24-bit local or host address.

The first bit in the network address is dedicated to indicating the network class, leaving 7 bits for the actual network address. Because the highest number that 7 bits can represent in binary is 128, there are 128 possible Class A network addresses. Of the 128 possible network addresses, two are reserved for special cases: the network address 127 is reserved for local loopback addresses, and a network address of all ones indicates a broadcast address.

There are 126 possible Class A network addresses and 16,777,216 possible local host addresses. In a Class A address, the highest order bit is set to 0.

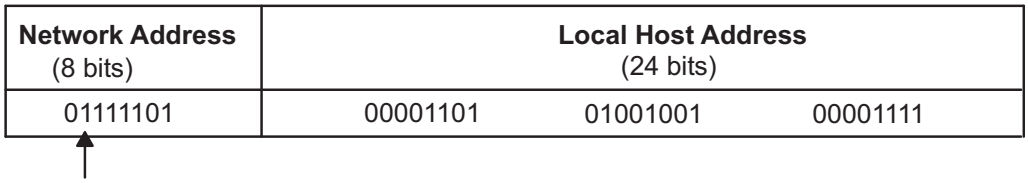


Figure 15. Class A address

This illustration shows a typical class A address structure. The first 8 bits contain the network address (always beginning with a zero). The remaining 24 bits contain the local host address.

The first octet of a Class A address is in the range 1 to 126.

Class B addresses:

A Class B address consists of a 16-bit network address and a 16-bit local or host address.

The first two bits in the network address are dedicated to indicating the network class, leaving 14 bits for the actual network address. There are 16,384 possible network addresses and 65,536 local host addresses. In a Class B address, the highest order bits are set to 1 and 0.

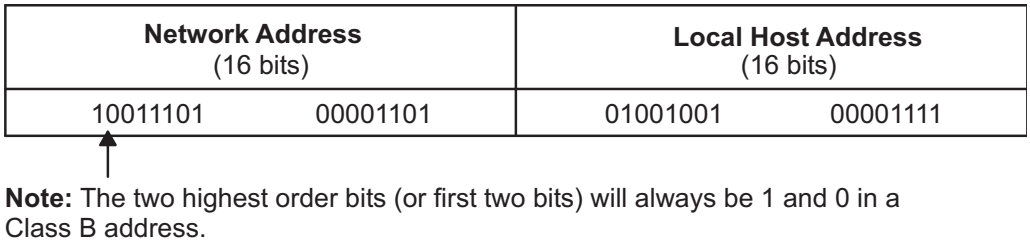


Figure 16. Class B address

This illustration shows a typical class B address structure. The first 16 bits contain the network address. The two highest order bits will always be a one and a zero. The remaining 16 bits contain the local host address.

The first octet of a Class B address is in the range 128 to 191.

Class C addresses:

A Class C address consists of a 24-bit network address and an 8-bit local host address.

The first three bits in the network address indicate the network class, leaving 21 bits for the actual network address. Therefore, there are 2,097,152 possible network addresses and 256 possible local host addresses. In a Class C address, the highest order bits are set to 1-1-0.

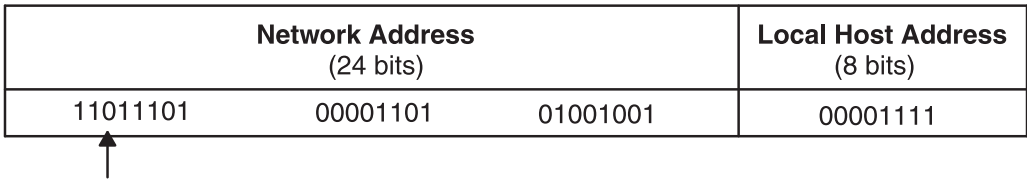


Figure 17. Class C address

This figure shows a typical class C address structure. The first 24 bits contain the network address (the three highest order bits will always be 1-1-0). The remaining 8 bits contain the local host address.

In other words, the first octet of a Class C address is in the range 192 to 223.

When deciding which network address class to use, you must consider how many local hosts there will be on the network and how many subnetworks will be in the organization. If the organization is small and the network will have fewer than 256 hosts, a Class C address is probably sufficient. If the organization is large, then a Class B or Class A address might be more appropriate.

Note: Class D (1-1-1-0 in the highest order bits) addresses provide for multicast addresses and are supported by UDP/IP under this operating system.

Machines read addresses in binary code. The conventional notation for Internet host addresses is the *dotted decimal*, which divides the 32-bit address into four 8-bit fields. The following binary value:

```
0001010   00000010   00000000   00110100
```

can be expressed as:

`010.002.000.052` or `10.2.0.52`

where the value of each field is specified as a decimal number and the fields are separated by periods.

Note: The **hostent** command does recognize the following addresses: .08, .008, .09, and .009. Addresses with leading zeros are interpreted as octal, and numerals in octal cannot contain 8s or 9s.

TCP/IP requires a unique Internet address for each network interface (adapter) on a network. These addresses are determined by entries in the configuration database, which must agree with entries in the `/etc/hosts` file or the **named** database if the network is using a name server.

Internet addresses using zeros:

When a C class Internet address contains a 0 as the host address portion, (for example, 192.9.200.0), TCP/IP sends a wildcard address on the network.

All machines with a Class C address of 192.9.200.*X* (where *X* represents a value between 0 and 254) should respond to the request. This results in a network flooded with requests to nonexistent machines.

Similarly, problems occur for Class B addresses such as 129.5.0.0. All machines with a Class B address of 129.5.*X*.*X*. (where *X* represents a value between 0 and 254) are obliged to respond to the request. In this case, because Class B addresses account for bigger networks than Class C addresses, the network is flooded with significantly more requests to nonexistent machines than for a Class C network.

Subnet addresses

Subnet addressing allows an autonomous system made up of multiple networks to share the same Internet address.

The subnetwork capability of TCP/IP also makes it possible to divide a single network into multiple logical networks (subnets). For example, an organization can have a single Internet network address that is known to users outside the organization, yet it can configure its network internally into departmental subnets. In either case, fewer Internet network addresses are required while local routing capabilities are enhanced.

A standard Internet Protocol address field has two parts: a network address and a local address. To make subnets possible, the local address part of an Internet address is divided into a subnet number and a host number. The subnet is identified so that the local autonomous system can route messages reliably.

In the basic Class A Internet address, which consists of an 8-bit network address and 24-bit local address, the local address identifies the specific host machine on the network.

Network Address (8 bits)	**Local Host Address** (24 bits)		
01111101	00001101	01001001	00001111

Figure 18. Class A address

This illustration shows a typical class A address structure. The first 8 bits contain the network address (always beginning with a zero). The remaining 24 bits contain the local host address.

To create a subnet address for this Class A Internet address, the local address can be divided into a number identifying the physical network (or subnet) and a number identifying the host on the subnet. Senders route messages to the advertised network address, and the local system takes responsibility for routing messages to its subnets and their hosts. When deciding how to partition the local address into subnet address and host address, you should consider the number of subnets and the number of hosts on those subnets.

In the following figure, the local address is partitioned into a 12-bit subnet address and a 12-bit host address.

Network Address (8 bits)	**Local Host Address** (24 bits)	
Network Address	**Subnet Address**	**Host Address**
01111101	00001101 0100	1001 00001111

Note: The high-order bit (or first bit) will always be 0 in a Class A address.

Figure 19. Class A address with corresponding subnet address

This illustration shows a typical class A address structure. The first 8 bits contain the network address (always beginning with a zero). The remaining 24 bits contain the local host address with the subnet address occupying the first 8 bits and the host address occupying the last 8 bits.

You have flexibility when assigning subnet addresses and host addresses. The bits of the local address can be divided according to the needs and potential growth of the organization and its network structure. The only restrictions are:

- `network_address` is the Internet address for the network.
- `subnet_address` is a field of a constant width for a given network.
- `host_address` is a field that is at least 1-bit wide.

If the width of the `subnet_address` field is 0, the network is not organized into subnets, and addressing to the network is performed using the Internet network address.

The bits that identify the subnet are specified by a bit mask and, therefore, are not required to be adjacent in the address. However, it is generally desirable for the subnet bits to be contiguous and located as the most significant bits of the local address.

Subnet masks:

When a host sends a message to a destination, the system must determine whether the destination is on the same network as the source or if the destination can be reached directly through one of the local interfaces. The system compares the destination address to the host address using the *subnet mask*.

If the destination is not local, the system sends the message on to a gateway. The gateway performs the same comparison to see if the destination address is on a network it can reach locally.

The subnet mask tells the system what the subnet partitioning scheme is. This bit mask consists of the network address portion and subnet address portion of the Internet address.

Network Address (8 bits)	**Local Host Address** (24 bits)	
Network Address	**Subnet Address**	**Host Address**
01111101	00001101 0100	1001 00001111

Class A Address with Corresponding Subnet Address

Network Address (8 bits)	**Local Host Address** (24 bits)	
Network Address	Subnet Address	Host Address
Subnet Mask		**Host Address**
01111101	00001101 0100	1001 00001111

Class A Address with Corresponding Subnet Mask

Figure 20. Class A address with corresponding subnet address

This illustration shows a typical class A address structure. The first 8 bits contain the network address (always beginning with a zero). The remaining 24 bits contain the local host address with the subnet address occupying the first 8 bits and the host address occupying the last 8 bits.

For example, the subnet mask of the Class A address with the partitioning scheme defined above is shown in this figure.

The subnet mask is a set of 4 bytes, just like the Internet address. The subnet mask consists of high bits (1's) corresponding to the bit positions of the network and subnetwork address, and low bits (0's) corresponding to the bit positions of the host address. A subnet mask for the previous address looks like the following figure.

Network Address (8 bits)	**Local Host Address** (24 bits)	
Network Address	**Subnet Address**	**Host Address**
11111111	11111111 1111	0000 00000000

Figure 21. Example subnet mask

This illustration shows a an example of a subnet mask structure. The first 8 bits contain the network address. The remaining 24 bits contain the local host address with the subnet address occupying the first 8 bits and the host address occupying the last 8 bits.

Address comparison:

The destination address and the local network address are compared by performing the logical AND and exclusive OR on the subnet mask of the source host.

The comparison process is outlined below:

1. Perform a logical AND of the destination address and the mask of the local subnet address.
2. Perform an exclusive OR on the result of the previous operation and the local net address of the local interface. If the result is all 0's, the destination is assumed to be reachable directly through one of the local interfaces.
3. If an autonomous system has more than one interface (therefore more than one Internet address), the comparison process is repeated for each local interface.

For example, assume that there are two local interfaces defined for a host network, T125. Their Internet addresses and the binary representations of those addresses are shown in the following example:

```
CLASS A   73.1.5.2    =  01001001 00000001 00000101 00000010


CLASS B   145.21.6.3  =  10010001 00010101 00000110 00000011
```

The corresponding subnet masks for the local network interfaces are shown in the following example:

```
CLASS A   73.1.5.2    =  11111111 11111111 11100000 00000000


CLASS B   145.21.6.3  =  11111111 11111111 11111111 11000000
```

If the source network, T125, is requested to send a message to a destination network with the host address 114.16.23.8 (represented in binary as: 01110010 00010000 00010111 00001000), the system checks whether the destination can be reached through a local interface.

Note: The **subnetmask** keyword must be set in the configuration database of each host that is to support subnets. Before the subnetwork capability can be used, all hosts on the network must support it. Set the subnet mask permanently in the configuration database using the Network Interface Selection menu in SMIT. The subnet mask can also be set in the running system using the **ifconfig** command. Using the **ifconfig** command to set the subnet mask is not a permanent change.

Broadcast addresses

The TCP/IP can send data to all hosts on a local network or to all hosts on all directly connected networks. Such transmissions are called *broadcast messages*.

For example, the **routed** routing daemon uses broadcast messages to query and respond to routing queries.

For data to be broadcast to all hosts on all directly connected networks, User Datagram Protocol (UDP) and Internet Protocol (IP) are used to send the data, and the host destination address in the IP header has all bits set to 1. For data to be broadcast to all hosts on a specific network, all the bits in the local address part of the IP address are set to 0. There are no user commands that use the broadcast capability, although such commands, or programs, can be developed.

The broadcast address can be changed temporarily by changing the *broadcast* parameter in the **ifconfig** command. Change the broadcast address permanently by using the SMIT fast path `smit chinet`. Changing the broadcast address may be useful if you need to be compatible with older versions of software that use a different broadcast address; for example, the host IDs are all set to 0.

Local loopback addresses

The Internet Protocol defines the special network address, 127.0.0.1, as a local loopback address.

Hosts use local loopback addresses to send messages to themselves. The local loopback address is set by the configuration manager during the system startup process. Local loopback is implemented in the kernel and can also be set with the **ifconfig** command. Loopback is invoked when the system is started.

TCP/IP name resolution

Although 32-bit Internet addresses provide machines an efficient means of identifying the source and destination of datagrams sent across an internetwork, users prefer meaningful, easily remembered names. **Transmission Control Protocol/Internet Protocol (TCP/IP)** provides a naming system that supports both flat and hierarchical network organizations.

Naming in flat networks is very simple. Host names consist of a single set of characters and generally are administered locally. In flat **TCP/IP** networks, each machine on the network has a file (`/etc/hosts`) containing the name-to-Internet-address mapping information for every host on the network. The administrative burden of keeping each machine naming file current grows as the **TCP/IP** network grows. When**TCP/IP** networks become very large, as on the Internet, naming is divided hierarchically. Typically, the divisions follow the network organization. In **TCP/IP**, hierarchical naming is known as the *domain name system* (DNS) and uses the DOMAIN protocol. The DOMAIN protocol is implemented by the **named** daemon in **TCP/IP**.

As in naming for flat networks, the domain name hierarchy provides for the assignment of symbolic names to networks and hosts that are meaningful and easy for users to remember. However, instead of each machine on the network keeping a file containing the name-to-address mapping for all other hosts on the network, one or more hosts are selected to function as *name servers*. Name servers translate (resolve) symbolic names assigned to networks and hosts into the efficient Internet addresses used by machines. A name server has complete information about some part of the domain, referred to as a *zone*, and it has *authority* for its zone.

Naming authority

In a flat network, all hosts in the network are administered by one central authority. This form of network requires that all hosts in the network have unique host names. In a large network, this requirement creates a large administrative burden on the central authority.

In a domain network, groups of hosts are administered separately within a tree-structured hierarchy of domains and subdomains. In this case, host names need to be unique only within the local domain, and

only the *root domain* is administered by a central authority. This structure allows subdomains to be administered locally and reduces the burden on the central authority. For example, the root domain of the Internet consists of such domains as `com` (commercial organizations), `edu` (educational organizations), `gov` (governmental organizations), and `mil` (military groups). New top-level domains can only be added by the central authority. Naming at the second level is delegated to designated agents within the respective domains. For example, in the following figure, `com` has naming authority for all commercial organization subdomains beneath it. Likewise, naming at the third level (and so on) is delegated to agents within that level. For example, in the Domain Structure of the Internet figure, Century has naming authority for its subdomains Austin, Hopkins, and Charlotte.

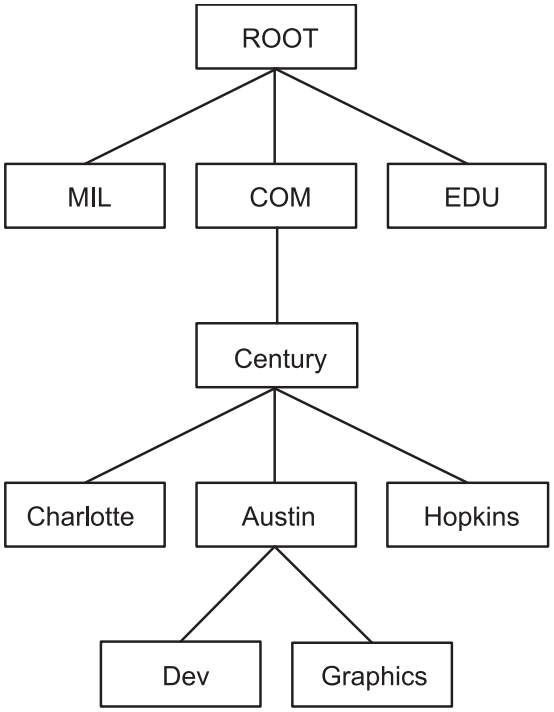


Figure 22. Domain structure of the Internet

This figure illustrates the hierarchical structure of the internet. It begins at the top with the root and branches to the next level containing the mil, com, and edu domains. Below the com domain is another level containing Charlotte, Austin, and Hopkins. Below Austin is Dev and Graphics.

Century's Austin subdomain might also be divided into zones, for example, Dev and Graphics. In this case, the zone `austin.century.com` has all the data contained in the domain `austin.century.com`, except that which was delegated to Dev and Graphics. The zone `dev.century.com` would contain only the data delegated to Dev; it would know nothing about Graphics, for example. The zone `austin.century.com` (as opposed to the domain of the same name) would contain only that data not delegated to other zones.

Naming conventions

In the hierarchical domain name system, names consist of a sequence of case-insensitive subnames separated by periods with no embedded blanks.

The DOMAIN protocol specifies that a local domain name must be fewer than 64 characters and that a host name must be fewer than 32 characters in length. The host name is given first, followed by a period (.), a series of local domain names separated by periods, and finally the root domain. A fully specified domain name for a host, including periods, must be fewer than 255 characters in length and in the following form:

```
host.subdomain1.[subdomain2 . . . subdomain].rootdomain
```

Because host names must be unique within a domain, you can use an abbreviated name when sending messages to a host within the same domain. For example, instead of sending a message to `smith.eng.lsu.edu`, a host in the `eng` domain could send a message to `smith`. Additionally, each host can have several aliases that other hosts can use when sending messages.

Naming hosts on your network

The purpose of using names for hosts is to provide a quick, easy, and unambiguous way to refer to the computers in your network. Internet system administrators have discovered that there are good, as well as poor, choices for host names. These suggestions are intended to help you avoid common pitfalls in choosing host names.

The following are some suggestions for choosing unambiguous, easy to remember host names:

- Terms that are rarely used, for example, `sphinx` or `eclipse`.
- Theme names, such as colors, elements (for example, `helium`, `argon`, or `zinc`), flowers, fish, and others.
- Real words (as opposed to random strings of characters).

The following are some examples of poor choices. In general, these are poor choices because they are difficult to remember or are confusing (either to humans or computers):

- Terms that are already in common use, for example, `up`, `down`, or `crash`.
- Names containing only numbers.
- Names that contain punctuation marks.
- Names that rely on case distinction, for example, `Orange` and `orange`.
- The name or initials of the primary user of the system.
- Names having more than 8 characters.
- Unusual or purposefully incorrect spellings, for example, `czek`, which could be confused with "check" or "czech."
- Names that are, or resemble, domain names, for example, `yale.edu`.

Name servers

In a flat name space, all names must be kept in the `/etc/hosts` file on each host on the network. If the network is very large, this can become a burden on the resources of each machine. In a hierarchical network, certain hosts designated as *name servers* resolve names into Internet addresses for other hosts.

This has two advantages over the flat name space. It keeps the resources of each host on the network from being tied up in resolving names, and it keeps the person who manages the system from having to maintain name resolution files on each machine on the network. The set of names managed by a single name server is known as its *zone of authority*.

Note: Although the host machine that performs the name resolution function for a zone of authority is commonly referred to as a *name server* host, the process controlling the function, the **named** daemon, is the actual name server process.

To further reduce unnecessary network activity, all name servers *cache* (store for a period of time) name-to-address mappings. When a client asks a server to resolve a name, the server checks its cache first to see if the name has been resolved recently. Because domain and host names do change, each item remains in the cache for a limited length of time specified by the TTL of the record. In this way, authorities can specify how long they expect the name resolution to be accurate.

Within any autonomous system there can be multiple name servers. Typically, name servers are organized hierarchically and correspond to the network organization. Referring to the "Domain Structure of the Internet" figure, each domain might have a name server responsible for all subdomains within the

domain. Each subdomain name server communicates with the name server of the domain above it (called the *parent* name server), as well as with the name servers of other subdomains.

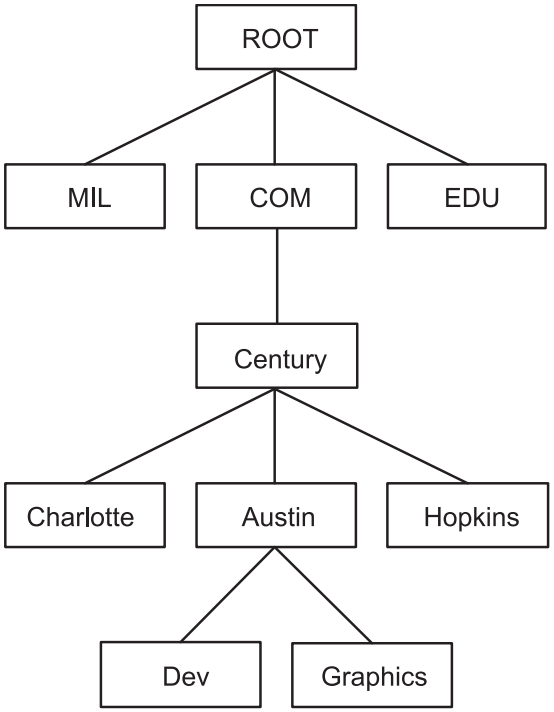


Figure 23. Domain structure of the Internet

This figure illustrates the hierarchical structure of the internet. It begins at the top with the root and branches to the next level containing the mil, com, and edu domains. Below the com domain is another level containing Charlotte, Austin, and Hopkins. Below Austin is Dev and Graphics.

For example, in the "Domain Structure of the Internet" figure, Austin, Hopkins, and Charlotte are all subdomains of the domain Century. If the tree hierarchy is followed in the network design, the Austin name server communicates with the name servers of Charlotte and Hopkins as well as with the parent Century name server. The Austin name server also communicates with the name servers responsible for its subdomains.

There are several types of name servers:

Item	Description
Master Name Server	Loads its data from a file or disk and can delegate authority to other servers in its domain.
Slave Name Server	Receives its information at system startup time for the given zone of authority from a master name server, and then periodically asks the master server to update its information. On expiration of the refresh value in the start of authority (SOA) Resource Record on a slave name server, or on receipt of a Notify message from the master name server, the slave reloads the database from the master if the serial number of the database on the master is greater than the serial number in the current database on the slave. If it becomes necessary to force a new zone transfer from the master, simply remove the existing slave databases and refresh the **named** daemon on the slave name server.
Stub Name Server	Although its method of database replication is similar to that of the slave name server, the stub name server only replicates the name server records of the master database rather than the whole database.

Item	Description
Hint Server	Indicates a name server that relies only on the hints that it has built from previous queries to other name servers. The hint name server responds to queries by asking other servers that have the authority to provide the information needed if a hint name server does not have a name-to-address mapping in its cache.
Forwarder or Client Server	Forwards queries it cannot satisfy locally to a fixed list of forwarding servers. Forwarding-only servers (a forwarder that obtains information and passes it on to other clients, but that is not actually a server) does not interact with the master name servers for the root domain and other domains. The queries to the forwarding servers are recursive. There can be one or more forwarding servers, which are tried in turn until the list is exhausted. A client and forwarder configuration is typically used when you do not want all the servers at a given site to interact with the rest of the Internet servers, or when you want to build a large cache on a select number of name servers.
Remote Server	Runs all the network programs that use the name server without the name server process running on the local host. All queries are serviced by a name server that is running on another machine on the network.

One name server host can perform in different capacities for different zones of authority. For example, a single name server host can be a master name server for one zone and a slave name server for another zone.

Name resolution

The process of obtaining an Internet address from a host name is known as name resolution and is done by the **gethostbyname** subroutine.

The process of translating an Internet address into a host name is known as reverse name resolution and is done by the **gethostbyaddr** subroutine. These routines are essentially accessors into a library of name translation routines known as *resolvers*.

Resolver routines on hosts running **TCP/IP** normally attempt to resolve names using the following sources:

1. BIND/DNS (named)
2. Network Information Service (NIS)
3. Local `/etc/hosts` file

To resolve a name in a domain network, the resolver routine first queries the domain name server database, which might be local if the host is a domain name server or on a foreign host. Name servers translate domain names into Internet addresses. The group of names for which a name server is responsible is its zone of authority. If the resolver routine is using a remote name server, the routine uses the domain name protocol (DOMAIN) to query for the mapping. To resolve a name in a flat network, the resolver routine checks for an entry in the local `/etc/hosts` file. When NIS is used, the `/etc/hosts` file on the master server is checked.

By default, resolver routines attempt to resolve names using the above resources. BIND/DNS is tried first. If the `/etc/resolv.conf` file does not exist or if BIND/DNS could not find the name, NIS is queried if it is running. NIS is authoritative over the local `/etc/hosts`, so the search ends here if it is running. If NIS is not running, then the local `/etc/hosts` file is searched. If none of these services can find the name, then the resolver routines return with HOST_NOT_FOUND. If all of the services are unavailable, then the resolver routines return with SERVICE_UNAVAILABLE.

The default order described above can be overwritten by creating the `/etc/irs.conf` configuration file and specifying the desired order. Also, both the default and `/etc/irs.conf` orderings can be overwritten with the environment variable, **NSORDER**. If either the `/etc/irs.conf` file or **NSORDER** environment variable are defined, then at least one value must be specified along with the option.

To specify host ordering with the /etc/irs.conf file:

```
hosts value [ continue ]
```

The order is specified with each method indicated on a line by itself. The *value* is one of the listed methods and the **continue** keyword indicates that another resolver method follows on the next line.

To specify host ordering with the **NSORDER** environment variable:

```
NSORDER=value,value,value
```

The order is specified on one line with values separated by commas. White spaces are permitted between the commas and the equal sign.

For example, if the local network is organized as a flat network, then only the /etc/hosts file is needed. Given this example, the /etc/irs.conf file contains the following line:

```
hosts local
```

Alternatively, the **NSORDER** environment variable can be set as:

```
NSORDER=local
```

If the local network is a domain network using a name server for name resolution and an /etc/hosts file for backup, then both services should be specified. Given this example, the /etc/irs.conf file contains the following lines:

```
hosts dns continue
hosts local
```

The **NSORDER** environment variable is set as:

```
NSORDER=bind,local
```

Note: The values listed must be in lowercase.

When following any defined or default resolver ordering, the search algorithm continues from one resolver to the next only if:

- The current service is not running, therefore, it is unavailable.
- The current service cannot find the name and is not authoritative.

If the /etc/resolv.conf file does not exist, then BIND/DNS is considered not set up or running, and therefore it is not available. If the **getdomainname** and **yp_bind** subroutines fail, then the NIS service is considered not set up or running, and therefore it is not available. If the /etc/hosts file could not be opened, then a local search is impossible, and therefore the file and service are unavailable.

When a service is listed as *authoritative*, it means that this service is the expert of its successors and has all pertinent names and addresses. Resolver routines do not try successor services, because successors might contain only a subset of the information in the authoritative service. Name resolution ends at service listed as authoritative, even if it does not find the name (in which case, the resolver routine returns HOST_NOT_FOUND). If an authoritative service is not available, then the next service specified is queried.

An authoritative source is specified with the string =auth directly behind a value. The entire word, authoritative can be typed in, but only the auth string is used. For example, if the **NSORDER** environment variable contains the following:

```
hosts = nis=auth,dns,local
```

The search ends after the NIS query (if NIS is running), regardless of whether the name was found. If NIS is not running, then the next source is queried, which is DNS.

TCP/IP name servers use caching to reduce the cost of searching for names of hosts on remote networks. Instead of searching for a host name each time a request is made, a name server first looks at its cache to see if the host name has been resolved recently. Because domain and host names do change, each item remains in the cache for a limited length of time specified by the time-to-live (TTL) value of the record. In this way, name servers can specify how long they expect their responses to be considered authoritative.

Potential host name conflict between name server and sendmail:

In a DNS environment, a host name that is set using the **hostname** command from the command line or in the `rc.net` file format must be the official name of the host as returned by the name server.

Generally, this name is the full domain name of the host in the form:

```
host.subdomain.subdomain.rootdomain
```

Note: Resolver routines require the default domain to be set. If the default domain is not set in the **hostname** command, then it must be set in the `/etc/resolv.conf` file.

If the host name is not set up as a fully qualified domain name, and if the system is set up to use a domain name server in conjunction with the **sendmail** program, the **sendmail** configuration file (`/etc/sendmail.cf`) must be edited to reflect this official host name. In addition, the domain name macros in this configuration file must be set for the **sendmail** program to operate correctly.

Note: The domain specified in the `/etc/sendmail.cf` file takes precedence over the domain set by the **hostname** command for all **sendmail** functions.

Potential domain name conflict between name server and sendmail:

Local domain names and domain name servers are specified in different files, depending on whether the host is a DOMAIN name server.

For a host that is in a DOMAIN network but is not a name server, the local domain name and domain name server are specified in the `/etc/resolv.conf` file. In a DOMAIN name server host, the local domain and other name servers are defined in files read by the **named** daemon when it starts.

Reverse Address Resolution Protocol

The **Reverse Address Resolution Protocol (RARP)** translates unique hardware addresses into Internet addresses on the Ethernet local area network (LAN) adapter (Ethernet protocol only).

Standard Ethernet protocol is supported with the following restrictions:

- The server only replies to **RARP** requests.
- The server only uses permanent **ARP** table entries.
- The server does not use dynamic **ARP** table entries.
- The server does not automatically reply for itself.

The system administrator must manually build and maintain a table of permanent **ARP** entries using the **arp** command. A specific **ARP** table entry must be added on the server for each host that requires **RARP** replies from an authoritative source.

Local name resolution (/etc/hosts) tasks

Configure the `/etc/hosts` file if your network is small, and you are using a flat naming scheme.

Even if you are using a hierarchical (or domain) naming scheme with name servers, you might want to configure the `/etc/hosts` file to identify hosts that are not known by the name servers.

Configure your system for local host resolution using the System Management Interface Tool (SMIT), or commands. If you choose the command method, be sure to preserve the format of the `/etc/hosts` file, as described in Hosts File Format for TCP/IP in the *Files Reference*.

Table 61. Local name resolution tasks

Task	SMIT fast path	Command or file
List All the Hosts	**smit lshostent**	Use the **hostent** command or **view** `/etc/hosts`
Add a Host	**smit mkhostent**	Use the **hostent** command or **edit** `/etc/hosts`
Change/Show Characteristics of a Host	**smit chhostent**	Use the **hostent** command or **edit** `/etc/hosts`
Remove a Host	**smit rmhostent**	Use the **hostent** command or **edit** `/etc/hosts`

Planning for DOMAIN name resolution

These suggestions can help you plan your own DOMAIN name resolution system.

If you are part of a larger internetwork, coordinate setting up your domain and name servers with the central authority.

- Because of the vast possibilities in architecture and configuration, become familiar with **TCP/IP**, DNS, and BIND before you solidify any plans. If you plan to use a network information service, become familiar with NFS and NIS as well. Books about these topics are widely available.
- Plan ahead.

  Changing a name is *much* more difficult than setting up the initial one. Obtain consensus from your organization on network, gateway, name server, and host names before you set up your files.
- Set up redundant name servers.

  If you cannot set up redundant name servers, be sure to set up slave and hint name servers so you have some type of backup.
- In selecting the name servers, keep the following in mind:
  - Choose machines that are physically closest to exterior systems.
  - The name servers should be as independent as possible. Try for different power supplies and independent cabling.
  - Find another network to back up your name resolution service, and do the same for other networks.
- Test the servers.
  - Test both regular and reverse name resolution.
  - Test zone transfer from master to slave name servers.
  - Test each name server after a system crash and reboot.
- Send name resolution requests to forwarder servers before they go to exterior name servers. This allows your name servers to share caches and improve performance by reducing the load on your master name servers.

```
objectclass container
        requires
                objectclass,
                cn
objectclass hosts
        requires
                objectclass,
                hname
        allows
                addr
                halias,
                comment
```

Name server resolution

In a hierarchical network, certain hosts are designated as *name servers*. These hosts resolve names into IP addresses for other hosts.

The **named** daemon controls the name server function and, therefore, must be run on a name server host.

Before you configure a name server, decide which type or types best fit the network it serves. There are several types of name servers.

A *master name server* actually stores the database that contains name-to-address mapping information. It loads its data from a file or disk and can delegate authority to other servers in its domain. A *slave name server* or *stub name server* receives its information at system startup time for a particular zone of authority from a master name server, and then periodically asks the master server to update its information. A *hint name server* responds to requests to resolve names by querying other servers that have the authority to provide the information needed.

Note: Previous generations of the **named** name server specified the master name server as the primary name server, the slave name server as the secondary name server, and the hint name server as the caching-only name server.

Keep in mind that a name server can function in different capacities for different zones of authority. For example, one name server host can be a master name server for one zone and a slave name server for another zone. If your system has NIS installed, these services can also provide name resolution.

There are several files that are involved in configuring name servers.

Item	Description
`conf`	This file is read when the **named** daemon starts. The records in the `conf` file tell the **named** daemon which type of server it is, which domains it has authority over (its zones of authority), and where to get the data for initially setting up its database. The default name of this file is `/etc/named.conf`. However, you can change the name of this file by specifying the name and path of the file on the command line when the **named** daemon is started. If you intend to use the `/etc/named.conf` as the `conf` file and it does not exist, a message is generated in `syslog` file and **named** terminates. However, if an alternative `conf` file is specified, and the alternative file does not exist, an error message is not generated, and **named** continues.
`cache`	Contains information about the local cache. The local cache file contains the names and addresses of the highest authority name servers in the network. The cache file uses the Standard Resource Record Format. The name of the cache file is set in the `conf` file.

`domain data`	There are three typical domain data files, also referred to as the **named** data files. The **named** *local* file contains the address resolution information for local loopback. The **named** *data* file contains the address resolution data for all machines in the name server zone of authority. The **named** *reverse data* file contains the reverse address resolution information for all machines in the name server zone of authority. The domain data files use the Standard Resource Record Format. Their file names are user definable and are set in the `conf` file. By convention, the names of these files generally include the name of the daemon (`named`), and the type of file and name of the domain is given in the extension. For example, the name server for the domain `abc` might have the following files:
	`named.abc.data`
	`named.abc.rev`
	`named.abc.local`
	
	When modifying the **named** data files the serial number in the SOA Resource Record must be incremented for slave name servers to properly realize the new zone changes.

`resolv.conf`	The presence of this file indicates to a host to go to a name server to resolve a name first. If the `resolv.conf` file does not exist, the host looks in the `/etc/hosts` file for name resolution. On a name server, the `resolv.conf` file must exist and can contain the local host address, the loopback address (127.0.0.1), or be empty. **Note:** The resolver routines require the default domain be set. If the default domain is not set in the `/etc/resolv.conf` file, then it must be set in the `hostname`

Time-to-live (TTL) is specified in resource records. If TTL is not specified in a record, the length of this time period defaults to the minimum field as defined in the start of authority (SOA) record for that zone. TTL is used when data is stored outside a zone (in a cache) to ensure that the data is not retained indefinitely.

Configuring domain name servers:

In this scenario, a master name server, slave name server, and hint name server will be configured to perform name resolution. Each name server will be a separate machine, and each will have an `/etc/named.conf` file configured, although the information in each will be different. The `/etc/named.conf` is read each time the **named** daemon is started, and it specifies what type of server it is (master, slave, or hint) and where it will get its name resolution data. Each of these name servers will be running BIND 8.

The master name server will be configured to provide name resolution for the `abc.aus.century.com` zone. In this scenario, the IP address of the master name server is `192.9.201.1`, and its host name is `venus.abc.aus.century.com`. It will provide name resolution for the venus, earth, mars, and jupiter host names. The `/etc/named.conf` file will be configured to specify that the **named** daemon should search the `/usr/local/domain` directory for its data files. The data files that will be configured for the master name server are `named.ca`, `named.abc.local`, `named.abc.data`, and `named.abc.rev`.

A slave name server will then be configured. The host name of the slave name server will be `earth.abc.aus.century.com`, and its IP address will be `192.9.201.5`. In the slave name server's `/etc/named.conf` file, we will specify the master name server's address so that the slave name server can replicate the master name server's `named.abc.data` and `named.abc.rev` files. In addition, the `named.ca` and `named.abc.local` data files will be configured for this server.

A hint name server will then be configured. The hint name server will store a local cache of host name and address mappings. If a requested address or host name is not in its cache, the hint server will contact the master name server, get the resolution information, and add it to its cache. In addition, the `named.ca` and `named.abc.local` data files will be configured for this server.

All information in the `named` data files (not the `/etc/named.conf` file) on the name servers must be in the Standard Resource Record Format. For explanations about the information about the `named` data files, see Standard Resource Record Format for TCP/IP in *Files Reference*.

The administrator for each of the name servers will be `gail.zeus.abc.aus.century.com`. This is specified in the local data files on each name server. In addition, in this scenario, the root name server is `relay.century.com` with IP address `129.114.1.2`.

At the end of this scenario, name resolution will be provided for the hosts venus, earth, mars, and jupiter. In addition, reverse name resolution (IP address-to-host name) will also be provided. When a request is received that cannot be resolved, the master name server will contact `relay.century.com` to find the information needed.

Things to Consider

- The information in this how-to scenario was tested using specific versions of AIX. The results you obtain might vary significantly depending on your version and level of AIX.

Step 1. Configure the Master Name Server

1. On the master name server, open the `/etc/named.conf` file. If there is no `/etc/named.conf` file in the `/etc` directory, create one by running the following command:

   ```
   touch /etc/named.conf
   ```

   Do the following to configure the `/etc/named.conf` file:

   a. Specify a directory clause in the options stanza. This enables the `named` data files to use paths relative to the `/usr/local/domain` directory. In this scenario, the following was added:

```
options {
    directory "/usr/local/domain";
};
```

If you choose not to specify a directory here, the `/etc` directory will be searched for the necessary data files.

b. To allow record data to be cached outside of the defined zones, specify the name of the hint zone file. In this scenario, the following was added:

```
zone "." IN {
    type hint;
    file "named.ca";
};
```

c. Add the following stanzas to specify each zone, the type of name server you are configuring, and your name server's domain data file. In this scenario, the master server for both forward and reverse zones is the following:

```
zone "abc.aus.century.com" in {
    type master;
    file "named.abc.data";
};
zone "201.9.192.in-addr.arpa" in {
    type master;
    file "named.abc.rev";
};
```

d. Define the name of the `named` local file. For example:

```
zone "0.0.127.in-addr.arpa" in {
    type master;
    file "named.abc.local";
};
```

After editing the file, save and close it.

2. Open the `/usr/local/domain/named.ca` file. Add the addresses of the root name servers for the domain. The following was added in this scenario:

```
; root name servers.
.          IN    NS    relay.century.com.
relay.century.com.   3600000    IN    A     129.114.1.2
```

After editing the file, save and close it.

3. Open the `/usr/local/domain/named.abc.local` file. Add the following information:

- The start of authority (SOA) of the zone and the default time-to-live information. The following was added in this scenario:

```
$TTL 3h    ;3 hour

@ IN SOA venus.abc.aus.century.com. gail.zeus.abc.aus.century.com.  (

                            1       ;serial
                            3600    ;refresh
                            600     ;retry
                            3600000 ;expire
                            3600    ;negative caching TTL
)
```

- The name server (NS) record. Insert a tab space at the beginning of the line; the **named** daemon will replace the tab space with the zone name:

```
<tab> IN    NS     venus.abc.aus.century.com.
```

- The pointer (PTR) record.

```
1      IN    PTR    localhost.
```

After editing the file, save and close it.

4. Open the /usr/local/domain/named.abc.data file. Add the following information:
   - The start of authority of the zone and the default time-to-live information for the zone. This record designates the start of a zone. Only one start of authority record per zone is allowed. In this scenario, the following was added:

     ```
     $TTL 3h    ;3 hour

     @ IN    SOA     venus.abc.aus.century.com. gail.zeus.abc.aus.century.com. (
                     1       ;serial
                     3600    ;refresh
                     600     ;retry
                     3600000 ;expire
                     3600    ;negative caching TTL
     )
     ```

   - The name server records for all master name servers in the zone. Insert a tab space at the beginning of the line; the **named** daemon will replace the tab space with the zone name:

     ```
     <tab> IN    NS       venus.abc.aus.century.com.
     ```

   - The name-to-address resolution information on all hosts in the name server zone of authority:

     ```
     venus        IN    A        192.9.201.1
     earth        IN    A        192.9.201.5
     mars         IN    A        192.9.201.3
     jupiter      IN    A        192.9.201.7
     ```

     Include other types of entries, such as canonical name records and mail exchanger records as needed.

   After editing the file, save and close it.
5. Open the /usr/local/domain/named.abc.rev file. Add the following information:
   - The start of authority of the zone and the default time-to-live information. This record designates the start of a zone. Only one start of authority record per zone is allowed:

     ```
     $TTL 3h    ;3 hour

     @  IN  SOA  venus.abc.aus.century.com. gail.zeus.abc.aus.century.com.  (
                              1       ;serial
                              3600    ;refresh
                              600     ;retry
                              3600000 ;expire
                              3600    ;negative caching TTL
     )
     ```

   - Other types of entries, such as name server records. If you are including these records, insert a tab space at the beginning of the line; the **named** daemon will replace the tab space with the zone name. In this scenario, the following was added:

     ```
     <tab> IN      NS      venus.abc.aus.century.com.
     ```

   - Address-to-name resolution information on all hosts to be in the name server's zone of authority.

     ```
     1                     IN    PTR    venus.abc.aus.century.com.
     5                     IN    PTR    earth.abc.aus.century.com.
     3                     IN    PTR    mars.abc.aus.century.com.
     7                     IN    PTR    jupiter.abc.aus.century.com.
     ```

     After editing the file, save and close it.
6. Create an /etc/resolv.conf file by running the following command:

   ```
   touch /etc/resolv.conf
   ```

   The presence of this file indicates that the host should use a name server for name resolution.
7. Add the following entry in the /etc/resolv.conf file:

   ```
   nameserver 127.0.0.1
   ```

   The 127.0.0.1 address is the loopback address, which causes the host to access itself as the name server. The /etc/resolv.conf file can also contain an entry similar to the following:

```
domain abc.aus.century.com
```

In this case, `abc.aus.century.com` is the domain name.

After editing the file, save and close it.

8. Use the `smit stnamed` SMIT fast path to enable the **named** daemon. This initializes the daemon with each system startup. Indicate whether you want to start the **named** daemon now, at the next system restart, or both.

Step 2. Configure the Slave Name Server

To configure a slave name server, use the following procedure. You will edit a series of files and then use SMIT to start the **named** daemon.

1. On the slave name server, open the `/etc/named.conf` file. If there is no `/etc/named.conf` file in the `/etc` directory, create on by running the following command:

   ```
   touch /etc/named.conf
   ```

   Do the following to configure the `/etc/named.conf` file:

   a. Specify a directory clause in the options stanza. This enables the `named` data files to use paths relative to the `/usr/local/domain` directory. In this scenario, the following was added:

      ```
      options {
          directory "/usr/local/domain";
      };
      ```

      If you choose not to specify a directory here, the **named** daemon will search the `/etc` directory for the necessary data files.

   b. To allow record data to be cached outside the defined zones, specify the name of the hint zone file for the name server:

      ```
      zone "." IN {
          type hint;
          file "named.ca";
      };
      ```

   c. Specify the slave zone clauses. Each stanza includes the zone type, a file name to which the name server can back up its data, and the IP address of the master name server, from which the slave name server will replicate its data files. In this scenario, we added the following slave zone clauses:

      ```
      zone "abc.aus.century.com" IN {
          type slave;
          file "named.abc.data.bak";
          masters { 192.9.201.1; };
      };
      zone "201.9.192.in-addr.arpa" IN {
          type slave;
          file "named.abc.rev.bak";
          masters { 192.9.201.1; };
      };
      ```

   d. To support resolving the loopback network address, specify a zone of type *master* with a source of `named.abc.local`, as well as the domain for which the name server is responsible.

      ```
      zone "0.0.127.in-addr.arpa" in {
          type master;
          file "named.abc.local";
      };
      ```

   After editing the file, save and close it.

2. Edit the `/usr/local/domain/named.ca` file.

   This file contains the address server that is the root domain server of the network. In this scenario, the following was added:

```
; root name servers.
.          IN    NS    relay.century.com.
relay.century.com.   3600000    IN    A     129.114.1.2
```

After editing the file, save and close it.

3. Open the /usr/local/domain/named.abc.local file. In this scenario, the following was added:
   - The start of authority (SOA) of the zone and the default time-to-live information:

     ```
     $TTL 3h    ;3 hour

     @ IN SOA earth.abc.aus.century.com. gail.zeus.abc.aus.century.com.  (

                                1       ;serial
                                3600    ;refresh
                                600     ;retry
                                3600000 ;expire
                                3600    ;negative caching TTL
     )
     ```

   - The name server (NS) record. Insert a tab space at the beginning of the line; the **named** daemon will replace the tab space with the zone name. For example:

     ```
     <tab> IN    NS     earth.abc.aus.century.com.
     ```

   - The pointer (PTR) record.

     ```
     1      IN    PTR    localhost.
     ```

   After editing the file, save and close it.
4. Create an /etc/resolv.conf file by running the following command:

   ```
   touch /etc/resolv.conf
   ```

5. Add the following entry to that file:

   ```
   nameserver 127.0.0.1
   domain abc.aus.century.com
   ```

   After editing the file, save and close it.
6. Use the smit stnamed SMIT fast path to enable the **named** daemon. This initializes the daemon with each system startup. Indicate whether you want to start the **named** daemon now, at the next system restart, or both.

Step 3. Configure the Hint Name Server

To configure a hint, or *cache-only*, name server, use the following procedure, which edits a series of files and then uses SMIT or the command line to start the **named** daemon.

1. On the hint name server, edit the /etc/named.conf file. If there is no /etc/named.conf file in the /etc directory, create one by running the following command:

   ```
   touch /etc/named.conf
   ```

   Do the following to configure the /etc/named.conf file:
   a. Specify a directory clause in the options stanza. This enables the named data files to use paths relative to the /usr/local/domain directory. In this scenario, the following was added:

      ```
      options {
          directory "/usr/local/domain";
      };
      ```

   b. To support resolving the loopback network address, specify a zone of type *master* with a source of named.abc.local, as well as the domain for which the name server is responsible. In this example, the options directory keyword was specified in the /etc/named.conf file.

      ```
      zone "0.0.127.in-addr.arpa" IN {
          type master;
         file "named.abc.local";
      };
      ```

c. Specify the name of the cache zone file. For example:

```
zone "." IN {
    type hint;
    file "named.ca";
};
```

After editing the file, save and close it.

2. Edit the /usr/local/domain/named.ca file.

   This file contains the addresses of the servers that are authoritative name servers for the root domain of the network. For example:

```
; root name servers.
.          IN    NS    relay.century.com.
relay.century.com.   3600000    IN    A     129.114.1.2
```

   After editing the file, save and close it.

3. Edit the /usr/local/domain/named.local file. In this scenario, the following information was added to this file:

   - The start of authority (SOA) of the zone and the default time-to-live information:

```
$TTL 3h    ;3 hour

@ IN SOA venus.abc.aus.century.com. gail.zeus.abc.aus.century.com.  (

                            1       ;serial
                            3600    ;refresh
                            600     ;retry
                            3600000 ;expire
                            3600    ;negative caching TTL
)
```

   - The name server (NS) record. Insert a tab space at the beginning of the line; the **named** daemon will replace the tab space with the zone name:

```
<tab> IN    NS     venus.abc.aus.century.com.
```

   - The pointer (PTR) record.

```
1      IN    PTR    localhost.
```

   After editing the file, save and close it.

4. Create an /etc/resolv.conf file by running the following command:

```
touch /etc/resolv.conf
```

5. Add the following entry to that file:

```
nameserver 127.0.0.1
domain abc.aus.century.com
```

   After editing the file, save and close it.

6. Use the smit stnamed SMIT fast path to enable the **named** daemon. This initializes the daemon with each system startup. Indicate whether you want to start the **named** daemon now, at the next system restart, or both.

When you reboot, your IPv6 configuration will be set. Repeat this process for each host.

Configuring a domain mail server:

Configuring a domain mail server provides users external to your organization a simple method for addressing mail to your users. That is, without a domain mail server, the mail address must specify a particular host in your organization.

For example sam@orange.widget.com, where widget.com is your organization's domain name, and orange is the host that sam uses. But with a domain mail server, users outside your organization can simply specify the user name and domain name, without having to know which host the user uses, for example, sam@widget.com.

To configure a domain mail server, use following procedure.

1. Create a mail exchanger (MX) record and an address (A) record for the mail server black.widget.com:

   ```
   widget.com           IN    MX     10 black.widget.com
   widget.com           IN     A     192.10.143.9
   black.widget.com     IN     A     192.10.143.9
   ```

2. Edit sendmail.cf on the mail server (black.widget.com) to add the domain alias (the **w** class):

   ```
   Cw $w $?D$w.$D$. widget.com
   ```

3. Mail clients must know where to send their non-local mail, so edit sendmail.cf on each client to point to the mail server (the **S** macro):

   ```
   DRblack.widget.com
   ```

4. Use the **NameServOpt** option to configure the **sendmail** daemon so everyone can use the MX records defined in the name server brown.widget.com.
5. Add aliases for users in the domain that do not have accounts on the mail server using the aliases file, for example:

   ```
   sam:sam@orange.widget.com
   david:david@green.widget.com
   judy:judy@red.widget.com
   ```

   Note: Mailbox (MB) records can serve the same function.
6. The serial number in the SOA Resource Record must be incremented because the database has been modified.
7. Refresh the name server database by issuing the refresh -s named command.
8. On the clients, run the refresh -s sendmail command to make the changes take effect.

There are other methods to configure a domain mail server. These procedures involve using mailbox (MB), mail rename (MR), and mail group (MG) records.

Configuring a domain mail server using mailbox records:

Use the following procedure to configure a domain mail server using mailbox records.

1. Define a mailbox (MB) record for each user in the domain. Add entries such as:

   ```
   sam IN MB orange.widget.com.
   ```

   to the /usr/local/domain/named.abc.data file on host brown.widget.com. These entries identify to the mail server black.widget.com where to send mail for each user in the domain.
2. Configure the **sendmail** daemon on the mail server black.widget.com to use the MB records defined in the name server brown.widget.com. Use the **NameServOpt** option.
3. Increment the serial number in the SOA Resource Record, because the database has been modified.
4. Refresh the name server database by running the refresh -s named command.
5. Type the refresh -s sendmail command to make the changes take effect.

Defining a mail rename record for a user:

Use the following procedure to define a mail rename record.

1. Edit the /usr/local/domain/named.abc.data file on your domain name server.
2. Add a Mail Rename (MR) record for each alias. For example, if a user sam has an alias sammy, the Mail Rename record is:

   ```
   sammy IN MR sam
   ```

This record causes all mail addressed to sammy to be delivered to sam. Each MR record should be entered on a line by itself.

3. The serial number in the SOA Resource Record must be incremented, because the database has been modified.
4. Refresh the name server database by typing the refresh -s named command.
5. Type the refresh -s sendmail command to make the changes take effect.

Defining mail group member records:

Use the following procedure to define mail group member records.

1. Edit the /usr/local/domain/named.abc.data file on your domain name server.
2. Add MG records for each mail group (MG). MG records function like the /etc/aliases file, with the aliases maintained on the name server. For example:

   ```
   users IN HINFO users-request widget.com
   users IN MG sam
   users IN MG david
   users IN MG judy
   ```

   This example causes all mail addressed to users@widget.com to be delivered to sam, david, and judy. Enter each MG record on a line by itself.

   Note: Users sam, david, and judy must have MB records defined.
3. The serial number in the SOA Resource Record must be incremented, because the database has been modified.
4. Refresh the name server database by typing the refresh -s named command.
5. Type the refresh -s sendmail command to make the changes take effect.

Defining mail exchanger records:

Use the following procedure to define mail exchanger records.

1. Edit the /usr/local/domain/named.abc.data file on your domain name server.
2. Add mail exchanger (MX) records for each machine not directly connected to your network to which you wish to forward mail. For example, if mail addressed to users on purple.widget.com should be forwarded to post.office.widget, the MX record looks similar to the following:

   ```
   purple.widget.com IN MX 0 post.office.widget.
   ```

   You must specify both host and machine names when using MX records. Enter each MG record on a line by itself. You can use wildcards, for example:

   ```
   *.widget.com IN MX 0 post.office.widget.
   ```

   This example causes mail to an unknown host (a host without an explicit MX record) in the widget.com domain to be forwarded to post.office.widget.

   Note: Wildcard MX records are not appropriate for use on the Internet.
3. The serial number in the SOA Resource Record must be incremented because the database has been modified.
4. Refresh the name server database by typing the refresh -s named command.
5. Type the refresh -s sendmail command to make the changes take effect.

Configuring a forwarder

To configure a forwarder server, use this procedure, which edits a series of files and then uses SMIT or the command line to start the **named** daemon.

1. Edit the /etc/named.conf file. If there is no named.conf file in the /etc directory, copy the /usr/samples/tcpip/named.conf sample file into the /etc directory and edit it. See the "named.conf File Format for TCP/IP" in the *Files Reference* for more information and a detailed example of a conf file.
   - Specify a forwarders line in the options stanza of the /etc/named.conf file that lists the IP addresses of the name servers that should receive the forwarded requests. For example:

     ```
     options {
         ...
         directory "/usr/local/domain";
         forwarders { 192.100.61.1; 129.35.128.222; };
         ...
     };
     ```
   - Specify the loopback zone. For example:

     ```
     zone "0.0.127.in-addr.arpa" in {
         type master;
         file "named.abc.local";
     };
     ```
   - Specify the hint zone. For example:

     ```
     zone "." IN {
         type hint;
         file "named.ca";
     };
     ```
2. Edit the /usr/local/domain/named.ca file. See the "DOMAIN Cache File Format for TCP/IP" in the *Files Reference* for more information and a detailed example of a cache file.

   This file contains the addresses of the servers that are authoritative name servers for the root domain of the network. For example:

   ```
   ; root name servers.
   .          IN    NS    relay.century.com.
   relay.century.com.   3600000    IN    A     129.114.1.2
   ```

   Note: All lines in this file must be in Standard Resource Record Format.
3. Edit the /usr/local/domain/named.abc.local file. See the DOMAIN Local Data File Format for TCP/IP in the *Files Reference* for more information and a detailed example of a local data file.
   a. Specify the start of authority (SOA) of the zone and the default time-to-live information. For example:

      ```
      $TTL 3h    ;3 hour

      @ IN SOA venus.abc.aus.century.com. gail.zeus.abc.aus.century.com.  (

                                  1       ;serial
                                  3600    ;refresh
                                  600     ;retry
                                  3600000 ;expire
                                  86400   ;negative caching TTL
      )
      ```
   b. Specify the name server (NS) record. For example:

      ```
      <tab> IN    NS     venus.abc.aus.century.com.
      ```
   c. Specify the pointer (PTR) record.

      ```
      1      IN    PTR    localhost.
      ```

   Note: All lines in this file must be in Standard Resource Record Format.
4. Create an /etc/resolv.conf file by typing the following command:

   ```
   touch /etc/resolv.conf
   ```

   The presence of this file indicates that the host should use a name server, not the /etc/hosts file, for name resolution.

Alternatively, the /etc/resolv.conf file might contain the following entry:

```
nameserver 127.0.0.1
```

The 127.0.0.1 address is the loopback address, which causes the host to access itself as the name server. The /etc/resolv.conf file may also contain an entry like the following:

```
domain domainname
```

In the previous example, the *domainname* value is austin.century.com.

5. Perform one of the following steps:
   - Enable the **named** daemon using the smit stnamed SMIT fast path. This initializes the daemon with each system startup. Indicate whether you want to start the **named** daemon now, at the next system restart, or both.
   - Edit the /etc/rc.tcpip file. Uncomment the line for the **named** daemon by removing the comment (#) symbol from the following line:

     ```
     #start /etc/named "$src_running"
     ```

     This initializes the daemon with each system startup.
6. If you chose not to initialize the named daemon through SMIT, start the daemon for this session by typing the following command:

   ```
   startsrc -s named
   ```

Configuring a forward only name server

To configure a forward only name server, use this procedure, which edits a series of files and then uses SMIT or the command line to start the **named** daemon.

Note: You can achieve a similar configuration without running a forward only name server. Instead, create an /etc/resolv.conf file that contains name server lines that point to the forwarders you wish to use.

1. Edit the /etc/named.conf file. If there is no named.conf file in the /etc directory, copy the /usr/samples/tcpip/named.conf sample file into the /etc directory and edit it. See the named.conf File Format for TCP/IP in the *Files Reference* for more information and a detailed example of a conf file.
   - Specify the forwarders and forward only lines in the options stanza of the /etc/named.conf file listing the IP addresses of the name servers receiving the forwarded requests. For example:

     ```
     options {
         ...
         directory "/usr/local/domain";
         forwarders { 192.100.61.1;  129.35.128.222; };
         forward only;
         ...
     };
     ```
   - Specify the loopback zone. For example:

     ```
     zone "0.0.127.in-addr.arpa" in {
         type master;
         file "named.abc.local";
     };
     ```
   - Specify the hint zone. For example:

     ```
     zone "." IN {
         type hint;
         file "named.ca";
     };
     ```
2. Edit the /usr/local/domain/named.ca file. For example: See the DOMAIN Cache File Format for TCP/IP in *Files Reference* for more information and a detailed example of a cache file. This file contains the addresses of the servers that are authoritative name servers for the root domain of the network.

```
; root name servers.
.          IN    NS    relay.century.com.
relay.century.com.   3600000    IN    A     129.114.1.2
```

Note: All lines in this file must be in Standard Resource Record Format.

3. Edit the `/usr/local/domain/named.abc.local` file. See the DOMAIN Local Data File Format for TCP/IP in the *Files Reference* for more information and a detailed example of a local data file.
   a. Specify the start of authority (SOA) of the zone and the default time-to-live information. For example:

```
$TTL 3h    ;3 hour

@ IN SOA venus.abc.aus.century.com. gail.zeus.abc.aus.century.com.  (

                            1       ;serial
                            3600    ;refresh
                            600     ;retry
                            3600000 ;expire
                            86400   ;negative caching TTL
)
```

   b. Specify the name server (NS) record. For example:

```
<tab> IN    NS     venus.abc.aus.century.com.
```

   c. Specify the pointer (PTR) record.

```
1      IN    PTR    localhost.
```

   Note: All lines in this file must be in Standard Resource Record Format.

4. Create an `/etc/resolv.conf` file by typing the following command:

```
touch /etc/resolv.conf
```

   The presence of this file indicates that the host should use a name server, not the `/etc/hosts` file, for name resolution.

   Alternatively, the `/etc/resolv.conf` file might contain the following entry:

```
nameserver 127.0.0.1
```

   The `127.0.0.1` address is the loopback address, which causes the host to access itself as the name server. The `/etc/resolv.conf` file can also contain an entry such as:

```
domain domainname
```

   In the previous example, the *domainname* value is `austin.century.com`.

5. Perform one of the following steps:
   - Enable the **named** daemon using the `smit stnamed` SMIT fast path. This initializes the daemon with each system startup. Indicate whether you want to start the **named** daemon now, at the next system restart, or both.
   - Edit the `/etc/rc.tcpip` file. Uncomment the line for the **named** daemon by removing the comment (#) symbol from the following line:

```
#start /etc/named "$src_running"
```

     This initializes the daemon with each system startup.

6. If you chose not to initialize the **named** daemon through SMIT, start the daemon for this session by typing the following command:

```
startsrc -s named
```

Configuring a host to use a name server

To configure a host to use a name server, use this procedure.

1. Create an `/etc/resolv.conf` file by running the following command:

```
touch /etc/resolv.conf
```

2. On the first line of the `/etc/resolv.conf` file, type the word `domain` followed by the full name of the domain that this host is in. For example:

```
domain abc.aus.century.com
```

3. On any blank line below the `domain` line, type the word `nameserver`, followed by at least one space, followed by the dotted decimal Internet address of the name server that this host is to use (the name server must serve the domain indicated by the `domain` statement). You can have up to 3 name server entries. For example, your `/etc/resolv.conf` file might contain the entries:

   ```
   nameserver 192.9.201.1
   nameserver 192.9.201.2
   ```

   The system queries the name servers in the order listed.

   ```
   search domainname_list
   ```

   Alternatively, the search keyword could be used to specify the order in which the resolver will query the domain list. In this case, `domainname_list` values are `abc.aus.century.com` and `aus.century.com`. The `domainname_list` can have a maximum of 1024 character strings, each separated by a space.

4. Assuming the name server is operational, you can test the communication between the host and the name server by typing the following command:

   ```
   host hostname
   ```

   Use the name of a host that should be resolved by the name server to see if the process is working. The output you receive should appear similar to the following:

   ```
   brown.abc.aus.century.com is 129.35.145.95
   ```

Other configuration tasks are shown in the following table.

Table 62. Configuring a host to use name server tasks

Task	SMIT fast path	Command or file
Create an `/etc/resolv.conf` File	`smit stnamerslv2`	create and edit `/etc/resolv.conf`[1]
List All the Name Servers Used by a Host	`smit lsnamerslv`	view `/etc/resolv.conf`
Add a Name Server	`smit mknamerslv`	edit `/etc/resolv.conf`[2]
Remove a Name Server	`smit rmnamerslv`	edit `/etc/resolv.conf`
Start/Restart Using Domain Name Resolution	`smit stnamerslv`	
Stop Using Domain Name Resolution	`smit spnamerslv`	
Change/Show the Domain	`smit mkdomain`	edit `/etc/resolv.conf`
Remove the Domain	`smit rmdomain`	edit `/etc/resolv.conf`

Related information

netsvc.conf File

Dynamic zones on the DNS name server

The **named** command allows for dynamic updates. The named database and configuration files need to be configured to allow for client machines to issue updates. A zone can be set to dynamic or static. The default zone is static.

To make a zone dynamic, you must add the keyword **allow-update** to that zone's stanza in the `/etc/named.conf` file. The **allow-update** keyword specifies an Internet address match list that defines hosts allowed to submit updates. See the named.conf File Format for TCP/IP in the *Files Reference* for more information and a detailed example of a `conf` file. In the following example, all hosts are allowed to update the dynamic zone:

```
zone "abc.aus.century.com" IN {
    type master;
    file "named.abc.data";
    allow-update { any; };
};
```

After a zone is marked dynamic, three modes of security can be initiated:

Item	Description
Unsecured	Allows anyone at anytime to update any information in the zone. **Attention:** Use of this mode is not recommended. It can lead to data loss, data interception, and user frustration. At the least, an unsecured zone should be limited to updates only from specific Internet addresses.
Controlled	Allows for the creation of new information and the replacement of existing information. This is probably the easiest mode to use for a secure transition environment. This mode also requires that all incoming updates be timestamped and have keyed signatures.
Presecured	Requires all updates to existing information be replaced with similar information. Does not allow for the creation of new information. This mode also requires that all incoming updates be timestamped and have keyed signatures.

A dynamic zone defaults to unsecured mode. To use one of the other modes, type `controlled` or `presecured` after the keyword **update-security** in the zone stanza of the `/etc/named.conf` file. This tells the **named** server the level of security to use with that zone. For example:

```
zone "abc.aus.century.com" IN {
    type master;
    file "named.abc.data";
    allow-update { any; };
    update-security controlled;
};
```

After a mode is selected, the actual data files must be modified for your level of security. In unsecured mode, the data files are used "as is." For controlled or presecured mode, you must generate a set of master server/host name key pairs for each name in the zone. This is done with the **nsupdate** command using the **-g** option. This command generates the key pair (a private and a public key). These keys are needed to authentically sign for updates. After generating all the keys for your list of zone names, you need to add them to the data file. The KEY format is as follows:

```
Index    ttl    Class    Type    KeyFlags    Protocol    Algorithm    KeyData
```

where:

Item	Description
Index	Specifies the name used to reference the data in the zone.
ttl	Specifies the time-to-live (TTL) for this data. This is an optional field.
Class	Specifies the class of the data. This is dependent on the zone, but usually it is IN.
Type	Indicates the type of the record. In this case, it is KEY.
KeyFlags	Gives named information about the key. 0x0000 defines the typical key record used for a host. 0x0100 defines the key record associated with the zone name.
Protocol	Specifies the protocol to use. Currently, there is only one, `0`.
Algorithm	Specifies the algorithm of the key. Currently, there is only one, 1. This is the MD5 Private/Public authentication method.
KeyData	Indicates the key in base64 representation. The **nsupdate** command generates both the public and private keys in base64 representation. The public key is listed last in the output file.

For example, to ensure security over a host name in a dynamic zone, a line similar to the following needs to be added to the zone file for the zone containing the host name.

```
bears    4660    IN    KEY    0x0000    0    1    AQOtg......
```

The preceding example indicates that `bears` has a KEY record defined. Someone wanting to update `bears` would have to sign his update with the private key matching the public key in the database. For the **nsupdate** command to succeed, the private key needs to be placed on the client in a keyfile (defaults to `/etc/keyfile`). It should follow the format:

```
hostname          mastername          base64          key
```

A similar KEY entry is required in the zone definition section. *A zone key is required for both presecured and controlled modes or the mode is considered to be unsecured.* This can be done as shown in the previous `bears` example, but the private key is left for the administrator to use with the **nsupdate** command's administrative mode.

1. To generate a key pair using the **nsupdate** command, type the following:

   ```
   nsupdate -g -h ZoneName -p ServerName -k AdminKeyFile
   ```

   This generates a key for the zone. In this example, **nsupdate** is linked to **nsupdate4**, which can be done by typing the following:

   ```
   ln -fs /usr/sbin/nsupdate4 /usr/sbin/nsupdate
   ```

2. Place the last key of the pair in the beginning section for the zone as follows:

   ```
   IN     KEY     0x0100  0  1  Key
   ```

   The entry for the `named.abc.data` file is as follows:

   ```
   $TTL 3h    ;3 hour

   @ IN    SOA     venus.abc.aus.century.com. gail.zeus.abc.aus.century.com. (
                   1       ;serial
                   3600    ;refresh
                   600     ;retry
                   3600000 ;expire
                   86400   ;negative caching TTL
   )
           IN      NS      venus.abc.aus.century.com.
           IN      KEY     0x0100  0 1 AQPlwHmIQeZzRk6Q/nQYhs3xwnhfTgF/
                           8YlBVzKSoKxVKPNLINnYW0mB7attTcfhHaZZcZr4u/
                           vDNikKnhnZwgn/
   venus   IN      A       192.9.201.1
   earth   IN      A       192.9.201.5
   mars    IN      A       192.9.201.3
   ```

3. The zone is now ready to be loaded by refreshing the name server. Place the AdminKeyFile on the client or DHCP server that is updating the zone. The zone key contained in the AdminKeyFile can be used to apply updates and maintenance operations to the name server.

BIND 9 security

BIND 9 offers Transaction Signatures (TSIG) and Signatures (SIG) as security measures for **named**.

The name server with BIND 9, by default, does not allow dynamic updates to authoritative zones, similarly to that of BIND 8.

BIND 9 primarily supports Transaction Signatures (TSIG) for server-to-server communication. This includes zone transfer, notify, and recursive query messages. TSIG is also useful for dynamic updates. A primary server for a dynamic zone should use access control to control updates, but IP-based access control is insufficient.

By using key base encryption rather than the current method of access control lists, TSIG can be used to restrict who can update to the dynamic zones. Unlike the Access Control List (ACL) method of dynamic updates, the TSIG key can be distributed to other updaters without having to modify the configuration files on the name server, which means there is no need for the name server to reread the configuration files.

It is important to note that BIND 9 does not have all the keywords implemented in BIND 8. In this example, we use the simple master configuration from BIND 8.

Note: To use named 9, you must relink the symbolic link to the **named** daemon to **named9**, and **nsupdate** to **nsupdate9** by running the following commands:

1. `ln -fs /usr/sbin/named9 /usr/sbin/named`
2. `ln -fs /usr/sbin/nsupdate9 /usr/sbin/nsupdate`

1. Generate the key using the **dnssec-keygen** command:

   ```
   dnssec-keygen -a HMAC-MD5 -b 128 -n HOST keyname
   ```

   - HMAC-MD5 is the algorithm used for encryption
   - 128 is the length of the key to use (or number of bits)
   - HOST: HOST is the TSIG keyword used to generate a host key for shared key encryption.

   The command

   ```
   dnssec-keygen -a HMAC-MD5 -b 128 -n HOST venus-batman.abc.aus.century.com
   ```

   would produce two key files, as follows:

   ```
   Kvenus-batman.abc.aus.century.com.+157+35215.key
   Kvenus-batman.abc.aus.century.com.+157+35215.private
   ```

   - 157 is the algorithm used (HMAC-MD5)
   - 35215 is the finger print, which is useful in DNNSEC because multiple keys per zone are allowed

2. Add the entry to named.conf on the master name server:

   ```
   // TSIG Key
   key venus-batman.abc.aus.century.com. {
           algorithm hmac-md5;
           secret "+UWSvbpxHWFdNwEAdy1Ktw==";
   };
   ```

   Assuming HMAC-MD5 is being used, both keyfiles contain the shared key, which are stored as the last entry in the files. Find a secure way to copy the shared secret key to the client. You do not need to copy the keyfile, just the shared secret key.

   Following is the entry for file Kvenus-batman.abc.aus.century.com.+157+35215.private:

   ```
   Private-key-format: v1.2
   Algorithm: 157 (HMAC_MD5)
   Key: +UWSvbpxHWFdNwEAdy1Ktw==
   ```

   Below is an example of the named.conf file for the master name server. The zone abc.aus.century.com allows zone transfer and dynamic updates only to servers with the key venus-batman.abc.aus.century.com. Do the same to the reverse zone, which requires updaters to have the shared key.

   ```
   // TSIG Key
   key venus-batman.abc.aus.century.com. {
           algorithm hmac-md5;
           secret "+UWSvbpxHWFdNwEAdy1Ktw==";
   };

   options {
           directory "/usr/local/domain";
   };

   zone "abc.aus.century.com" in {
           type master;
           file "named.abc.data";
           allow-transfer { key venus-batman.abc.aus.century.com.;};
           allow-update{ key venus-batman.abc.aus.century.com.; };
   };
   ```

   Because zones transfers are now restricted to those that have a key, the slave name server's named.conf file must also be edited. All requests to 192.9.201.1 (venus.abc.aus.century.com) are signed by a key. Note the name of the key (venus-batman.abc.aus.century.com.) must match those on the servers who use them.

   Below is an example of the named.conf file on the slave name server:

   ```
   // TSIG Key
   key venus-batman.abc.aus.century.com. {
           algorithm hmac-md5;
           secret "+UWSvbpxHWFdNwEAdy1Ktw==";
   };
   ```

```
server 192.9.201.1{
        keys { venus-batman.abc.aus.century.com.;};
};

options {
        directory "/usr/local/domain";
};

zone "abc.aus.century.com" IN {
    type slave;
    file "named.abc.data.bak";
    masters { 192.9.201.1; };
};
```

BIND 9 Transaction Signatures:

BIND 9 primarily supports Transaction Signatures (TSIG) for server-to-server communication.

This includes zone transfer, notify, and recursive query messages. TSIG is also useful for dynamic updates. A primary server for a dynamic zone should use access control to control updates, but IP-based access control is insufficient.

By using key base encryption rather than the current method of access control lists, TSIG can be used to restrict who can update to the dynamic zones. Unlike the Access Control List (ACL) method of dynamic updates, the TSIG key can be distributed to other updaters without having to modify the configuration files on the name server, which means there is no need for the name server to reread the configuration files.

It is important to note that BIND 9 does not have all the keywords implemented in BIND 8. In this example, we use the simple master configuration from BIND 8.

Note: To use named 9, you must relink the symbolic link to the **named** daemon to **named9**, and **nsupdate** to **nsupdate9** by running the following commands:

1. `ln -fs /usr/sbin/named9 /usr/sbin/named`
2. `ln -fs /usr/sbin/nsupdate9 /usr/sbin/nsupdate`

1. Generate the key using the **dnssec-keygen** command:

   ```
   dnssec-keygen -a HMAC-MD5 -b 128 -n HOST keyname
   ```

   - `HMAC-MD5` is the algorithm used for encryption
   - `128` is the length of the key to use (or number of bits)
   - `HOST`: `HOST` is the TSIG keyword used to generate a host key for shared key encryption.

   The command

   ```
   dnssec-keygen -a HMAC-MD5 -b 128 -n HOST venus-batman.abc.aus.century.com
   ```

   would produce two key files, as follows:

   ```
   Kvenus-batman.abc.aus.century.com.+157+35215.key
   Kvenus-batman.abc.aus.century.com.+157+35215.private
   ```

   - `157` is the algorithm used (HMAC-MD5)
   - `35215` is the finger print, which is useful in DNNSEC because multiple keys per zone are allowed
2. Add the entry to named.conf on the master name server:

   ```
   // TSIG Key
   key venus-batman.abc.aus.century.com. {
            algorithm hmac-md5;
            secret "+UWSvbpxHWFdNwEAdy1Ktw==";
   };
   ```

Assuming HMAC-MD5 is being used, both keyfiles contain the shared key, which are stored as the last entry in the files. Find a secure way to copy the shared secret key to the client. You do not need to copy the keyfile, just the shared secret key.

Following is the entry for file `Kvenus-batman.abc.aus.century.com.+157+35215.private`:

```
Private-key-format: v1.2
Algorithm: 157 (HMAC_MD5)
Key: +UWSvbpxHWFdNwEAdy1Ktw==
```

What follows is an example of the `named.conf` file for the master name server. The zone `abc.aus.century.com` allows zone transfer and dynamic updates only to servers with the key `venus-batman.abc.aus.century.com`. Do the same to the reverse zone, which requires updaters to have the shared key.

```
// TSIG Key
key venus-batman.abc.aus.century.com. {
         algorithm hmac-md5;
         secret "+UWSvbpxHWFdNwEAdy1Ktw==";
};

options {
         directory "/usr/local/domain";
};
zone "abc.aus.century.com" in {
         type master;
         file "named.abc.data";
         allow-transfer { key venus-batman.abc.aus.century.com.;};
         allow-update{ key venus-batman.abc.aus.century.com.; };
};
```

Because zones transfers are now restricted to those that have a key, the slave name server's `named.conf` file must also be edited. All requests to 192.9.201.1 (`venus.abc.aus.century.com`) are signed by a key. Note the name of the key (`venus-batman.abc.aus.century.com.`) must match those on the servers who use them.

Below is an example of the `named.conf` file on the slave name server:

```
// TSIG Key
key venus-batman.abc.aus.century.com. {
         algorithm hmac-md5;
         secret "+UWSvbpxHWFdNwEAdy1Ktw==";
};

server 192.9.201.1{
         keys { venus-batman.abc.aus.century.com.;};
};

options {
         directory "/usr/local/domain";
};

zone "abc.aus.century.com" IN {
     type slave;
     file "named.abc.data.bak";
     masters { 192.9.201.1; };
};
```

BIND 9 Signature:

BIND 9 partially supports DNSSEC SIG transaction signatures as specified in RFC 2535.

SIG uses public and private keys to authenticate messages.

SIG records allow administers to sign their zone data, thereby stating that it is authentic.

Securing the root zone:

When using these steps to secure the root zone, assume that other name servers on the internet are not using BIND 9, and you want to secure your zone data and allow other servers to verify your zone data.

You want to state that your zone (in our case `aus.century.com`) is a secure root, and will validate any secure zone data below it.

1. Generate the keys using the **dnssec-keygen** command:

   ```
   dnssec-keygen -a RSA -b 512 -r /usr/sbin/named -n ZONE aus.century.com.
   ```

   Note: RSA encryption can be used as the algorithm to generate the key if OpenSSL is installed, although you must first relink the DNS library to a secured DNS library by running the following command:

   ```
   ln -fs /usr/lib/libdns_secure.a /usr/lib/libdns.a
   ```

   - ZONE: ZONE is the DNSSEC keyword used to generate zone keys for private/public key encryption
   - The `r` flag specifies a random device
2. Add the public key entry similar to the `named.conf` file. The entry used in our case follows. Below are the contents of key file `Kaus.century.com.+001+03254.key`.

   ```
   abc.aus.century.com. IN KEY 256 3 1
   AQOnfGEAg0xpzSdNRe7KePq3Dl4NqQiq7HkwKl6TygUfaw6vz6ldmauB4UQFcGKOyL68/
   Zv5ZnEvyB1fMTAaDLYz
   ```

   The public key is contained in the file `Kzonename.+algor.+fingerprint.key`, or in our case `Kaus.century.com.+001+03254.key`. You have to remove the class IN and type KEY as well as quote the key. Once you add this entry to the `/etc/named.conf` file and refresh the name server, the zone `aus.century.com` is a secure root.

   ```
   trusted-keys {
           aus.century.com. 256 3 1 "AQOnfGEAg0xpzSdNRe7KePq3Dl4NqQiq7HkwKl6Tyg
   Ufaw6vz6ldmauB 4UQFcGKOyL68/Zv5ZnEvyB1fMTAaDLYz";
   };
   options {
           directory "/usr/local/domain";
   };

   zone "abc.aus.century.com" in {
           type master;
           file "named.abc.data.signed";
           allow-update{192.9.201.1;};
   };
   ```

Applying the chain of trust:

Now that you have a secured root, you can secure the rest of your child zones. In this case, we are working to secure the zone `abc.aus.century.com`.

Follow these steps to secure your remaining child zones:

1. Generate the key pairs using the **dnssec-keygen** command:

   ```
   dnssec-keygen -a RSA -b 512 -r /usr/sbin/named -n ZONE abc.aus.century.com.
   ```

   The `r` flag specifies a random input file.
2. Make a keyset by running the **dnssec-makekeyset** command:

   ```
   dnssec-makekeyset -t 172800 Kabc.aus.century.com.+001+11515.key
   ```

   where *Kabc.aus.century.com.+001+11515.key* is your own public key.

   This creates a keyset file called `keyset-abc.aus.century.com`.

3. Send this keyset file to the parent zone to get it signed. In this case, our parent zone is the secure root zone `aus.century.com`.
4. The parent must sign the key using its private key.

   ```
   dnssec-signkey keyset-abc.aus.century.com. Kaus.century.com.+001+03254.private
   ```

   This will generate a file called `signedkey-abc.aus.century.com`, and the parent will need to send this file back to the child zone.
5. On the child name server for zone `abc.aus.century.com`, add `$INCLUDE Kabc.aus.century.com.+001+11515.key` to the plain zone file `named.abc.data`. Remember to place the `signedkey-abc.aus.century.com` file in the same location as the zone file `named.abc.data`. When the zone is signed in the following step, the program will know to include `signedkey-abc.aus.century.com`, which was received from the parent.

   ```
   $TTL 3h    ;3 hour

   @ IN    SOA     venus.abc.aus.century.com. gail.zeus.abc.aus.century.com.  (
                   1       ;serial
                   3600    ;refresh
                   600     ;retry
                   3600000 ;expire
                   86400   ;negative caching TTL
   )
   $INCLUDE Kabc.aus.century.com.+001+03254.key
   ```
6. Sign the zone using the **dnssec-signzone** command:

   ```
   dnssec-signzone -o abc.aus.century.com. named.abc.data
   ```
7. Modify the **named.conf** file on the child zone `abc.aus.century.com` to use the new signed zone file (`named.abc.data.signed`). For example:

   ```
   options {
           directory "/usr/local/domain";
   };

   zone "abc.aus.century.com" in {
           type master;
           file "named.abc.data.signed";
           allow-update{192.9.201.1;};
   };
   ```
8. Refresh the name server.

For information on troubleshooting, see "Name resolution problems" on page 394.

Planning and configuring for LDAP name resolution (IBM SecureWay Directory schema)

The **Lightweight Directory Access Protocol (LDAP)** is an open industry standard that defines a method for accessing and updating information in a directory.

An **LDAP** schema defines the rules for ordering data. The **ibm-HostTable** object class, part of the IBM SecureWay Directory schema, can be used to store the name-to-Internet-address mapping information for every host on the network.

The **ibm-HostTable** object class is defined as follows:

```
Object Class name:     ibm-HostTable
Description:           Host Table entry which has a collection of hostname to
                       IP address mappings.
OID:                   TBD
RDN:                   ipAddress
Superior object class: top
Required Attributes:   host, ipAddress
Optional Attributes:   ibm-hostAlias, ipAddressType, description
```

The attribute definitions follow:

```
Attribute Name: ipAddress
Description:    IP Address of the hostname in the Host Table
OID:            TBD
Syntax:         caseIgnoreString
Length:         256
Single Valued:  Yes
Attribute Name: ibm-hostAlias
Description:    Alias of the hostname in the Host Table
OID:            TBD
Syntax:         caseIgnoreString
Length:         256
Single Valued:  Multi-valued
Attribute Name: ipAddressType
Description:    Address Family of the IP Address (1=IPv4, 2=IPv6)
OID:            TBD
Syntax:         Integer
Length:         11
Single Valued:  Yes
Attribute Name: host
Description:    The hostname of a computer system.
OID:            1.13.18.0.2.4.486
Syntax:         caseIgnoreString
Length:         256
Single Valued:  Multi-valued
Attribute Name: description
Description:    Comments that provide a description of a directory object entry.
OID:            2.5.4.13
Syntax:         caseIgnoreString
Length:         1024
Single Valued:  Multi-valued
```

Use the following procedure to configure the **LDAP** server compliant with the IBM SecureWay Directory schema, for storing the name-to-Internet-address mapping host information.

1. Add a suffix on the **LDAP** server. The suffix is the starting point of the hosts database. For example, "cn=hosts". This can done using the web-based IBM SecureWay Directory Server Administration tool.
2. Create an LDAP Data Interchange Format (LDIF) file. This can be done manually or with the **hosts2ldif** command, which creates a LDIF file from the `/etc/hosts` file. See the hosts2ldif Command for more information. The following is a sample LDIF file:

   ```
   dn: cn=hosts
   objectclass: top
   objectclass: container
   cn: hosts
   dn: ipAddress=1.1.1.1, cn=hosts
   host: test
   ipAddress: 1.1.1.1
   objectclass: ibm-HostTable
   ipAddressType: 1
   ibm-hostAlias: e-test
   ibm-hostAlias: test.austin.ibm.com
   description: first ethernet interface
   dn: ipAddress=fe80::dead, cn=hosts
   host: test
   ipAddress: fe80::dead
   objectclass: ibm-HostTable
   ipAddressType: 2
   ibm-hostAlias: test-ll
   ibm-hostAlias: test-ll.austin.ibm.com
   description: v6 link level interface
   ```

3. Import the hosts directory data from the LDIF file on the **LDAP** server. This can be done with the **ldif2db** command or through the web-based IBM SecureWay Directory Server Administration tool.

To configure the client to access the hosts database on the LDAP server, using the **LDAP** mechanism, follow these steps:

1. Create the `/etc/resolv.ldap` file. See the resolv.ldap File Format for TCP/IP in the *Files Reference* for more information and a detailed example of a `resolv.ldap` file.
2. Change the default name resolution through the **NSORDER** environment variable, the `/etc/netsvc.conf` file, or the `/etc/irs.conf` file. See the netsvc.conf File Format for TCP/IP or the irs.conf File Format for TCP/IP in the *Files Reference* for more information.

Although still supported, the use of `ldap` mechanism is deprecated. This existing `ldap` mechanism works with IBM SecureWay Directory Schema, while `nis_ldap` (NIS_LDAP) works with the RFC 2307 schema. Use of the `nis_ldap` mechanism instead of the `ldap` mechanism is recommended. For information on `nis_ldap` name resolution, see "Planning and configuring NIS_LDAP name resolution (RFC 2307 schema)."

Planning and configuring NIS_LDAP name resolution (RFC 2307 schema)

AIX 5.2 offers a new naming mechanism called NIS_LDAP.

The difference between the existing LDAP mechanism and the new NIS_LDAP mechanism is in the LDAP schema (the set of attributes and object classes that determine how attributes are grouped together for describing an entity). The existing LDAP mechanism works with the IBM SecureWay Directory schema compliant LDAP server and it supports only the host naming service. The NIS_LDAP mechanism works with the RFC 2307 schema compliant LDAP server, and it supports all the NIS services: users and groups, hosts, services, protocols, networks, and netgroup. RFC 2307 defines a set of attributes and object classes that can be used to describe network information services, including users and groups.

- To configure the LDAP server, you will need to set up the LDAP server and migrate the required data to the server.
  1. Use the **mksecldap** command to set up a server. The `nis_ldap` mechanism works only with the RFC 2307 schema. While setting up the LDAP server, the **mksecldap** command should be invoked with either the `-S rfc2307` or `-S rfc2307aix` option (not the `-S aix` option, which specifies the IBM SecureWay Directory schema). By default, the **mksecldap** command migrates users and groups defined on the local system to the LDAP server. If you want to disable this migration, use the `-u NONE` option.

     ```
     mksecldap -s -a cn=admin -p adminpwd -S rfc2307aix
     ```

     This sets up an LDAP server with administrator DN being `cn=admin` and password being `adminpwd`. The default suffix, `cn=aixdata`, is also added to the `/etc/slapd32.conf` file, the LDAP server configuration file.

     By default, the **mksecldap** command migrates users and groups defined on the local system to the LDAP server. If you want to disable this migration, use the `-u NONE` option, which prevents the migration of local users and groups to the LDAP server, so that you can only add NIS users and groups later.

     ```
     mksecldap -s -a cn=admin -p adminpwd -u NONE
     ```

     For more information on the **mksecldap** command, see the command description in *Commands Reference, Volume 3*.
  2. Migrate the NIS data. Use the **nistoldif** command from the NIS server to migrate the NIS maps to the LDAP server. The **nistoldif** command can also be used to migrate data from flat files. Run the **nistoldif** command on a system that contains NIS data that needs to be migrated to the LDAP server.

     ```
     nistoldif -h server1.ibm.com -a cn=admin -p adminpwd -d cn=aixdata
     ```

This migrates the NIS maps from the local system to the LDAP server, `server1.ibm.com`. The NIS data is placed under the `cn=aixdata` DN. You can also run the **nistoldif** command to migrate data from flat files on any system to the LDAP server. The flat files will be used for any maps missing from the NIS server.

For more information on the **nistoldif** command, see the command description in *Commands Reference, Volume 4*.

Note: Names are represented by the `cn` attribute of the LDAP server. The `cn` attribute defined by RFC 2307 is not case-sensitive. Names that differ only by case will be merged on the server. Matches are also not case-sensitive. Searching for `TCP`, `tcp`, or `Tcp` would all return the protocol entry for TCP.

- To configure the LDAP client to access names from the LDAP server, run the **mksecldap** command with client setup options.
  1. The **mksecldap** command saves the LDAP server name, port, admindn, password, and basedn to the `/etc/security/ldap/ldap.cfg` file, which is read by the **secldapclntd** daemon at its startup time. The **mksecldap** command starts the **secldapclntd** daemon automatically, if the setup is successful.

     See the `/etc/security/ldap/ldap.cfg` file in *Files Reference* and the **secldapclntd** daemon in the *Commands Reference, Volume 5* for more information.
  2. The **mksecldap** command adds `nis_ldap` mechanism to the `/etc/netsvc.conf` file and the `/etc/irs.conf` file so that name resolution can be directed to LDAP. You can also manually set the **NSORDER** environment variable to `nis_ldap` to use the NIS_LDAP name resolution.

     ```
     mksecldap -c -a cn=admin -p adminpwd -h server1.ibm.com
     ```

     This sets up the local system to use the `server1.ibm.com` LDAP server. The LDAP server administrator DN and password must be supplied for this client to authenticate to the server. The `/etc/netsvc.conf` and the `/etc/irs.conf` files are updated so that the naming resolution is resolved through NIS_LDAP.

     See the `/etc/netsvc.conf` file format for TCP/IP or the `/etc/irs.conf` file format for TCP/IP in the *Files Reference* for more information.
  3. Naming resolution for users and groups is not controlled by the `/etc/netsvc.conf` or `/etc/irs.conf` files. Rather it is through the `/etc/security/user` file. To enable a LDAP user to login to an AIX system, set the user's `SYSTEM` and `registry` variables to `LDAP` in the `/etc/security/user` file of that client system. You can run the **chuser** command to do this.

     ```
     chuser -R LDAP SYSTEM=LDAP registry=LDAP foo
     ```

     You can configure your system to allow all LDAP users to login to a system. To do so, edit the `/etc/security/user` file. Add `registry = files` to the root stanza. Then add `SYSTEM = LDAP` and `registry = LDAP` to the default stanza.

     For more information on user authentication, refer to Light Directory Access Protocol in *Security*.

Related information:

Migrating from NIS to RFC 2307-compliant LDAP services

TCP/IP address and parameter assignment - Dynamic Host Configuration Protocol

Transmission Control Protocol/Internet Protocol (TCP/IP) enables communication between machines with configured addresses. Part of the burden a network administrator must face is address assignment and parameter distribution for all machines on the network. Commonly, this is a process in which the administrator dictates the configuration to each user, allowing the user to configure his own machine. However, misconfigurations and misunderstandings can generate service calls that the administrator must deal with individually. The **Dynamic Host Configuration Protocol (DHCP)** gives the network administrator a method to remove the end user from this configuration problem and maintain the network configuration in a centralized location.

DHCP is an application-layer protocol that allows a client machine on the network, to get an IP address and other configuration parameters from the server. It gets information by exchanging packets between a daemon on the client and another on the server. Most operating systems now provide a **DHCP** client in their base package.

To obtain an address, the **DHCP** client daemon (**dhcpcd**) broadcasts a **DHCP** discover message, which is received by the server and processed. (Multiple servers can be configured on the network for redundancy.) If a free address is available for that client, a **DHCP** offer message is created, This message contains an IP address and other options that are appropriate for that client. The client receives the server **DHCP** offer and stores it while waiting for other offers. When the client chooses the best offer, it broadcasts a **DHCP** request that specifies which server offer it wants.

All configured **DHCP** servers receive the request. Each checks to see if it is the requested server. If not, the server frees the address assigned to that client. The requested server marks the address as assigned and returns a **DHCP** acknowledgment, at which time, the transaction is complete. The client has an address for the period of time (lease) designated by the server.

When half of the lease time is used, the client sends the server a *renew* packet to extend the lease time. If the server is willing to renew, it sends a **DHCP** acknowledgment. If the client does not get a response from the server that owns its current address, it broadcasts a **DHCP** rebind packet to reach the server if, for example, the server has been moved from one network to another. If the client has not renewed its address after the full lease time, the interface is brought down and the process starts over. This cycle prevents multiple clients on a network from being assigned the same address.

The **DHCP** server assigns addresses based on keys. Four common keys are network, class, vendor, and client ID. The server uses these keys to get an address and a set of configuration options to return to the client.

network
Identifies which network segment the packet came from. The network key allows the server to check its address database and assign an address by network segment.

class Is completely client configurable. It can specify an address and options. This key can be used to denote machine function in the network or to describe how machines are grouped for administrative purposes. For example, the network administrator might want to create a `netbios` class that contains options for NetBIOS clients or an `accounting` class that represents Accounting department machines that need access to a specific printer.

vendor
Helps identify the client by its hardware/software platform (for example, a Microsoft Windows 95 client or an OS/2 Warp client).

client ID
Identifies the client either through the machine host name or its medium access control (MAC) layer address. The client ID is specified in the configuration file of the **dhcpcd** daemon. Also, the client ID can be used by the server to pass options to a specific client or prohibit a particular client from receiving any parameters.

These keys can be used by the configuration either singularly or in combinations. If multiple keys are provided by the client and multiple addresses can be assigned, only one is chosen, and the option set is derived from the chosen key first. For more detailed information about the selection of keys and addresses, see "DHCP configuration" on page 205.

A relay agent is needed so initial broadcasts from the client can leave the local network. This agent is called the BOOTP relay agent. The relay agents act as forwarding agents for **DHCP** and **BOOTP** packets.

DHCP servers

In the AIX operating system, the **DHCP** server has been segmented into three main pieces.

The main components of the **DHCP** server are a database, a protocol engine, and a set of service threads, each with its own configuration information.

DHCP database:

The `db_file.dhcpo` database is used to track clients and addresses and for access control (for example, allowing certain clients on some networks but not others, or disabling **BOOTP** clients on a particular network).

Options are also stored in the database for retrieval and delivery to clients. The database is implemented as a dynamically loadable object, which allows for easy server upgrade and maintenance.

Using the information in the configuration file, the database is primed and verified for consistency. A set of checkpoint files handles updates to the database and reduces the overhead of writes to the main storage file. The database also contains the address and option pools, but these are static and are discussed in "DHCP configuration" on page 205.

The main storage file and its back up are flat ASCII files that can be edited. The format for the database main storage files are:

```
DF01
"CLIENT ID" "0.0.0.0" State LeaseTimeStart LeaseTimeDuration LeaseTimeEnd
  "Server IP Address" "Class ID" "Vendor ID" "Hostname" "Domain Name"
"CLIENT ID" "0.0.0.0" State LeaseTimeStart LeaseTimeDuration LeaseTimeEnd
  "Server IP Address" "Class ID" "Vendor ID" "Host Name" "Domain Name"
...
```

The first line is a version identifier for the file: `DF01c`. The lines that follow are client record definition lines. The server reads from the second line to the end of the file. (The parameters in quotes must be enclosed in quotes.)

"*CLIENT ID***"**
The ID the client uses to represent itself to the server.

"0.0.0.0"
is the IP address currently assigned to the **DHCP** server. If no address has been assigned, it is "0.0.0.0".

State The current state of the client. The **DHCP** protocol engine contains the allowable set, and the states are maintained in the **DHCP** database. The number next to *State* represents its value. The states can be:

(1) FREE
Represents addresses that are available for use. In general, clients do not have this state unless they have no address assigned. **dadmin** and the output from **lssrc** report this state as `Free`.

(2) BOUND
Indicates client and address are tied and that the client has been assigned this address for some amount of time. **dadmin** and the output from **lssrc** report this state as `Leased`.

(3) EXPIRED
Indicates the client and address are tied together, but only for informational purposes, in a similar manner to released addresses. The expired state, however, represents clients that let their leases expire. An expired address is available for use and is reassigned after all free addresses are unavailable and before released addresses are reassigned. **dadmin** and the output from **lssrc** report this state as `Expired`.

(4) RELEASED
Indicates the client and address are tied for informational purposes only. The **DHCP** protocol suggests that **DHCP** servers maintain information about the clients it has served

for future reference (mainly to try giving the same address to that client that has been assigned that address in the past). This state indicates that the client has released the address. The address is available for use by other clients, if no other addresses are available. **dadmin** and the output from **lssrc** report this as `Released`.

(5) RESERVED
Indicates client and address are tied, but loosely. The client has issued a **DHCP** discover message and the **DHCP** server has responded, but the client has not yet responded with a **DHCP** request for that address. **dadmin** and the output from **lssrc** report this state as `Reserved`.

(6) BAD
Represents an address that is in use in the network but has not been handed out by the **DHCP** server. This state also represents addresses that clients have rejected. This state does not apply to clients. **dadmin** and the output from **lssrc** report this state as `Used` and `Bad`, respectively.

LeaseTimeStart
Is the start of the current lease time (in the number of seconds since January 1, 1970).

LeaseTimeDuration
Represents the duration of the lease (in seconds).

LeaseTimeEnd
Uses the same format as *LeaseTimeStart*, but it represents the end of the lease. Some configuration options use different values for the start and end of a lease and these values can be overridden by configuration file options. See "DHCP server file syntax for db_file database" on page 220.

"*Server IP Address*"
Is the IP address of the DHCP server that owns this record.

"*Class ID*" "*Vendor ID*" "*Host Name*" "*Domain Name*"
Values that the server uses to determine which options are sent to the server (stored as quoted strings). These parameters increase performance because option lists can be pregenerated for these clients when the **DHCP** server starts up.

DHCP checkpoint files:

The syntax for the checkpoint files is not specified.

If the server crashes or you have to shut down and cannot do a normal closing of the database, the server can process the checkpoint and backup files to reconstruct a valid database. The client that is being written to the checkpoint file when the server crashes is lost. The default files are:

/etc/db_file.cr
normal database operation

/etc/db_file.crbk
backups for the database

/etc/db_file.chkpt and /etc/db_file.chkpt2
rotating checkpoint files

The **DHCP** server is threaded. To maintain high throughput, database operations (including save operations) are thread-efficient. When a save is requested, the existing checkpoint file is rotated to the next checkpoint file, the existing database file is copied to the backup file, and the new save file is created. Each client record is then logged and a bit is toggled to indicate that the client should use the new checkpoint file for logging. When all client records are recorded, the save is closed, and the backup and old checkpoint files are deleted. Clients can still be processed and, depending on whether the client record has been saved, database changes go into a new save file or to a new checkpoint file.

The DHCP protocol engine:

The **DHCP** protocol engine supports RFC 2131 and is still compatible with RFC 1541. (The server can also process options as defined in RFC 2132.) The protocol engine uses the database to determine what information is returned to the client.

The configuration of the address pools have some configuration options that affect the state of each machine. For example, the **DHCP** server pings addresses before it hands them out. The amount of time the server waits for a response is now configurable for each address pool.

DHCP threaded operations:

The last piece of the **DHCP** server is actually a set of operations that are used to keep things running. Because the **DHCP** server is threaded, these operations are actually set up as threads that occasionally do things to make sure everything is together.

The first thread, the **main** thread, handles the SRC requests (such as **startsrc**, **stopsrc**, **lssrc**, **traceson**, and **refresh**). This thread also coordinates all operations that affect all threads and handles signals. For example,

- A SIGHUP (-1) causes a refresh of all databases in the configuration file.
- A SIGTERM (-15) will cause the server to gracefully stop.

The next thread, the **dadmin** thread, interfaces with **dadmin** client program and the **DHCP** server. The **dadmin** tool can be used to get status as well as modify the database to avoid editing the database files manually. Previous versions of the **DHCP** server prevented any clients from getting addresses if a status request was running. With the addition of the **dadmin** and **src** threads, the server can handle service requests and still handle client requests.

The next thread is the **garbage** thread, which runs timers that periodically clean the database, save the database, purge clients that do not have addresses, and remove reserved addresses that have been in reserve state for too long. All these timers are configurable (see "DHCP configuration" on page 205). The other threads are packet processors. The number of these is configurable; the default is 10. Each of these can handle a request from a **DHCP** client. The number of packet processors required is somewhat load- and machine-dependent. If the machine is used for other services than **DHCP**, it is not wise to start up 500 threads.

DHCP planning

To use this protocol, the network administrator needs to set up a **DHCP** server and configure BOOTP relay agents on links that do not have a **DHCP** server. Advance planning can reduce **DHCP** load on the network.

For example, one server can be configured to handle all your clients, but all packets must be passed through it. If you have a single router between two large networks, it is wiser to place two servers in your network, one on each link.

Another aspect to consider is that **DHCP** implies a pattern of traffic. For example, if you set your default lease time to fewer than two days and your machines are powered off for the weekend, Monday morning becomes a period of high **DHCP** traffic. Although **DHCP** traffic does not cause huge overhead for the network, it needs to be considered when deciding where to place **DHCP** servers on a network and how many to use.

After enabling **DHCP** to get the client on the network, a client has no requirement to enter anything. The **DHCP** client, `dhcpcd`, reads the `dhcpcd.ini` file, which contains information on logging and other parameters needed to start running. After installation, decide which method to use for **TCP/IP** configuration: minimum configuration or **DHCP**. If **DHCP** is selected, choose an interface and specify

some optional parameters. To choose the interface, select the keyword **any**, which tells `dhcpcd` to find the first interface that works and use it. This method minimizes the amount of input on the client side.

DHCP configuration

By default, the **DHCP** server is configured by reading the `/etc/dhcpsd.cnf` file, which specifies the initial database of options and addresses.

The server is started in the `/etc/rc.tcpip` file. It can also be started from SMIT, or through SRC commands. The **DHCP** client can be configured by running the System Management Interface Tool (SMIT), or editing a flat ASCII file.

Configuring the **DHCP** server is usually the hardest part of using **DHCP** in your network. First, decide what networks you want to have **DHCP** clients on. Each subnet in your network represents a pool of addresses that the **DHCP** server must add to its database. For example:

```
database db_file
{

    subnet 9.3.149.0 255.255.255.0
      {  option 3 9.3.149.1 # The default gateway clients on this network should use
         option 6 9.3.149.2 # The nameserver clients on this network should use
      }
    ... options or other containers added later
}
```

The example above shows a subnet, `9.3.149.0`, with a subnet mask `255.255.255.0`. All addresses in this subnet, 9.3.149.1 through 9.3.149.254, are in the pool. Optionally, a range can be specified on the end of the line or a range or exclude statement can be included in the subnet container. See “DHCP server file known options” on page 213 for common configuration methods and definitions.

The database clause with `db_file` indicates which database method to use for processing this part of the configuration file. Comments begin with a # (pound sign). Text from the initial #, to the end of the line, is ignored by the **DHCP** server. Each `option` line is used by the server to tell the client what to do. “DHCP server file known options” on page 213 describes the currently supported and known options. See “DHCP server file syntax for general server operation” on page 216 for ways to specify options that the server does not know about.

If the server does not understand how to parse an option, it uses default methods to send the option to the client. This also allows the **DHCP** server to send site-specific options that are not RFC defined, but can be used be certain clients or client configurations.

DHCP configuration file:

The configuration file has an address section and an option definition section. These sections use containers to hold options, modifiers, and, potentially, other containers.

A *container* (basically, a method to group options) uses an identifier to classify clients into groups. The container types are subnet, class, vendor, and client. Currently, there is not a generic user-definable container. The identifier uniquely defines the client so that the client can be tracked if, for example, it moves between subnets. More than one container type can be used to define client access.

Options are identifiers that are returned to the client, such as default gateway and DNS address.

Modifiers are single statements that modify some aspect of a container, such as lease time default.

DHCP containers:

When the **DHCP** server receives a request, the packet is parsed and identifying keys determine which containers, options, and addresses are extracted.

The example in "DHCP configuration" on page 205shows a subnet container. Its identifying key is the position of the client in the network. If the client is from that network, then it falls into that container.

Each type of container uses a different option to identify a client:

- The subnet container uses the **giaddr** field or the interface address of the receiving interface to determine from which subnet the client came.
- The class container uses the value in option 77 (User Site Class Identifier).
- The vendor uses the value in option 60 (Vendor Class Identifier).
- The client container uses the option 61 (Client Identifier) for **DHCP** clients and the **chaddr** field in the **BOOTP** packet for **BOOTP** clients.

Except for subnets, each container allows the specification of the value that matches it, including regular expression matching.

There is also an implicit container, the *global* container. Options and modifiers are placed in the global container unless overridden or denied. Most containers can be placed inside other containers implying a scope of visibility. Containers may or may not have address ranges associated with them. Subnets, by their nature, have ranges associated with them.

The basic rules for containers and subcontainers are:

- All containers are valid at the global level.
- Subnets can not be placed inside other containers.
- Restricted containers cannot have regular containers of the same type within them. (For example, a container with an option that only allows a class of `Accounting` cannot include a container with an option that allows all classes that start with the letter "a". This is illegal.)
- Restricted client containers cannot have subcontainers.

Given the above rules, you can generate a hierarchy of containers that segment your options into groups for specific clients or sets of clients.

If a client matches multiple containers, how are options and addresses handed out? The **DHCP** server receives messages, it passes the request to the database (`db_file` in this case), and a container list is generated. The list is presented in order of depth and priority. Priority is defined as an implicit hierarchy in the containers. Strict containers are higher priority than regular containers. Clients, classes, vendors, and finally subnets are sorted, in that order, and within container type by depth. This generates a list ordered by most specific to least specific. For example:

```
Subnet 1
--Class 1
--Client 1
Subnet 2
--Class 1
----Vendor 1
----Client 1
--Client 1
```

The example shows two subnets, `Subnet 1` and `Subnet 2`. There is one class name, `Class 1`, one vendor name, `Vendor 1`, and one client name, `Client 1`. `Class 1` and `Client 1` are defined in multiple places. Because they are in different containers, their names can be the same but values inside them can be different. If `Client 1` sends a message to the **DHCP** server from `Subnet 1` with `Class 1` specified in its option list, the **DHCP** server would generate the following container path:

`Subnet 1, Class 1, Client 1`

The most specific container is listed last. To get an address, the list is examined in reverse hierarchy to find the first available address. Then, the list is examined in forward hierarchy to get the options. Options override previous values unless an option **deny** is present in the container. Also, because `Class 1` and `Client 1` are in `Subnet 1`, they are ordered according to the container priority. If the same client is in `Subnet 2` and sends the same message, the container list generated is:

`Subnet 2, Class 1, Client 1` (at the `Subnet 2` level), `Client 1` (at the `Class 1` level)

`Subnet 2` is listed first, then `Class 1`, then the `Client 1` at the `Subnet 2` level (because this client statement is only one level down in the hierarchy). The hierarchy implies that a client matching the first client statement is less specific than the client matching `Client 1` of `Class 1` within `Subnet 2`.

Priority selected by depth within the hierarchy is not superseded by the priority of the containers themselves. For example, if the same client issues the same message and specifies a vendor identifier, the container list is:

`Subnet 2, Class 1, Vendor 1, Client 1` (at `Subnet 2` level), `Client 1` (at `Class 1` level)

Container priority improves search performance because it follows a general concept that client containers are the most specific way to define one or more clients. The class container holds less specific addresses than a client container; vendor is even less specific; and subnet is the least specific.

DHCP addresses and address ranges:

Any container type can have associated addresses ranges; subnets must have associated address ranges. Each range within a container must be a subset of the range and must not overlap with ranges of other containers.

For example, if a class is defined within a subnet and the class has a range, the range must be a subset of the subnet range. Also, the range within that class container cannot overlap with any other ranges at its level.

Ranges can be expressed on the container line and modified by range and exclude statements to allow for disjoint address sets associated with a container. If you have the top ten addresses and the second ten addresses of a subnet available, the subnet can specify these addresses by range in the subnet clause to reduce both memory use and the chance of address collision with other clients not in the specified ranges.

After an address has been selected, any subsequent container in the list that contains address ranges is removed from the list along with its children. Network-specific options in removed containers are not valid if an address is not used from within that container.

DHCP configuration file options:

After the list has been culled to determine addresses, a set of options is generated for the client.

In this selection process, options overwrite previously selected options unless a *deny* is encountered, in which case, the denied option is removed from the list being sent to the client. This method allows inheritance from parent containers to reduce the amount of data that must be specified.

DHCP modifiers:

Modifiers are items that change some aspect of a particular container, such as access or lease time.

Define the address and option pools before modifying the container. The most common modifiers are **leasetimedefault**, **supportBootp**, and **supportUnlistedclients**.

leasetimedefault
Defines the amount of time an address is to be leased to a client.

supportBootp
Defines whether or not the server responds to **BOOTP** clients.

supportUnlistedclients
Indicates whether clients are to be explicitly defined by a client statement to receive addresses. The value for **supportUnlistedClients** can be **none**, **dhcp**, **bootp**, or **both**. This allows for you to restrict access to bootp client and allow all DHCP clients to get addresses.

Other modifiers are listed in "DHCP server file syntax for db_file database" on page 220.

DHCP logging:

After selecting modifiers, the next item to set up is logging.

Logging parameters are specified in a container like the database, but the container keyword is **logging_info**. When learning to configure **DHCP**, it is advisable to turn logging to its highest level. Also, it is best to specify the logging configuration before any other configuration file data to ensure that configuration errors are logged after the logging subsystem is initialized. Use the **logitem** keyword to turn on a logging level or remove the **logitem** keyword to disable a logging level. Other keywords for logging allow the specification of the log filename, file size, and the number of rotating log files.

DHCP server-specific options:

The last set of parameters to specify are server-specific options that allow the user to control the number of packet processors, how often the garbage collection threads are run, and so on.

For example, two server-specific options are:

reservedTime
Indicates how long an address stays in the reserved state after sending an OFFER to the **DHCP** client

reservedTimeInterval
Indicates how often the **DHCP** server scans through the addresses to see if there are any that have been in the reserved state longer than **reservedTime**.

These options are useful if you have several clients that broadcast DISCOVER messages and, either they do not broadcast their REQUEST message, or their REQUEST message gets lost in the network. Using these parameters keeps addresses from being reserved indefinitely for a noncompliant client.

Another particularly useful option is **SaveInterval**, which indicates how often saves occur. All server-specific options are listed in "DHCP server file syntax for general server operation" on page 216 with the logging keywords.

DHCP performance considerations:

It is important to understand that certain configuration keywords and the structure of the configuration file have an effect on the memory use and performance of the **DHCP** server.

First, excessive memory use can be avoided by understanding the inheritance model of options from parent to child containers. In an environment that supports no unlisted clients, the administrator must explicitly list each client in the file. When options are listed for any specific client, the server uses more memory storing that configuration tree than when options are inherited from a parent container (for

example, the subnet, network, or global containers). Therefore, the administrator should verify whether any options are repeated at the client level within the configuration file and determine whether these options can be specified in the parent container and shared by the set of clients as a whole.

Also, when using the **logItem** entries INFO and TRACE, numerous messages are logged during the processing of every **DHCP** client message. Appending a line to the log file can be an expensive operation; therefore, limiting the amount of logging improves the performance of the **DHCP** server. When an error with the **DHCP** server is suspected, logging can be dynamically re-enabled using either the SRC **traceson** or **dadmin** commands.

Finally, selecting a **numprocessors** value depends on the size of the **DHCP**-supported network, the **pingTime db_file** configuration parameter, and the typical propagation delay on the network. Because each packet processor thread issues an ICMP Echo Request to verify the status of a server-owned address before offering it to a client, the amount of time that any Echo Response is waited for directly affects the amount of processing time for a DISCOVER message. Essentially, the packet processor thread is able to do nothing more than wait for any response or for the **pingTime** timeout. Lowering the **numprocessors** value improves the response time of the server by lowering the number of client retransmissions, yet still maintaining the ping benefit of the server design.

For best performance, select a **pingTime** based on the propagation delay of any remote networks supported by the **DHCP** server. Also, select the **numprocessors** value based on this **pingTime** value and the size of the network. Selecting a value that is too small can cause all packet processing threads to be stopped. The server is then caused to wait for any Echo Responses while incoming **DHCP** client messages are queueing on the server port. This causes the server to handle client messages in batches rather than in a constant stream.

A selected value that is too small can cause all packet processing threads to be stopped waiting for any Echo Responses.

To prevent this situation, set the value for **numprocessors** to a number higher than the estimated number of DISCOVER messages that can be received within one **pingTime** interval during a period of high **DHCP** client activity. However, do not set the **numprocessors** value so high that it could burden the kernel with thread management.

For example, the values **numprocessors 5** and **pingTime 300** cause poor performance in an environment with a potential 10 DISCOVER messages per second because at peak demand, only 5 messages are handled every 3 seconds. Configure this environment with values similar to **numprocessors 20** and **pingTime 80**.

DHCP configuration file customization:

There are various factors involved in customizing your **DHCP** configuration file.

Many networks include multiple client types; for example, a single network may include computers running a variety of operating systems, such as Windows, OS/2, Java™ OS, and UNIX. Each of these require unique vendor identifiers (the field used to identify the type of machine to the DHCP server). Java OS clients and IBM Thin Client machines can require unique parameters such as bootfiles, and configuration options that need to be tailored specifically for them. Windows 95 computers do not handle Java-specific options well.

Machine-specific options can be encapsulated within vendor containers if the primary use for certain machines is based on the type of user for those machines. For instance, the development staff might use this operating system's clients for programming, the marketing staff might use the OS/2 clients, sales might use Java OS clients and IBM Thin Client machines, and accounting might use Windows 95 machines. Each of these user families might need different configuration options (different printers, name

servers, or default web servers, and so forth). In this case, such options could be included in the vendor container, since each group uses a different machine type.

If the same machine type is used by multiple groups, placing the options within a subordinate class identifier instead, would allow your marketing managers, for example, to use a specific set of printers that other employees could not access.

Note: The following fictional example represents part of a configuration file. Comments are preceded by a pound sign (#) and describe how each line defines the installation.

```
vendor "AIX_CLIENT"
{
# No specific options, handles things based on class
}

vendor "OS/2 Client"
{
# No specific options, handles things based on class
}

vendor "Windows 95"
{ option 44 9.3.150.3          # Default NetBIOS Nameserver
}

vendor "Java OS"
{ bootstrapserver 9.3.150.4    # Default TFTP server for the Java OS boxes
  option 67 "javaos.bin"       # The bootfile of the Java OS box
}

vendor "IBM Thin Client"
{ bootstrapserver 9.3.150.5    # Default TFTP server for Thin Client boxes
  option 67 "thinos.bin"       # Default bootfile for the Thin Client boxes
}

subnet 9.3.149.0 255.255.255.0
{ option 3 9.3.149.1           # The default gateway for the subnet
  option 6 9.3.150.2           # This is the nameserver for the subnet
  class accounting 9.3.149.5-9.3.149.20
  {       # The accounting class is limited to address range 9.3.149.5-9.3.149.20
          # The printer for this group is also in this range, so it is excluded.
     exclude 9.3.149.15
     option 9 9.3.149.15       # The LPR server (print server)
     vendor "Windows 95"
     {
     option 9 deny              # This installation of Windows 95 does not support
                                # this printer, so the option is denied.
     }
  }
 . . .
}
```

DHCP and the Dynamic Domain Name System

The **DHCP** server provides options that enable operation in a Dynamic Domain Name System (DDNS) environment.

To use **DHCP** in a DDNS environment, you must set and use a Dynamic Zone on a DNS server.

After the DDNS server is configured, decide if the **DHCP** server is going to do A-record updates, PTR-record updates, updates for both record types, or none at all. This decision depends on whether a client machine can do part or all of this work.

- If the client can share update responsibility, configure the server to do the PTR-record updates and configure the client to do the A-record updates.
- If the client can do both updates, configure the server to do none.

- If the client cannot do updates, configure the server to do both.

The **DHCP** server has a set of configuration keywords that allow you to specify a command to run when an update is required. These are:

updatedns
(Deprecated.) Represents the command to issue to do any type of update. It is called for both the PTR-record and the A-record update.

updatednsA
Specifies the command to update the A-record.

updatednsP
Specifies the command to update the PTR-record.

These keywords specify executable strings that the **DHCP** server runs when an update is required. The keyword strings must contain four %s (percent symbol, letter s). The first %s is the host name; the second is the domain name; the third is the IP address; and the fourth is the lease time. These are used as the first four parameters for the **dhcpaction** command. The remaining two parameters for the **dhcpaction** command indicate the record to update (A, PTR, NONE, or BOTH) and whether NIM should be updated (NIM or NONIM). See "DHCP and Network Installation Management suggestions" on page 262 for more information about NIM and **DHCP** interaction. For example:

```
updatednsA "/usr/sbin/dhcpaction '%s' '%s' '%s' '%s' '%s' A NONIM"
                    # This does the dhcpaction command only on the A record
updatednsP "/usr/sbin/dhcpaction '%s' '%s' '%s' '%s' '%s' PTR NONIM"
                    # This does the command only on the PTR record
updatedns "/usr/sbin/dhcpaction '%s' '%s' '%s' '%s' '%s' BOTH NIM"
                    # This does the command on both records and updates NIM
```

The **DHCP** server also has a set of keywords to remove the DNS entries when a lease is released or expires. The keywords are:

releasednsA
Removes the A-record.

releasednsP
Removes the PTR-record.

removedns
Removes both record types.

These keywords specify executable strings that the **DHCP** server runs when an address is released or expired. The **dhcpremove** command works similarly to **dhcpaction**, but only takes three parameters:

1. The IP address, specified as a %s in the command string
2. Which record to remove (A, PTR, NONE, or BOTH).
3. Whether NIM should be updated (NIM or NONIM).

For example:

```
releasednsA "/usr/sbin/dhcpremove '%s' A NONIM"
                     # This does the dhcpremove command only the A record
releasednsP "/usr/sbin/dhcpremove '%s' PTR NONIM"
                     # This does the command only on the PTR record
removedns "/usr/sbin/dhcpremove '%s' BOTH NIM"
                     # This does the command on both records and updates NIM
```

The **dhcpaction** and **dhcpremove** scripts do some parameter checking, then set up a call to **nsupdate**, which has been updated to work with this operating system's servers and with OS/2 DDNS servers. See the **nsupdate** command description for more information.

If NIM interaction is **NOT** required by the name update, the DHCP server can be configured to use a socket transfer between the **DHCP** daemon and the **nsupdate** command to improve performance and enable DNS updates to be retried upon failure. To configure this option, the **updateDNSA**, **updateDNSP**, **releaseDNSA**, or the **releaseDNSP** keyword must specify "nsupdate_daemon" as the first quoted word. The parameters and flags for this update are identical to those that are accepted by the **nsupdate** command. Additionally, the following variable names can be used for substitution:

Item	Description
$hostname	Replaced by the host name of the client on DNS update or the host name previously associated with the client for DNS removal.
$domain	Replaced by the DNS domain for the update or the previously used domain of the client host name for a DNS removal.
$ipadress	Replaced by the IP address to be associated or disassociated from the **DHCP** client name.
$leasetime	Replaced by the lease time (in seconds).
$clientid	Replaced by the string representation of the **DHCP** client identifier or the combination hardware type and hardware address for **BOOTP** clients.

For example:

```
updateDNSA "nsupdate_daemon -p 9.3.149.2 -h $hostname -d $domain
 -s"d;a;*;a;a;$ipaddress;s;$leasetime;3110400""

updateDNSP "nsupdate_daemon -p 9.3.149.2 -r $ipaddress
 -s"d;ptr;*;a;ptr;$hostname.$domain.;s;$leasetime;3110400""

releaseDNSA "nsupdate_daemon -p 9.3.149.2 -h $hostname -d $domain -s"d;a;*;s;1;3110400""

releaseDNSP "nsupdate_daemon -p 9.3.149.2 -r $ipaddress -s"d;ptr;*;s;1;3110400""
```

See the **nsupdate** command description for more information.

Also, administrator-defined policies have been added for hostname exchanges between the server and the clients. By default, the hostname that is returned to the client and used for a DDNS update is option 12 (defined in the server configuration file). Alternatively, the default hostname can be the client-suggested hostname, either through option 81 (the DHCPDDNS option) or through option 12 (the HOSTNAME option). However, the administrator can override the default hostname by using the **hostnamepolicy**, **proxyarec**, and **appenddomain** configuration keywords. These options and their parameters are defined in "DHCP server file syntax for db_file database" on page 220.

DHCP compatibility with older versions

The **DHCP** server recognizes the previous versions configuration and database files, `dhcps.ar` and `dhcps.cr`.

It parses the old configuration files and generates new database files in the old locations. The old databases are converted automatically to the new file. The configuration file itself is not converted.

The **DHCP** server database module, `db_file`, can read the old format. The **DHCP** server can recognize when a database container is not in the configuration file and treats the whole file as configuring the server parameters, logging parameters, and the `db_file` database parameters.

Note:

1. Some old configuration file syntax is deprecated, but is still supported. Other deprecations are as follows:
2. The network container is completely deprecated. To specify correctly, either convert the network clause with a range into a valid subnet container with a subnet address, subnet netmask, and the range. If the network container has subnet containers, remove the network container keyword and its

braces and then place the subnet mask in the appropriate place on the line. To start using the database container, group everything that pertains to networks and client access into one database container of type `db_file`.

3. The **updatedns** and **removedns** keywords are deprecated and replaced in favor of specifying the action for A and PTR records separately.
4. The **clientrecorddb** and **addressrecorddb** keywords have been deprecated to **clientrecorddb** and **backupfile**, respectively.
5. The **option sa** and **option ga** keywords have been replaced by **bootstrapserver** and **giaddrfield** keywords, respectively. See "DHCP server file syntax for general server operation" on page 216 and "DHCP server file syntax for db_file database" on page 220 for more information.

DHCP server file known options

The known options of the **DHCP** server file are identified here.

Note: The options that are shown in this table as not allowed to be specified (No in the Can Specify? column) can be specified in the configuration file, but are overwritten by the correct value. For a better definition of each option, see RFC 2132.

Option Number	Default Data Type	Can Specify?	Description/Use
0	None	No	The server pads the option field, if necessary.
1	Dotted quad	No	The net mask of the subnet from which the address was drawn.
2	32-bit integer	Yes	Specifies the offset of the client subnet, in seconds from Coordinated Universal Time (UTC).
3	One or more dotted quads	Yes	A list of the default gateways' IP addresses.
4	One or more dotted quads	Yes	A list of time server IP addresses.
5	One or more dotted quads	Yes	A list of name server IP addresses.
6	One or more dotted quads	Yes	A list of DNS IP addresses.
7	One or more dotted quads	Yes	A list of log server IP addresses.
8	One or more dotted quads	Yes	A list of cookie server IP addresses.
9	One or more dotted quads	Yes	A list of LPR server IP addresses.
10	One or more dotted quads	Yes	A list of Impress server IP addresses.
11	One or more dotted quads	Yes	A list of Resource Location server IP addresses.
12	An ASCII string	Yes	A host name for the client to use.
13	16-bit unsigned integer	Yes	The size of the bootfile.
14	An ASCII string	Yes	The path for Merit Dump file.
15	An ASCII string	Yes	The default DNS domain name.
16	An IP address	Yes	The address of the Swap server.
17	An ASCII string	Yes	The default root path.
18	An ASCII string	Yes	The path to extensions for the client.
19	Yes, No, True, False, 1, 0	Yes	Specify whether IP Forwarding should be turned on.
20	Yes, No, True, False, 1, 0	Yes	Specify whether non-local source routing should be used.
21	One or more pairs of dotted quads, in the form *DottedQuad*:*DottedQuad*	Yes	The filter policies for IP addresses.
22	16-bit unsigned integer	Yes	The maximum size to allow for datagram fragments.
23	8-bit unsigned integer	Yes	The IP time-to-live (TTL).
24	32-bit unsigned integer	Yes	The number of seconds to use in the Path MTU aging timeout.
25	List of one or more 16-bit unsigned integers	Yes	The path MTU Plateau table. Specifies a set of values that represent the MTU sizes to use when using Path MTU discovery.
26	16-bit unsigned integer	Yes	Specifies MTU size for the receiving interface.
27	Yes, No, True, False, 1, 0	Yes	Specifies whether all subnets are local.
28	An IP address (dotted quad)	Yes	Specifies broadcast address for the interface.
29	Yes, No, True, False, 1, 0	Yes	Specifies whether ICMP netmask discovery should be used.
30	Yes, No, True, False, 1, 0	Yes	Specifies whether client should become an ICMP netmask supplier.

Option Number	Default Data Type	Can Specify?	Description/Use
31	Yes, No, True, False, 1, 0	Yes	Specifies whether ICMP Router Discovery messages should be used.
32	IP address (dotted quad)	Yes	Specifies address to use for router solicitation.
33	One or more IP address pairs, in the form *DottedQuad*:*DottedQuad*	Yes	Each address pair represents a static route.
34	Yes/No, True/False, 1/0	Yes	Specifies whether trailer encapsulation should be used.
35	32-bit unsigned integer	Yes	ARP cache timeout value.
36	Yes/No, True/False, 1/0	Yes	Specifies whether Ethernet encapsulation should be used.
37	8-bit unsigned integer	Yes	The TCP time-to-live (TTL).
38	32-bit unsigned integer	Yes	The **TCP** keep alive interval.
39	Yes/No, True/False, 1/0	Yes	Specifies whether TCP keep alive should be used.
40	An ASCII string	Yes	The NIS default domain.
41	One or more dotted quads	Yes	Specifies the IP addresses of the NIS servers.
42	One or more dotted quads	Yes	Specifies the IP addresses of the NTP servers.
43	hex string of digits, in the form of hex "*digits*", hex "*digits*", or 0x*digits*	Yes, but really only specified with vendor container	Encapsulated option container for the vendor container.
44	One or more dotted quads	Yes	Specifies NetBIOS name server IP addresses.
45	One or more dotted quads	Yes	Specifies NetBIOS datagram distribution server IP addresses.
46	8-bit unsigned integer	Yes	Specifies NetBIOS Node Type.
47	hex string of digits, in form of hex "*digits*", hex "*digits*", or 0x*digits*	Yes	NetBIOS Scope.
48	One or more dotted quads	Yes	Specifies X Windows font server IP addresses.
49	One or more dotted quads	Yes	Specifies X Windows Display Manager.
50	None	No	Requested IP Address, used by client to indicate the address it wants.
51	32-bit unsigned integer	Yes	Lease time for the returned address. By default, the **DHCP** server uses the **leasetimedefault** keyword, but direct specification of option 51 overrides it.
52	None	No	Option overload. Client uses this to indicate the **sname** and **file** fields of the **BOOTP** packet may have options.
53	None	No	**DHCP** server or client uses this option to indicate the type of **DHCP** message.
54	None	No	**DHCP** server or client uses this option to indicate the server's address or the server to which the message is directed.
55	None	No	**DHCP** client uses this to indicate desired options.
56	An ASCII string	Yes	A string the **DHCP** server sends to the client. In general, this can be used by the **DHCP** server and client to indicate problems.
57	No	No	**DHCP** client uses this option to tell the **DHCP** server the maximum **DHCP** packet size the client can receive.
58	32-bit unsigned integer	Yes	Specifies the number of seconds until the client should send a renew packet.
59	32-bit unsigned integer	Yes	Specifies the number of seconds until the client should send a rebind packet.
60	None	No	**DHCP** client uses this option to indicate its vendor type. The **DHCP** server uses this field to match vendor containers.
61	None	No	**DHCP** client uses this to uniquely identify itself. The **DHCP** server uses this field to match client containers.
66	An ASCII string	Yes	Specifies the **TFTP** server name. This is a hostname and is used instead of the **siaddr** field if the client understands this option.
67	An ASCII string	Yes	Specifies the bootfile name. This can be used instead of the **bootfile** keyword, which places the file in the **filename** field of the packet.
68	One or more dotted quads, or NONE	Yes	Specifies addresses of home agents.
69	One or more dotted quads	Yes	Specifies default SMTP servers to use.

Option Number	Default Data Type	Can Specify?	Description/Use
70	One or more dotted quads	Yes	Specifies default POP3 servers to use.
71	One or more dotted quads	Yes	Specifies default NNTP servers to use.
72	One or more dotted quads	Yes	Specifies default WWW servers to use.
73	One or more dotted quads	Yes	Specifies default Finger servers to use.
74	One or more dotted quads	Yes	Specifies default IRC servers to use.
75	One or more dotted quads	Yes	Specifies default Street Talk servers to use.
76	One or more dotted quads	Yes	Specifies default Street Talk directory assistance servers to use.
77	An ASCII string	Yes	The user site class identifier. The **DHCP** server uses this field to match class containers.
78	Mandatory Byte, one or more dotted quads	Yes	The SLP directory Agent Option specifies a list of IP addresses for Directory Agents
79	Mandatory Byte, and ASCII string	Yes	The ASCII string is a scope list, which is a comma delimited list, that indicates the scopes that an SLP Agent is configured to use
81	An ASCII string plus other items	No	The DHCP client uses this option to define the policy the **DHCP** server should use with respect to DDNS.
85	One or more dotted quads	Yes	NDS server option specifies one or more NDS servers for the client to contact for access to the DNS database. Servers should be listed in order of preference.
86	An ASCII string	Yes	NDS tree name option specifies the name of the NDS tree the client will be contacting.
87	An ASCII string	Yes	NDS context option specifies the initial NDS context the client should use.
93	None	No	The DHCP client uses this option to define the client system architecture.
94	None	No	The DHCP client uses this option to define the client network interface identifier.

117	One or more 16-bit unsigned integers	Yes	Name Service Search Option gives the preferred order of integer option code for name services. For example:
			`Name Services value`
			`Domain Name Server Option 6`
			`NIS Option 41`

118	One dotted quads	No	Subnet Selection Option is an option sent by the client asking the dhcp server to allocate the IP address from the specified subnet.
255	None	No	DHCP server and client use this option to indicate the end of an option list.

Preboot execution environment vendor container suboption

When supporting a preboot execution environment (PXE) client, the **DHCP** server passes the following option to the BINLD server, which is used by BINLD to configure itself.

Opt Num	Default Data Type	Can Specify?	Description
7	one dotted quad	Yes	Multicast IP address. Boot server discovery multicast IP address.

The following example shows how this option can be used:

```
pxeservertype    proxy_on_dhcp_server

Vendor pxeserver
{
     option   7    9.3.4.68
}
```

In the above example, the **DHCP** server informs the client that the proxy server is running on the same machine but is listening on port 4011 for client requests. The vendor container is required here because

the BINLD server broadcasts an INFORM/REQUEST message on port 67 with option 60 set to "PXEServer." In response, the **DHCP** server sends the Multicast IP address on which the BINLD has to listen for PXEClient's request.

In the following example, the **dhcpsd** server either gives the bootfile name to the PXEClient or it directs the PXEClient to the BINLD server by sending suboptions. The **pxeboofile** keyword is used to create a list of boot files for a given client architecture and major and minor versions of the client system.

```
pxeservertype     dhcp_pxe_binld

subnet default
{
     vendor pxe
     {
         option 6 2    # Disable Multicast
         option 8 5 4 10.10.10.1 12.1.1.15 12.5.5.5 12.6.6.6\
                  2 2 10.1.1.10 9.3.4.5 1 1 10.5.5.9\
                  1 1 9.3.149.15\
                  4 0
         option 9 5 "WorkSpace On Demand" 2 "Intel"\
                  1 "Microsoft Windows NT" 4 "NEC ESMPRO"
         option 10 2 "Press F8 to View Menu"
     }
     vendor pxeserver
     {
         option 7 239.0.0.239
     }

}

subnet  9.3.149.0  255.255.255.0
{
     option 3   9.3.149.1
     option 6   9.3.149.15

     vendor pxe
     {
       option    6    4    # bootfile is present in the offer packet
       pxebootfile   1   2   1   os2.one
       pxebootfile   2   2   1   aix.one
     }
}
```

Each option line in PXE container is used by the server to tell the client what to do. “PXE vendor container suboptions” on page 288 describes the currently supported and known PXE suboptions.

DHCP server file syntax for general server operation

The **DHCP** file syntax for general server operation and the valid values for each field are defined here.

Note: Time Units (*time_units*) shown in the following table are optional and represent a modifier to the actual time. The default time unit is minutes. Valid values are seconds (1), minutes (60), hours (3600), days (86400), weeks (604800), months (2392000), and years (31536000). The number shown in parentheses is a multiplier applied to the specified value *n* to express the value in seconds.

Keyword	Form	Subcontainers	Default Value	Meaning
database	database *db_type*	Yes	None	The primary container that holds the definitions for the address pools, options, and client access statements. *db_type* is the name of a module that is loaded to process this part of the file. The only value currently available is **db_file**.
logging_info	logging_info	Yes	None	The primary logging container that defines the logging parameters.
logitem	logitem NONE	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed.
logitem	logitem SYSERR	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed.
logitem	logitem OBJERR	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed.
logitem	logitem PROTOCOL	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed.
logitem	logitem PROTERR	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed.
logitem	logitem WARN	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed.
logitem	logitem WARNING	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed.
logitem	logitem CONFIG	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed.
logitem	logitem EVENT	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed.
logitem	logitem PARSEERR	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed.
logitem	logitem ACTION	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed.
logitem	logitem ACNTING	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed.
logitem	logitem STAT	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed.
logitem	logitem TRACE	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed.
logitem	logitem RTRACE	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed.

Keyword	Form	Subcontainers	Default Value	Meaning
logitem	logitem START	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed.
numLogFiles	numLogFiles *n*	No	0	Specifies the number of log files to create. The log rotates when the first one fills. *n* is the number of files to create.
logFileSize	logFileSize *n*	No	0	Specifies the size of each log file in 1024-byte units.
logFileName	logFileName *path*	No	None	Specifies the path to the first log file. The original log file is named *filename* or *filename.extension*. When a file is rotated, it is renamed beginning with the base *filename*, then either appending a number or replacing the extension with a number. For example, if the original file name is `file`, the rotated file name becomes `file01`. If the original file name is `file.log`, it becomes `file.01`.
CharFlag	charflag yes	No	true	Not applicable to this operating system's **DHCP** server, but the OS/2 **DHCP** server uses it to produce debug windows.
CharFlag	charflag true	No	true	Not applicable to this operating system's **DHCP** server, but the OS/2 **DHCP** server uses it to produce debug windows.
CharFlag	charflag false	No	true	Not applicable to this operating system's **DHCP** server, but the OS/2 **DHCP** server uses it to produce debug windows.
CharFlag	charflag no	No	true	Not applicable to this operating system's **DHCP** server, but the OS/2 **DHCP** server uses it to produce debug windows.
StatisticSnapShot	StatisticSnapShot *n*	No	-1, never	Specifies how often statistics are written to the log file in seconds.
UsedIpAddressExpireInterval	UsedIpAddressExpireInterval *n time_units*	No	-1, never	Specifies how often addresses placed in the BAD state are recouped and retested for validity.
leaseExpireInterval	leaseExpireInterval *n time_units*	No	900 seconds	Specifies how often addresses in the BOUND state are checked to see if they have expired. If the address has expired, the state is moved to EXPIRED.

Keyword	Form	Subcontainers	Default Value	Meaning
reservedTime	reservedTime *n time_units*	No	-1, never	Specifies how long addresses should sit in RESERVED state before being recouped into the FREE state.
reservedTimeInterval	reservedTimeInterval *n time_units*	No	900 seconds	Specifies how often addresses in the RESERVE state are checked to see if they should be recouped into the FREE state.
saveInterval	saveInterval *n time_units*	No	3600 seconds	Specifies how often the **DHCP** server should force a save of the open databases. For heavily loaded servers, this should be 60 or 120 seconds.
clientpruneintv	clientpruneintv *n time_units*	No	3600 seconds	Specifies how often the **DHCP** server has the databases remove clients are not associated with any address (in the UNKNOWN state). This reduces the memory use of the **DHCP** server.
numprocessors	numprocessors *n*	No	10	Specifies the number of packet processors to create. Minimum of one.
userObject	userObject *obj_name*	Yes	None	Indicates that the server should load a user-defined shared object and call routines within this object through each interaction with **DHCP** clients. The object to be loaded is located in the `/usr/sbin` directory by the name `obj_name.dhcpo`. See the DHCP Server User-Defined Extension API for more information.
pxeservertype	pxeservertype *server_type*	No	dhcp_only	Indicates the type of **dhcpd** server that it is. *server_type* can be one of the following: **dhcp_pxe_binld** **DHCP** performs **dhcpsd**, **pxed**, and **bindl** functions. **proxy_on_dhcp_server** **DHCP** refers the PXE client to the proxy server port on the same machine. The default is `dhcp_only`, meaning the **dhcpsd** does not support PXE clients in default mode.

Keyword	Form	Subcontainers	Default Value	Meaning
supportsubnetselection	supportsubnetselection **global** supportsubnetselection **subnetlevel** supportsubnetselection **no**	No	None	Indicates whether the dhcp server will support the option 118 (subnet selection option) in the clients DISCOVER or REQUEST packet. **global**: all subnets in the configuration file will support option 118. **subnetlevel**: subnets that have been configured to support this option by keyword supportoption118 will support this option. **no**: does not support option 118.

DHCP server file syntax for db_file database

The file syntax for the `db_file` database has the following properties.

Note:

1. Time Units (*time_units*) shown in the following table are optional and represent a modifier to the actual time. The default time unit is minutes. Valid values are seconds (1), minutes (60), hours (3600), days (86400), weeks (604800), months (2392000), and years (31536000). The number shown in parentheses is a multiplier applied to the specified value *n* to express the value in seconds.
2. Items that are specified in one container can be overridden inside a subcontainer. For example, you could globally define **BOOTP** clients, but within a certain subnet allow **BOOTP** clients by specifying the supportBootp keyword in both containers.
3. The client, class, and vendor containers allow for regular expression support. For class and vendor, a quoted string with the first character after the quote being an exclamation point (!) indicates that the rest of the string should be treated as a regular expression. The client container allows for regular expressions on both the hwtype and the hwaddr fields. A single string is used to represent both fields with the following format:

   ```
   decimal_number-data
   ```

   If decimal_number is zero, then data is an ASCII string. If any other number, data is hex digits.

Keyword	Form	Subcontainers	Default Value	Meaning
subnet	subnet default	Yes	None	Specifies a subnet without an associated range. This subnet is used by the server only when responding to a client INFORM/REQUEST packet from the client and the client's address does not have another matching subnet container.

Keyword	Form	Subcontainers	Default Value	Meaning
subnet	subnet *subnet id netmask*	Yes	None	Specifies a subnet and a pool of addresses. All addresses are assumed to be in the pool unless a range is specified on the line or addresses are modified later in the container by a range or exclude statement. The optional range is a pair of IP addresses in dotted quad format separated by a dash. An optional label and priority can be specified. These are used by virtual subnets to identify and order the subnets in the virtual subnet. The label and priority are separated by a colon. These containers are only allowed at the global or database container level.
subnet	subnet *subnet id netmask range*	Yes	None	Specifies a subnet and a pool of addresses. All addresses are assumed to be in the pool unless a range is specified on the line or addresses are modified later in the container by a range or exclude statement. The optional range is a pair of IP addresses in dotted quad format separated by a dash. An optional label and priority can be specified. These are used by virtual subnets to identify and order the subnets in the virtual subnet. The label and priority are separated by a colon. These containers are only allowed at the global or database container level.
subnet	subnet *subnet id netmask label:priority*	Yes	None	Specifies a subnet and a pool of addresses. All addresses are assumed to be in the pool unless a range is specified on the line or addresses are modified later in the container by a range or exclude statement. The optional range is a pair of IP addresses in dotted quad format separated by a dash. An optional label and priority can be specified. These are used by virtual subnets to identify and order the subnets in the virtual subnet. The label and priority are separated by a colon. These containers are only allowed at the global or database container level.

Keyword	Form	Subcontainers	Default Value	Meaning
subnet	subnet *subnet id netmask range label:priority*	Yes	None	Specifies a subnet and a pool of addresses. All addresses are assumed to be in the pool unless a range is specified on the line or addresses are modified later in the container by a range or exclude statement. The optional range is a pair of IP addresses in dotted quad format separated by a dash. An optional label and priority can be specified. These are used by virtual subnets to identify and order the subnets in the virtual subnet. The label and priority are separated by a colon. These containers are only allowed at the global or database container level.
subnet	subnet *subnet id range*	Yes	None	Specifies a subnet that goes within a network container. It defines a range of addresses that is the whole subnet unless the optional range part is specified. The netmask associated with the subnet is taken from the surrounding network container. **Note:** This method is deprecated in favor of the other subnet forms.
option	option *number data* ...	No	None	Specifies an option to send to a client or, in the case of deny, an option to prevent from being sent to the client. The option `* deny` clause means all options not specified in the current container are not to be returned to the client. The option *number*deny only denies the specified option. *number* is an unsigned 8-bit integer. *data* is specific to the option (see above) or can be specified as a quoted string (indicating ASCII text) or 0x*hexdigits* or hex"*hexdigits*" or hex "*hexdigits*". If the option is in a vendor container, the option will be encapsulated with other options in an option 43.

Keyword	Form	Subcontainers	Default Value	Meaning
option	option *number*deny	No	None	Specifies an option to send to a client or, in the case of deny, an option to prevent from being sent to the client. The option `* deny` clause means all options not specified in the current container are not to be returned to the client. The option *number*deny only denies the specified option. *number* is an unsigned 8-bit integer. *data* is specific to the option (see above) or can be specified as a quoted string (indicating ASCII text) or 0x*hexdigits* or hex"*hexdigits*" or hex "*hexdigits*". If the option is in a vendor container, the option will be encapsulated with other options in an option 43.
option	option * deny	No	None	Specifies an option to send to a client or, in the case of deny, an option to prevent from being sent to the client. The option `* deny` clause means all options not specified in the current container are not to be returned to the client. The option *number*deny only denies the specified option. *number* is an unsigned 8-bit integer. *data* is specific to the option (see above) or can be specified as a quoted string (indicating ASCII text) or 0x*hexdigits* or hex"*hexdigits*" or hex "*hexdigits*". If the option is in a vendor container, the option will be encapsulated with other options in an option 43.
exclude	exclude *an IP address*	No	None	Modifies the range on the container in which the exclude statement is in. The exclude statement is not valid in the global or database container levels. The exclude statement removes the specified address or range from the current range on the container. The exclude statement allows you to create noncontiguous ranges for subnets or other containers.

Keyword	Form	Subcontainers	Default Value	Meaning
exclude	exclude *dotted_quad-dotted_quad*	No	None	Modifies the range on the container in which the exclude statement is in. The exclude statement is not valid in the global or database container levels. The exclude statement removes the specified address or range from the current range on the container. The exclude statement allows you to create noncontiguous ranges for subnets or other containers.
range	range *IP_address*	No	None	Modifies the range on the container in which the range statement is in. The range statement is not valid in the global or database container levels. If the range is the first in the container that does not specify a range on the container definition line, then the range for the container becomes the range specified by the range statement. Any range statement after the first range or all range statements for a containers that specifies ranges in its definition are added to the current range. With the range statement, a single address or set of addresses can be added to the range. The range must fit inside the subnet container definition.
range	range *dotted_quad-dotted_quad*	No	None	Modifies the range on the container in which the range statement is in. The range statement is not valid in the global or database container levels. If the range is the first in the container that does not specify a range on the container definition line, then the range for the container becomes the range specified by the range statement. Any range statement after the first range or all range statements for a containers that specifies ranges in its definition are added to the current range. With the range statement, a single address or set of addresses can be added to the range. The range must fit inside the subnet container definition.

Keyword	Form	Subcontainers	Default Value	Meaning
client	client *hwtype hwaddr* NONE	Yes	None	Specifies a client container that denies the client specified by the *hwaddr* and *hwtype* from getting an address. If *hwtype* is 0, then *hwaddr* is an ASCII string. Otherwise, *hwtype* is the hardware type for the client and *hwaddr* is the hardware address of the client. If the *hwaddr* is a string, then quotes are accepted around the string. If the *hwaddr* is a hexstring, then the address may be specified by 0x*hexdigits* or hex *digits*. *range* allows the client specified by the *hwaddr* and *hwtype* to get an address in the *range*. Must be regular expressions to match multiple clients.
client	client *hwtype hwaddr* ANY	Yes	None	Specifies a client container that denies the client specified by the *hwaddr* and *hwtype* from getting an address. If *hwtype* is 0, then *hwaddr* is an ASCII string. Otherwise, *hwtype* is the hardware type for the client and *hwaddr* is the hardware address of the client. If the *hwaddr* is a string, then quotes are accepted around the string. If the *hwaddr* is a hexstring, then the address may be specified by 0x*hexdigits* or hex *digits*. *range* allows the client specified by the *hwaddr* and *hwtype* to get an address in the *range*. Must be regular expressions to match multiple clients.

Keyword	Form	Subcontainers	Default Value	Meaning
client	client *hwtype hwaddr dotted_quad*	Yes	None	Specifies a client container that denies the client specified by the *hwaddr* and *hwtype* from getting an address. If *hwtype* is 0, then *hwaddr* is an ASCII string. Otherwise, *hwtype* is the hardware type for the client and *hwaddr* is the hardware address of the client. If the *hwaddr* is a string, then quotes are accepted around the string. If the *hwaddr* is a hexstring, then the address may be specified by 0x*hexdigits* or hex *digits*. *range* allows the client specified by the *hwaddr* and *hwtype* to get an address in the *range*. Must be regular expressions to match multiple clients.
client	client *hwtype hwaddr range*	Yes	None	Specifies a client container that denies the client specified by the *hwaddr* and *hwtype* from getting an address. If *hwtype* is 0, then *hwaddr* is an ASCII string. Otherwise, *hwtype* is the hardware type for the client and *hwaddr* is the hardware address of the client. If the *hwaddr* is a string, then quotes are accepted around the string. If the *hwaddr* is a hexstring, then the address may be specified by 0x*hexdigits* or hex *digits*. *range* allows the client specified by the *hwaddr* and *hwtype* to get an address in the *range*. Must be regular expressions to match multiple clients.
class	class *string*	Yes	None	Specifies a class container with name *string*. String can be quoted or not. If quoted, the quotes are removed before comparison. Quotes are required for strings with spaces or tabs. This container is valid at any level. A range can be supplied to indicate a set of addresses to hand out to a client with this class. The range is either a single dotted quad IP address or two dotted quad IP addresses separated by a dash.

Keyword	Form	Subcontainers	Default Value	Meaning
class	class *string range*	Yes	None	Specifies a class container with name *string*. String can be quoted or not. If quoted, the quotes are removed before comparison. Quotes are required for strings with spaces or tabs. This container is valid at any level. A range can be supplied to indicate a set of addresses to hand out to a client with this class. The range is either a single dotted quad IP address or two dotted quad IP addresses separated by a dash.
network	network *network id netmask*	Yes	None	Specifies a container to be matched against any arbitrary incoming option specified by the client. *number* specifies the option number. *option_data* specifies the key to match for this container to be selected during address and option selection for the client. *option_data* is specified in expected form — quoted string, IP address, integer value — for well known options, or it can be optionally specified as a hexadecimal string of bytes if preceded by the characters `0x`. For options that are not well known to the server, a hexadecimal string of bytes can be specified in the same fashion. Additionally, the *option_data* can indicate a regular expression to be compared against the string representation of the client's option data. Regular expressions are specified in a quoted string beginning `"!` (double quote followed by an exclamation mark). The string form of options not well known to the server will be a hexadecimal string of bytes NOT preceded with the characters `0x`

Keyword	Form	Subcontainers	Default Value	Meaning
network	network *network id*	Yes	None	Specifies a container to be matched against any arbitrary incoming option specified by the client. *number* specifies the option number. *option_data* specifies the key to match for this container to be selected during address and option selection for the client. *option_data* is specified in expected form — quoted string, IP address, integer value — for well known options, or it can be optionally specified as a hexadecimal string of bytes if preceded by the characters `0x`. For options that are not well known to the server, a hexadecimal string of bytes can be specified in the same fashion. Additionally, the *option_data* can indicate a regular expression to be compared against the string representation of the client's option data. Regular expressions are specified in a quoted string beginning `"!` (double quote followed by an exclamation mark). The string form of options not well known to the server will be a hexadecimal string of bytes NOT preceded with the characters `0x`

Keyword	Form	Subcontainers	Default Value	Meaning
network	network *network id range*	Yes	None	Specifies a container to be matched against any arbitrary incoming option specified by the client. *number* specifies the option number. *option_data* specifies the key to match for this container to be selected during address and option selection for the client. *option_data* is specified in expected form — quoted string, IP address, integer value — for well known options, or it can be optionally specified as a hexadecimal string of bytes if preceded by the characters `0x`. For options that are not well known to the server, a hexadecimal string of bytes can be specified in the same fashion. Additionally, the *option_data* can indicate a regular expression to be compared against the string representation of the client's option data. Regular expressions are specified in a quoted string beginning `"!` (double quote followed by an exclamation mark). The string form of options not well known to the server will be a hexadecimal string of bytes NOT preceded with the characters `0x`

Keyword	Form	Subcontainers	Default Value	Meaning
vendor	vendor *vendor_id*	Yes	None	Specifies a container to be matched against any arbitrary incoming option specified by the client. *number* specifies the option number. *option_data* specifies the key to match for this container to be selected during address and option selection for the client. *option_data* is specified in expected form — quoted string, IP address, integer value — for well known options, or it can be optionally specified as a hexadecimal string of bytes if preceded by the characters `0x`. For options that are not well known to the server, a hexadecimal string of bytes can be specified in the same fashion. Additionally, the *option_data* can indicate a regular expression to be compared against the string representation of the client's option data. Regular expressions are specified in a quoted string beginning `"!` (double quote followed by an exclamation mark). The string form of options not well known to the server will be a hexadecimal string of bytes NOT preceded with the characters `0x`

Keyword	Form	Subcontainers	Default Value	Meaning
vendor	vendor *vendor_id* hex""	Yes	None	Specifies a container to be matched against any arbitrary incoming option specified by the client. *number* specifies the option number. *option_data* specifies the key to match for this container to be selected during address and option selection for the client. *option_data* is specified in expected form — quoted string, IP address, integer value — for well known options, or it can be optionally specified as a hexadecimal string of bytes if preceded by the characters `0x`. For options that are not well known to the server, a hexadecimal string of bytes can be specified in the same fashion. Additionally, the *option_data* can indicate a regular expression to be compared against the string representation of the client's option data. Regular expressions are specified in a quoted string beginning `"!` (double quote followed by an exclamation mark). The string form of options not well known to the server will be a hexadecimal string of bytes NOT preceded with the characters `0x`

Keyword	Form	Subcontainers	Default Value	Meaning
vendor	vendor *vendor_id* hex ""	Yes	None	Specifies a container to be matched against any arbitrary incoming option specified by the client. *number* specifies the option number. *option_data* specifies the key to match for this container to be selected during address and option selection for the client. *option_data* is specified in expected form — quoted string, IP address, integer value — for well known options, or it can be optionally specified as a hexadecimal string of bytes if preceded by the characters `0x`. For options that are not well known to the server, a hexadecimal string of bytes can be specified in the same fashion. Additionally, the *option_data* can indicate a regular expression to be compared against the string representation of the client's option data. Regular expressions are specified in a quoted string beginning `"!` (double quote followed by an exclamation mark). The string form of options not well known to the server will be a hexadecimal string of bytes NOT preceded with the characters `0x`

Keyword	Form	Subcontainers	Default Value	Meaning
vendor	vendor *vendor_id* 0xdata	Yes	None	Specifies a container to be matched against any arbitrary incoming option specified by the client. *number* specifies the option number. *option_data* specifies the key to match for this container to be selected during address and option selection for the client. *option_data* is specified in expected form — quoted string, IP address, integer value — for well known options, or it can be optionally specified as a hexadecimal string of bytes if preceded by the characters `0x`. For options that are not well known to the server, a hexadecimal string of bytes can be specified in the same fashion. Additionally, the *option_data* can indicate a regular expression to be compared against the string representation of the client's option data. Regular expressions are specified in a quoted string beginning `"!` (double quote followed by an exclamation mark). The string form of options not well known to the server will be a hexadecimal string of bytes NOT preceded with the characters `0x`

Keyword	Form	Subcontainers	Default Value	Meaning
vendor	vendor *vendor_id* ""	Yes	None	Specifies a container to be matched against any arbitrary incoming option specified by the client. *number* specifies the option number. *option_data* specifies the key to match for this container to be selected during address and option selection for the client. *option_data* is specified in expected form — quoted string, IP address, integer value — for well known options, or it can be optionally specified as a hexadecimal string of bytes if preceded by the characters `0x`. For options that are not well known to the server, a hexadecimal string of bytes can be specified in the same fashion. Additionally, the *option_data* can indicate a regular expression to be compared against the string representation of the client's option data. Regular expressions are specified in a quoted string beginning `"!` (double quote followed by an exclamation mark). The string form of options not well known to the server will be a hexadecimal string of bytes NOT preceded with the characters `0x`

Keyword	Form	Subcontainers	Default Value	Meaning
vendor	vendor *vendor_id range*	Yes	None	Specifies a container to be matched against any arbitrary incoming option specified by the client. *number* specifies the option number. *option_data* specifies the key to match for this container to be selected during address and option selection for the client. *option_data* is specified in expected form — quoted string, IP address, integer value — for well known options, or it can be optionally specified as a hexadecimal string of bytes if preceded by the characters `0x`. For options that are not well known to the server, a hexadecimal string of bytes can be specified in the same fashion. Additionally, the *option_data* can indicate a regular expression to be compared against the string representation of the client's option data. Regular expressions are specified in a quoted string beginning `"!` (double quote followed by an exclamation mark). The string form of options not well known to the server will be a hexadecimal string of bytes NOT preceded with the characters `0x`

Keyword	Form	Subcontainers	Default Value	Meaning
vendor	vendor *vendor_id range* hex""	Yes	None	Specifies a container to be matched against any arbitrary incoming option specified by the client. *number* specifies the option number. *option_data* specifies the key to match for this container to be selected during address and option selection for the client. *option_data* is specified in expected form — quoted string, IP address, integer value — for well known options, or it can be optionally specified as a hexadecimal string of bytes if preceded by the characters `0x`. For options that are not well known to the server, a hexadecimal string of bytes can be specified in the same fashion. Additionally, the *option_data* can indicate a regular expression to be compared against the string representation of the client's option data. Regular expressions are specified in a quoted string beginning `"!` (double quote followed by an exclamation mark). The string form of options not well known to the server will be a hexadecimal string of bytes NOT preceded with the characters `0x`

Keyword	Form	Subcontainers	Default Value	Meaning
vendor	vendor *vendor_id range* hex ""	Yes	None	Specifies a container to be matched against any arbitrary incoming option specified by the client. *number* specifies the option number. *option_data* specifies the key to match for this container to be selected during address and option selection for the client. *option_data* is specified in expected form — quoted string, IP address, integer value — for well known options, or it can be optionally specified as a hexadecimal string of bytes if preceded by the characters `0x`. For options that are not well known to the server, a hexadecimal string of bytes can be specified in the same fashion. Additionally, the *option_data* can indicate a regular expression to be compared against the string representation of the client's option data. Regular expressions are specified in a quoted string beginning `"!` (double quote followed by an exclamation mark). The string form of options not well known to the server will be a hexadecimal string of bytes NOT preceded with the characters `0x`

Keyword	Form	Subcontainers	Default Value	Meaning
vendor	vendor *vendor_id range* 0xdata	Yes	None	Specifies a container to be matched against any arbitrary incoming option specified by the client. *number* specifies the option number. *option_data* specifies the key to match for this container to be selected during address and option selection for the client. *option_data* is specified in expected form — quoted string, IP address, integer value — for well known options, or it can be optionally specified as a hexadecimal string of bytes if preceded by the characters `0x`. For options that are not well known to the server, a hexadecimal string of bytes can be specified in the same fashion. Additionally, the *option_data* can indicate a regular expression to be compared against the string representation of the client's option data. Regular expressions are specified in a quoted string beginning `"!` (double quote followed by an exclamation mark). The string form of options not well known to the server will be a hexadecimal string of bytes NOT preceded with the characters `0x`

Keyword	Form	Subcontainers	Default Value	Meaning
vendor	vendor *vendor_id range* ""	Yes	None	Specifies a container to be matched against any arbitrary incoming option specified by the client. *number* specifies the option number. *option_data* specifies the key to match for this container to be selected during address and option selection for the client. *option_data* is specified in expected form — quoted string, IP address, integer value — for well known options, or it can be optionally specified as a hexadecimal string of bytes if preceded by the characters `0x`. For options that are not well known to the server, a hexadecimal string of bytes can be specified in the same fashion. Additionally, the *option_data* can indicate a regular expression to be compared against the string representation of the client's option data. Regular expressions are specified in a quoted string beginning `"!` (double quote followed by an exclamation mark). The string form of options not well known to the server will be a hexadecimal string of bytes NOT preceded with the characters `0x`

Keyword	Form	Subcontainers	Default Value	Meaning
inoption	inoption *number option_data*	Yes	None	Specifies a container to be matched against any arbitrary incoming option specified by the client. *number* specifies the option number. *option_data* specifies the key to match for this container to be selected during address and option selection for the client. *option_data* is specified in expected form — quoted string, IP address, integer value — for well known options, or it can be optionally specified as a hexadecimal string of bytes if preceded by the characters `0x`. For options that are not well known to the server, a hexadecimal string of bytes can be specified in the same fashion. Additionally, the *option_data* can indicate a regular expression to be compared against the string representation of the client's option data. Regular expressions are specified in a quoted string beginning `"!` (double quote followed by an exclamation mark). The string form of options not well known to the server will be a hexadecimal string of bytes NOT preceded with the characters `0x`

Keyword	Form	Subcontainers	Default Value	Meaning
inoption	inoption *number option_data range*	Yes	None	Specifies a container to be matched against any arbitrary incoming option specified by the client. *number* specifies the option number. *option_data* specifies the key to match for this container to be selected during address and option selection for the client. *option_data* is specified in expected form — quoted string, IP address, integer value — for well known options, or it can be optionally specified as a hexadecimal string of bytes if preceded by the characters `0x`. For options that are not well known to the server, a hexadecimal string of bytes can be specified in the same fashion. Additionally, the *option_data* can indicate a regular expression to be compared against the string representation of the client's option data. Regular expressions are specified in a quoted string beginning `"!` (double quote followed by an exclamation mark). The string form of options not well known to the server will be a hexadecimal string of bytes NOT preceded with the characters `0x`
virtual	virtual fill *id id* ...	No	None	Specifies a virtual subnet with a policy. `fill` means use all addresses in the container before going to the next container. `rotate` means select an address from the next pool in the list on each request. `sfill` and `srotate` are the same as `fill` and `rotate`, but a search is done to see if the client matches containers, vendors, or classes in the subnet. If a match is found that can supply an address, the address is taken from that container instead of following the policy. There can be as many IDs as needed. *id* is either the subnet ID from the subnet definition or the label from the subnet definition. The label is required if there are multiple subnets with the same subnet id.

Keyword	Form	Subcontainers	Default Value	Meaning
virtual	virtual sfill *id id* ...	No	None	Specifies a virtual subnet with a policy. `fill` means use all addresses in the container before going to the next container. `rotate` means select an address from the next pool in the list on each request. `sfill` and `srotate` are the same as `fill` and `rotate`, but a search is done to see if the client matches containers, vendors, or classes in the subnet. If a match is found that can supply an address, the address is taken from that container instead of following the policy. There can be as many IDs as needed. *id* is either the subnet ID from the subnet definition or the label from the subnet definition. The label is required if there are multiple subnets with the same subnet id.
virtual	virtual rotate *id id* ...	No	None	Specifies a virtual subnet with a policy. `fill` means use all addresses in the container before going to the next container. `rotate` means select an address from the next pool in the list on each request. `sfill` and `srotate` are the same as `fill` and `rotate`, but a search is done to see if the client matches containers, vendors, or classes in the subnet. If a match is found that can supply an address, the address is taken from that container instead of following the policy. There can be as many IDs as needed. *id* is either the subnet ID from the subnet definition or the label from the subnet definition. The label is required if there are multiple subnets with the same subnet id.

Keyword	Form	Subcontainers	Default Value	Meaning
virtual	virtual srotate *id id* ...	No	None	Specifies a virtual subnet with a policy. `fill` means use all addresses in the container before going to the next container. `rotate` means select an address from the next pool in the list on each request. `sfill` and `srotate` are the same as `fill` and `rotate`, but a search is done to see if the client matches containers, vendors, or classes in the subnet. If a match is found that can supply an address, the address is taken from that container instead of following the policy. There can be as many IDs as needed. *id* is either the subnet ID from the subnet definition or the label from the subnet definition. The label is required if there are multiple subnets with the same subnet id.
inorder:	inorder: *id id* ...	No	None	Specifies a virtual subnet with a policy of fill, which means use all addresses in the container before going to the next container. There can be as many IDs as needed. *id* is either the subnet ID from the subnet definition or the label from the subnet definition. The label is required if there are multiple subnets with the same subnet ID.
balance:	balance: *id id* ...	No	None	Specifies a virtual subnet with a policy of rotate, which means use the next address in the next container. There can be as many IDs as needed. *id* is either the subnet ID from the subnet definition or the label from the subnet definition. The label is required if there are multiple subnets with the same subnet ID.
supportBootp	supportBootp true	No	Yes	Specifies whether the current container and all below it (until overridden) should support **BOOTP** clients.
supportBootp	supportBootp 1	No	Yes	Specifies whether the current container and all below it (until overridden) should support **BOOTP** clients.

Keyword	Form	Subcontainers	Default Value	Meaning
supportBootp	supportBootp yes	No	Yes	Specifies whether the current container and all below it (until overridden) should support **BOOTP** clients.
supportBootp	supportBootp false	No	Yes	Specifies whether the current container and all below it (until overridden) should support **BOOTP** clients.
supportBootp	supportBootp 0	No	Yes	Specifies whether the current container and all below it (until overridden) should support **BOOTP** clients.
supportBootp	supportBootp no	No	Yes	Specifies whether the current container and all below it (until overridden) should support **BOOTP** clients.
supportBootp				Specifies whether the current container and all below it (until overridden) should support **BOOTP** clients.
supportUnlistedclients	supportUnlistedclients BOTH	No	Both	Specifies whether the current container and all below it (until overridden) should support unlisted clients. The value indicates whether all clients should be allowed access without specific client statements, **DHCP** clients only, **BOOTP** clients only, or no one. **Note:** The true and false values are supported for compatibility with previous versions and are deprecated. The true value corresponds to BOTH and the false value corresponds to NONE.
supportUnlistedclients	supportUnlistedclients DHCP	No	Both	Specifies whether the current container and all below it (until overridden) should support unlisted clients. The value indicates whether all clients should be allowed access without specific client statements, **DHCP** clients only, **BOOTP** clients only, or no one. **Note:** The true and false values are supported for compatibility with previous versions and are deprecated. The true value corresponds to BOTH and the false value corresponds to NONE.

Keyword	Form	Subcontainers	Default Value	Meaning
supportUnlistedclients	supportUnlistedclients BOOTP	No	Both	Specifies whether the current container and all below it (until overridden) should support unlisted clients. The value indicates whether all clients should be allowed access without specific client statements, **DHCP** clients only, **BOOTP** clients only, or no one. **Note:** The true and false values are supported for compatibility with previous versions and are deprecated. The true value corresponds to BOTH and the false value corresponds to NONE.
supportUnlistedclients	supportUnlistedclients NONE	No	Both	Specifies whether the current container and all below it (until overridden) should support unlisted clients. The value indicates whether all clients should be allowed access without specific client statements, **DHCP** clients only, **BOOTP** clients only, or no one. **Note:** The true and false values are supported for compatibility with previous versions and are deprecated. The true value corresponds to BOTH and the false value corresponds to NONE.
supportUnlistedclients	supportUnlistedclients true	No	Both	Specifies whether the current container and all below it (until overridden) should support unlisted clients. The value indicates whether all clients should be allowed access without specific client statements, **DHCP** clients only, **BOOTP** clients only, or no one. **Note:** The true and false values are supported for compatibility with previous versions and are deprecated. The true value corresponds to BOTH and the false value corresponds to NONE.

Keyword	Form	Subcontainers	Default Value	Meaning
supportUnlistedclients	supportUnlistedclients yes	No	Both	Specifies whether the current container and all below it (until overridden) should support unlisted clients. The value indicates whether all clients should be allowed access without specific client statements, **DHCP** clients only, **BOOTP** clients only, or no one. **Note:** The true and false values are supported for compatibility with previous versions and are deprecated. The true value corresponds to BOTH and the false value corresponds to NONE.
supportUnlistedclients	supportUnlistedclients 1	No	Both	Specifies whether the current container and all below it (until overridden) should support unlisted clients. The value indicates whether all clients should be allowed access without specific client statements, **DHCP** clients only, **BOOTP** clients only, or no one. **Note:** The true and false values are supported for compatibility with previous versions and are deprecated. The true value corresponds to BOTH and the false value corresponds to NONE.
supportUnlistedclients	supportUnlistedclients false	No	Both	Specifies whether the current container and all below it (until overridden) should support unlisted clients. The value indicates whether all clients should be allowed access without specific client statements, **DHCP** clients only, **BOOTP** clients only, or no one. **Note:** The true and false values are supported for compatibility with previous versions and are deprecated. The true value corresponds to BOTH and the false value corresponds to NONE.

Keyword	Form	Subcontainers	Default Value	Meaning
supportUnlistedclients	supportUnlistedclients no	No	Both	Specifies whether the current container and all below it (until overridden) should support unlisted clients. The value indicates whether all clients should be allowed access without specific client statements, **DHCP** clients only, **BOOTP** clients only, or no one. **Note:** The true and false values are supported for compatibility with previous versions and are deprecated. The true value corresponds to BOTH and the false value corresponds to NONE.
supportUnlistedclients	supportUnlistedclients 0	No	Both	Specifies whether the current container and all below it (until overridden) should support unlisted clients. The value indicates whether all clients should be allowed access without specific client statements, **DHCP** clients only, **BOOTP** clients only, or no one. **Note:** The true and false values are supported for compatibility with previous versions and are deprecated. The true value corresponds to BOTH and the false value corresponds to NONE.
addressrecorddb	addressrecrddb *path*	No	None	If specified, it works like the **backupfile** keyword. Only valid in the global or database container level. **Note:** This method is deprecated.
backupfile	backupfile *path*	No	/etc/db_file.crbk	Specifies the file to use for database backups. Only valid in the global or database container level.
checkpointfile	checkpointfile *path*	No	/etc/db_file.chkpt	Specifies the database checkpoint files. The first checkpoint file is the *path*. The second checkpoint file is *path* with the last character replaced with a 2. So, the checkpoint file should not end in 2. Only valid in the global or database container level.
clientrecorddb	clientrecorddb *path*	No	/etc/db_file.cr	Specifies the database save file. The file contains all the client records the **DHCP** server has serviced. Only valid in the global or database container level.

Keyword	Form	Subcontainers	Default Value	Meaning
bootstrapserver	bootstrapserver *IP address*	No	None	Specifies the server clients should use from which to **TFTP** files after receiving **BOOTP** or **DHCP** packets. This value fills in the **siaddr** field in the packet. This is valid at any container level.
giaddrfield	giaddrfield *IP address*	No	None	Specifies the giaddrfield for response packets. **Note:** This specification is illegal in the **BOOTP** and **DHCP** protocols, but some clients require the **giaddr** field to be the default gateway for the network. Because of this potential conflict, giaddrfield should only be used within a client container, although it can work at any level.
pingTime	pingTime *n time_unit*	No	3 seconds	Specifies the amount of time to wait for a ping response before handing out an address. The default time unit is hundredths of a second. The time unit value is defined in the note preceding this table. This is valid at any container level. The *time_unit* parameter is optional.
bootptime	bootptime *n time_unit*	No	-1, infinite	Specifies the amount of time to lease an address to a **BOOTP** client. The default is -1, which means infinite. The normal time unit values are available. The *time unit* parameter is optional. This is valid at any container level.
AllRoutesBroadcast	allroutesbroadcast no	No	0	Specifies whether responses should be broadcast to all routes, if a broadcast response is required. This is valid at any container level. This is ignored by the operating system's **DHCP** servers, because the actual MAC address of the client, including RIFs, are stored for the return packet. This is valid at any container level.

Keyword	Form	Subcontainers	Default Value	Meaning
AllRoutesBroadcast	allroutesbroadcast false	No	0	Specifies whether responses should be broadcast to all routes, if a broadcast response is required. This is valid at any container level. This is ignored by the operating system's **DHCP** servers, because the actual MAC address of the client, including RIFs, are stored for the return packet. This is valid at any container level.
AllRoutesBroadcast	allroutesbroadcast 0	No	0	Specifies whether responses should be broadcast to all routes, if a broadcast response is required. This is valid at any container level. This is ignored by the operating system's **DHCP** servers, because the actual MAC address of the client, including RIFs, are stored for the return packet. This is valid at any container level.
AllRoutesBroadcast	allroutesbroadcast yes	No	0	Specifies whether responses should be broadcast to all routes, if a broadcast response is required. This is valid at any container level. This is ignored by the operating system's **DHCP** servers, because the actual MAC address of the client, including RIFs, are stored for the return packet. This is valid at any container level.
AllRoutesBroadcast	allroutesbroadcast true	No	0	Specifies whether responses should be broadcast to all routes, if a broadcast response is required. This is valid at any container level. This is ignored by the operating system's **DHCP** servers, because the actual MAC address of the client, including RIFs, are stored for the return packet. This is valid at any container level.
AllRoutesBroadcast	allroutesbroadcast 1	No	0	Specifies whether responses should be broadcast to all routes, if a broadcast response is required. This is valid at any container level. This is ignored by the operating system's **DHCP** servers, because the actual MAC address of the client, including RIFs, are stored for the return packet. This is valid at any container level.

Keyword	Form	Subcontainers	Default Value	Meaning
addressassigned	addressassigned "*string*"	No	None	Specifies a quoted string to execute when an address is assigned to a client. The string should have two %s. The first %s is the client id in the form *type-string*. The second %s is an IP address in dotted quad format. This is valid at any container level.
addressreleased	addressreleased "*string*"	No	None	Specifies a quoted string to execute when an address is released by a client. The string should have one %s. The %s is the IP address being released in dotted quad format. This is valid at any container level.
appenddomain	appenddomain 0	No	No	Specifies whether to append the defined option 15 domain name to the client-suggested host name in the event that the client does not suggest a domain name as well. This is valid at any container level.
appenddomain	appenddomain no	No	No	Specifies whether to append the defined option 15 domain name to the client-suggested host name in the event that the client does not suggest a domain name as well. This is valid at any container level.
appenddomain	appenddomain false	No	No	Specifies whether to append the defined option 15 domain name to the client-suggested host name in the event that the client does not suggest a domain name as well. This is valid at any container level.
appenddomain	appenddomain 1	No	No	Specifies whether to append the defined option 15 domain name to the client-suggested host name in the event that the client does not suggest a domain name as well. This is valid at any container level.
appenddomain	appenddomain yes	No	No	Specifies whether to append the defined option 15 domain name to the client-suggested host name in the event that the client does not suggest a domain name as well. This is valid at any container level.

Keyword	Form	Subcontainers	Default Value	Meaning
appenddomain	appenddomain true	No	No	Specifies whether to append the defined option 15 domain name to the client-suggested host name in the event that the client does not suggest a domain name as well. This is valid at any container level.
canonical	canonical 0	No	0	Specifies that the client id is in canonical format. This is valid only in the client container.
canonical	canonical no	No	0	Specifies that the client id is in canonical format. This is valid only in the client container.
canonical	canonical false	No	0	Specifies that the client id is in canonical format. This is valid only in the client container.
canonical	canonical 1	No	0	Specifies that the client id is in canonical format. This is valid only in the client container.
canonical	canonical yes	No	0	Specifies that the client id is in canonical format. This is valid only in the client container.
canonical	canonical true	No	0	Specifies that the client id is in canonical format. This is valid only in the client container.
leaseTimeDefault	leaseTimeDefault *n time_unit*	No	86400 seconds	Specifies the default lease time for clients. This is valid at any container level. The *time_unit* parameter is optional.
proxyarec	proxyarec never	No	usedhcpddnsplus	Specifies what options and methods should be used for A record updates in the DNS. `never` means never update the A record. `usedhcpddns` means use option 81 if the client specifies it. `usedhcpddnsplus` means use option 81 or option 12 and 15, if specified. `always` means do the A record update for all clients. `XXXXprotected` modifies the **nsupdate** command to make sure the client is allowed. `standard` is a synonym for `always`. `protected` is a synonym for `alwaysprotected`. This is valid at any container level.

Keyword	Form	Subcontainers	Default Value	Meaning
proxyarec	proxyarec usedhcpddns	No	usedhcpddnsplus	Specifies what options and methods should be used for A record updates in the DNS. `never` means never update the A record. `usedhcpddns` means use option 81 if the client specifies it. `usedhcpddnsplus` means use option 81 or option 12 and 15, if specified. `always` means do the A record update for all clients. `XXXXprotected` modifies the **nsupdate** command to make sure the client is allowed. `standard` is a synonym for `always`. `protected` is a synonym for `alwaysprotected`. This is valid at any container level.
proxyarec	proxyarec usedhcpddnsplus	No	usedhcpddnsplus	Specifies what options and methods should be used for A record updates in the DNS. `never` means never update the A record. `usedhcpddns` means use option 81 if the client specifies it. `usedhcpddnsplus` means use option 81 or option 12 and 15, if specified. `always` means do the A record update for all clients. `XXXXprotected` modifies the **nsupdate** command to make sure the client is allowed. `standard` is a synonym for `always`. `protected` is a synonym for `alwaysprotected`. This is valid at any container level.
proxyarec	proxyarec always	No	usedhcpddnsplus	Specifies what options and methods should be used for A record updates in the DNS. `never` means never update the A record. `usedhcpddns` means use option 81 if the client specifies it. `usedhcpddnsplus` means use option 81 or option 12 and 15, if specified. `always` means do the A record update for all clients. `XXXXprotected` modifies the **nsupdate** command to make sure the client is allowed. `standard` is a synonym for `always`. `protected` is a synonym for `alwaysprotected`. This is valid at any container level.

Keyword	Form	Subcontainers	Default Value	Meaning
proxyarec	proxyarec usedhcpddnsprotected	No	usedhcpddnsplus	Specifies what options and methods should be used for A record updates in the DNS. `never` means never update the A record. `usedhcpddns` means use option 81 if the client specifies it. `usedhcpddnsplus` means use option 81 or option 12 and 15, if specified. `always` means do the A record update for all clients. `XXXXprotected` modifies the **nsupdate** command to make sure the client is allowed. `standard` is a synonym for `always`. `protected` is a synonym for `alwaysprotected`. This is valid at any container level.
proxyarec	proxyarec usedhcpddnsplusprotected	No	usedhcpddnsplus	Specifies what options and methods should be used for A record updates in the DNS. `never` means never update the A record. `usedhcpddns` means use option 81 if the client specifies it. `usedhcpddnsplus` means use option 81 or option 12 and 15, if specified. `always` means do the A record update for all clients. `XXXXprotected` modifies the **nsupdate** command to make sure the client is allowed. `standard` is a synonym for `always`. `protected` is a synonym for `alwaysprotected`. This is valid at any container level.
proxyarec	proxyarec alwaysprotected	No	usedhcpddnsplus	Specifies what options and methods should be used for A record updates in the DNS. `never` means never update the A record. `usedhcpddns` means use option 81 if the client specifies it. `usedhcpddnsplus` means use option 81 or option 12 and 15, if specified. `always` means do the A record update for all clients. `XXXXprotected` modifies the **nsupdate** command to make sure the client is allowed. `standard` is a synonym for `always`. `protected` is a synonym for `alwaysprotected`. This is valid at any container level.

Keyword	Form	Subcontainers	Default Value	Meaning
proxyarec	proxyarec standard	No	usedhcpddnsplus	Specifies what options and methods should be used for A record updates in the DNS. `never` means never update the A record. `usedhcpddns` means use option 81 if the client specifies it. `usedhcpddnsplus` means use option 81 or option 12 and 15, if specified. `always` means do the A record update for all clients. `XXXXprotected` modifies the **nsupdate** command to make sure the client is allowed. `standard` is a synonym for `always`. `protected` is a synonym for `alwaysprotected`. This is valid at any container level.
	proxyarec protected	No	usedhcpddnsplus	Specifies what options and methods should be used for A record updates in the DNS. `never` means never update the A record. `usedhcpddns` means use option 81 if the client specifies it. `usedhcpddnsplus` means use option 81 or option 12 and 15, if specified. `always` means do the A record update for all clients. `XXXXprotected` modifies the **nsupdate** command to make sure the client is allowed. `standard` is a synonym for `always`. `protected` is a synonym for `alwaysprotected`. This is valid at any container level.
releasednsA	releasednsA "*string*"	No	None	Specifies the execution string to use when an address is released. The string is used to remove the A record associated with the address released. This is valid at any container level.
releasednsP	releasednsP "*string*"	No	None	Specifies the execution string to use when an address is released. The string is used to remove the PTR record associated with the address released. This is valid at any container level.
removedns	removedns "*string*"	No	None	Specifies the execution string to use when an address is released. The string is used to remove the PTR and A record associated with the address released. This is valid at any container level. **Note:** This is deprecated in favor of the **releasednsA** and **releasednsP** keywords.

Keyword	Form	Subcontainers	Default Value	Meaning
updatedns	updatedns "*string*"	No	None	Specifies the execution string to use when an address is bound. The string is used to update both the A and the PTR record associated with the address. This is valid at any container level. **Note:** This is deprecated in favor of the **updatednsA** and **updatednsP** keywords.
updatednsA	updatednsA "*string*"	No	None	Specifies the execution string to use when an address is bound. The string is used to update the A record associated with the address. This is valid at any container level.
updatednsP	updatednsP "*string*"	No	None	Specifies the execution string to use when an address is bound. The string is used to update the PTR record associated with the address. This is valid at any container level.
hostnamepolicy	hostnamepolicy suggested	No	default	Specifies which host name to return to the client. Default policy is to prefer the defined host name and domain name over suggested names. Other policies imply strict adherence (for example: `defined` will return the defined name or none if no name is defined in the configuration). Also, policies using the `always` modifier will dictate the server to return the host name option regardless of whether the client requested it through the parameter list option. Note that suggesting a host name also implies requesting it, and host names can be suggested through option 81 or through options 12 and 15. This keyword is valid at any container level.

Keyword	Form	Subcontainers	Default Value	Meaning
hostnamepolicy	hostnamepolicy resolved	No	default	Specifies which host name to return to the client. Default policy is to prefer the defined host name and domain name over suggested names. Other policies imply strict adherence (for example: `defined` will return the defined name or none if no name is defined in the configuration). Also, policies using the `always` modifier will dictate the server to return the host name option regardless of whether the client requested it through the parameter list option. Note that suggesting a host name also implies requesting it, and host names can be suggested through option 81 or through options 12 and 15. This keyword is valid at any container level.
hostnamepolicy	hostnamepolicy always_resolved	No	default	Specifies which host name to return to the client. Default policy is to prefer the defined host name and domain name over suggested names. Other policies imply strict adherence (for example: `defined` will return the defined name or none if no name is defined in the configuration). Also, policies using the `always` modifier will dictate the server to return the host name option regardless of whether the client requested it through the parameter list option. Note that suggesting a host name also implies requesting it, and host names can be suggested through option 81 or through options 12 and 15. This keyword is valid at any container level.

Keyword	Form	Subcontainers	Default Value	Meaning
hostnamepolicy	hostnamepolicy defined	No	default	Specifies which host name to return to the client. Default policy is to prefer the defined host name and domain name over suggested names. Other policies imply strict adherence (for example: `defined` will return the defined name or none if no name is defined in the configuration). Also, policies using the `always` modifier will dictate the server to return the host name option regardless of whether the client requested it through the parameter list option. Note that suggesting a host name also implies requesting it, and host names can be suggested through option 81 or through options 12 and 15. This keyword is valid at any container level.
hostnamepolicy	hostnamepolicy always_defined	No	default	Specifies which host name to return to the client. Default policy is to prefer the defined host name and domain name over suggested names. Other policies imply strict adherence (for example: `defined` will return the defined name or none if no name is defined in the configuration). Also, policies using the `always` modifier will dictate the server to return the host name option regardless of whether the client requested it through the parameter list option. Note that suggesting a host name also implies requesting it, and host names can be suggested through option 81 or through options 12 and 15. This keyword is valid at any container level.

Keyword	Form	Subcontainers	Default Value	Meaning
hostnamepolicy	hostnamepolicy default	No	default	Specifies which host name to return to the client. Default policy is to prefer the defined host name and domain name over suggested names. Other policies imply strict adherence (for example: `defined` will return the defined name or none if no name is defined in the configuration). Also, policies using the `always` modifier will dictate the server to return the host name option regardless of whether the client requested it through the parameter list option. Note that suggesting a host name also implies requesting it, and host names can be suggested through option 81 or through options 12 and 15. This keyword is valid at any container level.
bootfilepolicy	bootfilepolicy suggested	No	suggested	Specifies a preference for returning the bootfile name to a client. `suggested` prefers the client-suggested bootfile name to any server-configured name. `merge` appends the client suggested name to the server-configured home directory. `defined` prefers the defined name over any suggested bootfile name. `always` returns the defined name regardless of whether the client requests the bootfile option through the parameter list option.
bootfilepolicy	bootfilepolicy merge	No	suggested	Specifies a preference for returning the bootfile name to a client. `suggested` prefers the client-suggested bootfile name to any server-configured name. `merge` appends the client suggested name to the server-configured home directory. `defined` prefers the defined name over any suggested bootfile name. `always` returns the defined name regardless of whether the client requests the bootfile option through the parameter list option.

Keyword	Form	Subcontainers	Default Value	Meaning
bootfilepolicy	bootfilepolicy defined	No	suggested	Specifies a preference for returning the bootfile name to a client. `suggested` prefers the client-suggested bootfile name to any server-configured name. `merge` appends the client suggested name to the server-configured home directory. `defined` prefers the defined name over any suggested bootfile name. `always` returns the defined name regardless of whether the client requests the bootfile option through the parameter list option.
bootfilepolicy	bootfilepolicy always	No	suggested	Specifies a preference for returning the bootfile name to a client. `suggested` prefers the client-suggested bootfile name to any server-configured name. `merge` appends the client suggested name to the server-configured home directory. `defined` prefers the defined name over any suggested bootfile name. `always` returns the defined name regardless of whether the client requests the bootfile option through the parameter list option.
stealfromchildren	stealfromchildren true	No	No	Specifies whether the parent container should "steal" from children containers when the parent container runs out of addresses. This means that if you have a subnet with class defined with a range of addresses, those addresses are reserved for those clients that specify that class. If `stealfromchildren` is true, then addresses will be pulled from a child to try and satisfy the request. The default is to not steal an address.

Keyword	Form	Subcontainers	Default Value	Meaning
stealfromchildren	stealfromchildren 1	No	No	Specifies whether the parent container should "steal" from children containers when the parent container runs out of addresses. This means that if you have a subnet with class defined with a range of addresses, those addresses are reserved for those clients that specify that class. If `stealfromchildren` is true, then addresses will be pulled from a child to try and satisfy the request. The default is to not steal an address.
stealfromchildren	stealfromchildren yes	No	No	Specifies whether the parent container should "steal" from children containers when the parent container runs out of addresses. This means that if you have a subnet with class defined with a range of addresses, those addresses are reserved for those clients that specify that class. If `stealfromchildren` is true, then addresses will be pulled from a child to try and satisfy the request. The default is to not steal an address.
stealfromchildren	stealfromchildren false	No	No	Specifies whether the parent container should "steal" from children containers when the parent container runs out of addresses. This means that if you have a subnet with class defined with a range of addresses, those addresses are reserved for those clients that specify that class. If `stealfromchildren` is true, then addresses will be pulled from a child to try and satisfy the request. The default is to not steal an address.

Keyword	Form	Subcontainers	Default Value	Meaning
stealfromchildren	stealfromchildren 0	No	No	Specifies whether the parent container should "steal" from children containers when the parent container runs out of addresses. This means that if you have a subnet with class defined with a range of addresses, those addresses are reserved for those clients that specify that class. If `stealfromchildren` is true, then addresses will be pulled from a child to try and satisfy the request. The default is to not steal an address.
stealfromchildren	stealfromchildren no	No	No	Specifies whether the parent container should "steal" from children containers when the parent container runs out of addresses. This means that if you have a subnet with class defined with a range of addresses, those addresses are reserved for those clients that specify that class. If `stealfromchildren` is true, then addresses will be pulled from a child to try and satisfy the request. The default is to not steal an address.
homedirectory	homedirectory *path*	No	None	Specifies the home directory to use in the file section of the response packet. This can be specified at any container level. The bootfile policy defines how items specified in the file section of the incoming packet interact with the bootfile and the home directory statements.
bootfile	bootfile *path*	No	None	Specifies the bootfile to use in the file section of the response packet. This can be specified at any container level. The bootfile policy defines how items specified in the file section of the incoming packet interact with the bootfile and the home directory statements.

Keyword	Form	Subcontainers	Default Value	Meaning
pxebootfile	pxebootfile *system_architecture major_version minor_version boofilename*	No	None	Specifies the bootfile to be given for a client. This is used only when **dhcpsd** supports PXE clients (**pxeservertype** is **dhcp_pxe_binld**). The configuration file parser generates an error if the number of parameters after `pxebootfile` is less than four, and it ignores any additional parameters. `pxebootfile` can only be used within a container.
supportoption118	supportoption118 *no/yes*	No. Can be defined only in subnet container.	None	This keyword specifies whether this container supports the option 118. Yes mean it is supported, and No means it is not supported. For this option to take effect, you have to also use the keyword **supportsubnetselection**.

DHCP and Network Installation Management suggestions

The concept of dynamically assigning Internet Protocol (IP) addresses is fairly new. The following suggestions are provided to help with **DHCP** and Network Installation Management (NIM) interaction.

1. When configuring objects in the NIM environment, use host names whenever possible. This allows you to use a dynamic name server that updates the IP addresses when the host name is converted to an IP address in the NIM environment.
2. Place the NIM master and the **DHCP** server on the same system. The **DHCP** server has an option in the update DNS string that, when set to `NIM`, attempts to keep the NIM objects out of those states that need static IP addresses when those addresses change.
3. For NIM clients, set the default lease time to twice the time it takes to install a client. This allows a leased IP address to be valid during the installation. After the installation, restart the client. **DHCP** will be started or will need to be configured, depending on the type of installation.
4. The `dhcpsd` server should be responsible for both the PTR and the A DNS records. When NIM reinstalls the machine, the file containing the RSA is deleted, and the client cannot update its records. The server updates the system records. To do this, change the `updatedns` line in `/etc/dhcpcd.ini` to:

   ```
   updatedns "/usr/sbin/dhcpaction '%s' '%s' '%s' '%s' '%s' NONE NONIM"
   ```

   In the `/etc/dhcpsd.cnf` file, change the `updatedns` line to:

   ```
   updatedns "/usr/sbin/dhcpaction '%s' '%s' '%s' '%s' '%s' BOTH NIM"
   ```

   Note: When a NIM object is placed into the BOS installation-pending state, the `dhcpsd` server might pass arguments that are different from those originally intended. Minimize the time the client is in this pending state to avoid this situation.

These suggestions allow the NIM environment to work with dynamic clients.

For more information on Network Installation Management, see *AIX 5L™ Version 5.3 Network Installation Management Guide and Reference*.

Dynamic Host Configuration Protocol version 6

The **Dynamic Host Configuration Protocol (DHCP)** provides a method to maintain network configurations in a centralized location. This topic is **DHCPv6**-specific; all references to "IP address" refer to IPv6 addresses, and all references to "**DHCP**" refer to **DHCPv6** (unless otherwise stated).

A **DHCPv4** server can coexist on the same link with a **DHCPv6** server. For an in-depth explanation of the protocol, see RFC 3315.

DHCP is an application-layer protocol that allows a client machine on the network to get IP addresses and other configuration parameters from the server. These parameters are defined in *options*. Options are obtained by exchanging packets between a daemon on the client and another on the server. These message exchanges are in the form of **UDP** packets. A client uses a link-local address, whether through the **autoconf6** command or other methods, to identify its source address to the server. The server listens on a reserved link-scope multicast address. A relay agent will allow the client and server to communicate if they are not located on the same link.

This topic explains the four-message exchange handshake for a single interface with one IA_NA and one address for this IA_NA. To obtain an IP address, the **DHCP** client daemon (**dhcpcd6**) sends a `SOLICIT` message to the **All_DHCP_Relay_Agents_and_Servers** address, which is received by the server and processed. (Multiple servers can be configured on the network for redundancy.) If a free address is available for that client, an `ADVERTISE` message is created and sent back to the client. This message contains an IP address and other options that are appropriate for that client. The client receives the server `DHCP ADVERTISE` message and stores it while waiting for other advertisements. When the client has chosen the best advertisement, it sends a `DHCP REQUEST` to the **All_DHCP_Relay_Agents_and_Servers** address specifying which server advertisement it wants.

All configured **DHCP** servers receive the `REQUEST` message. Each checks to see if it is the requested server. The server does not process any packet with a server DUID that does not match its own. The requested server marks the address as assigned and returns a `DHCP REPLY`, at which time, the transaction is complete. The client has an address for the period of time (`valid-lifetime`) designated by the server.

When the preferred-lifetime expires for the address, the client sends the server a `RENEW` packet to extend the lease time. If the server is willing to renew the address, it sends a `DHCP REPLY`. If the client does not get a response from the server that owns its current address, it multicasts a `DHCP REBIND` packet if, for example, the server has been moved from one network to another. If the client has not renewed its address after the valid-lifetime, the address is removed from the interface and the process starts over. This cycle prevents multiple clients on a network from being assigned the same address.

A client can have multiple IA_NA options, and each IA_NA can have multiple addresses. A client can also have multiple IA_TA options and each can also have multiple addresses:

- **Identity association for non-temporary addresses (IA_NA)**: An IA that carries assigned addresses that are not temporary addresses
- **Identity association for temporary addresses (IA_TA)**: An IA that carries temporary addresses (see RFC 3041).
- **DUID**: A **DHCP**-Unique identifier for a **DHCP** participant; each **DHCP** client and server has a unique DUID which remains the same across reboots.

The **DHCP** server assigns addresses based on keys. Four common keys are **class**, **vendor**, **client ID**, and **inoption**. The server uses these keys to allocate an address, and the set of configuration options to return to the client.

class The **class** key is completely client-configurable. It can specify an address and options. This key can be used to denote machine function in the network or to describe how machines are grouped for administrative purposes. For example, the network administrator might want to create a

NetBIOS class that contains options for NetBIOS clients or an accounting class that represents accounting department machines that need access to a specific printer.

vendor
The **vendor** key helps identify the client by its hardware and software platform.

client ID
The **client ID** keyidentifies the client through the DUID. The client ID is specified in the `duid` file of the **dhcpcd** daemon. Also, the client ID can be used by the server to pass options to a specific client or prohibit a particular client from receiving any parameters.

Inoption
The **inoption** key identifies the client by the option requested by the client.

These keys can be used either singularly or in combinations. If multiple keys are provided by the client and multiple addresses can be assigned, only one is chosen, and the option set is derived from the chosen key first.

A relay agent is needed so initial multicast from the client can leave the local network. The relay agents act as forwarding agents for **DHCP** packets.

DHCPv6 server

There are three main components of the **DHCPv6** server.

The **DHCP** server has been segmented into three main components: a database, a protocol engine, and a set of service threads. Each component has its own configuration information.

DHCPv6 database:

The `db_filev6.dhcpo` database is used to track clients and addresses and for access control.

Options are also stored in the database for retrieval and delivery to clients. The database is implemented as a dynamically-loadable object.

Using the information in the configuration file, the database is primed and verified for consistency. The database also contains the address and option pools

The main storage file and its back up are ASCII files. The format for the database main storage files are as follows:

Note: Do not manually edit these files.

```
DB6-1.0
Client-Info {
duid 1-0006085b68e20004ace491d3
state 7
authinfo {
    protocol 2
    algorithm 1
    rdm 0
    replay 1206567640
}
  Interface 0 {
    Inoptions {
    interface-id "en1"
    policies 2
    maxopcode 16
    numiana 1
      Ianalist {
      option 3 40 000000010000003200000050000050018deaddeadaaaaaaaa0000000000000006000000640000000c8
      }
    numiata 0
```

```
      Optiontable {
      option 6 10 00030004001700180237
      option 8 2 e659
      option 15 14 000369626d000373756e00026870
      option 16 18 000004d200073078313131311000369626d
      }
    }
    Ianarec {
    IAID 1
    t1 50
    t2 80
      Addrec {
      Address dead:dead:aaaa:aaaa::6
      state 3
      starttime  1087592918
      preferred-lifetime 100
      valid-lifetime 200
      }
    }
  }
}
```

The first line is a version identifier for the file: `DB6-1.0`. The lines that follow are client record definition lines. The server reads from the second line to the end of the file. (The parameters in quotes must be enclosed in quotes.)

duid The ID the client uses to represent itself to the server.

Interface
A client can have multiple interfaces. If a client has a single interface and creates individual **SOLICIT** messages for each IA_NA or IA_TA, the file will contain multiple interfaces for this client.

Inoptions
The incoming options from the client.

policies
Flag to identify unicast, reconfig-option, and rapid-commit.

maxopcode
The largest option code.

numiana
The number of IA_NAs for this interface.

Ianalist
The list of IA_NA options incoming from the client.

numiata
Number of IA_TAs for this interface.

Optiontable
The list of options requested by the client excluding the IA_NA and IA_TA options.

Ianarec
The saved IA_NA record container from the server database.

IAID The ID of the IA_NA.

t1 The preferred-lifetime percent for this IA_NA.

t2 The valid-lifetime percent for this IA_NA.

Addrec
The address record container from the server database.

Address
Address given to the client for this address record.

state The current state of the client. The **DHCP** protocol engine contains the allowable set, and the states are maintained in the **DHCP** database. The number next to **state** represents its value. The states can be:

(1) FREE
Represents addresses that are available for use. In general, clients do not have this state unless they have no address assigned. The **dadmin** and **lssrc** commands report this state as `Free`.

(2) BOUND
Indicates client and address are tied and that the client has been assigned this address for some amount of time. The **dadmin** and **lssrc** commands report this state as `Leased`.

(3) EXPIRED
Indicates the client and address are tied together, but only for informational purposes, in a similar manner to released addresses. The expired state, however, represents clients that let their leases expire. An expired address is available for use and is reassigned after all free addresses are unavailable and before released addresses are reassigned. The **dadmin** and **lssrc** commands report this state as `Expired`.

(4) RELEASED
Indicates the client and address are tied for informational purposes only. The **DHCP** protocol suggests that **DHCP** servers maintain information about the clients it has served for future reference (mainly to try giving the same address to that client that has been assigned that address in the past). This state indicates that the client has released the address. The address is available for use by other clients, if no other addresses are available. The **dadmin** and **lssrc** commands report this state as `Released`.

(5) RESERVED
Indicates client and address are tied, but loosely. The client has issued a **DHCP** discover message and the **DHCP** server has responded, but the client has not yet responded with a **DHCP** request for that address. The **dadmin** and **lssrc** commands report this state as `Reserved`.

(6) BAD
Represents an address that is in use in the network but has not been handed out by the **DHCP** server. This state also represents addresses that clients have rejected. This state does not apply to clients. The **dadmin** command reports this state as `Used`, and the **lssrc** command reports this state `Bad`.

Starttime
The time that this address was handed out, represented as seconds since January 1, 2000.

preferred-lifetime
Number in seconds before this address needs to be renewed.

valid-lifetime
Number in seconds before this address becomes invalid and can no longer be used.

protocol
The authentication protocol the client is using:

(1) DELAYED
The client is using delayed authentication.

(2) RECONFIGURE KEY
The client is using reconfigure key authentication.

algorithm
The authentication algorithm the client is using:

(1) HMAC-MD5
The client is using the keyed MD5 algorithm to create the message digest.

rdm The replay detection method the client is using:

(0) Monotonically increasing counter
The client is using a monotonically increasing counter to modify the replay value.

replay The current value of the replay field.

The syntax for the checkpoint files is not specified. If the server crashes or you have to shut down and cannot do a normal closing of the database, the server can process the checkpoint and backup files to reconstruct a valid database. Any client not written to the checkpoint file when the server crashes is lost. Currently, there are no intermittent saves when a client is processed. The default files are:

/etc/dhcpv6/db_file6.cr
Normal database operation

/etc/dhcpv6/db_file6.crbk
Backups for the database

DHCP threaded operations:

The last piece of the **DHCP** server is actually a set of operations that are used to keep things running.

Because the **DHCP** server is threaded, these operations are actually set up as threads that occasionally do things to make sure everything is together.

main thread
This thread handles signals. For example,

- A SIGHUP (-1) causes a refresh of all databases in the configuration file.
- A SIGTERM (-15) will cause the server to gracefully stop.
- A SIGUSR1 (-30) will cause the server to dump the configuration database

src thread
This thread handles the SRC requests (such as **startsrc**, **stopsrc**, **lssrc**, **traceson**, and **refresh**).

dadmin thread
This thread interfaces with the **dadmin** client program and the **DHCP** server. The **dadmin** tool can be used to get status as well as modify the database to avoid editing the database files manually. With the addition of the **dadmin** and **src** threads, the server can handle service requests and still handle client requests.

garbage thread
This thread runs timers that periodically clean the database, save the database, purge clients that do not have addresses, and remove reserved addresses that have been in reserve state for too long. All these timers are configurable.

packet processors
Each of these can handle a request from a **DHCPv6** client. The number of packet processors required is somewhat load and machine-dependent. The number of these is configurable; the default is 1. The maximum number of packet threads is 50.

logging threads
In a system where significant amounts of data is being logged to log files, the number of logging threads can be increased more than the default (1) to max (50).

table manager thread
This thread ensures that the **dhcpsdv6** daemon does not process duplicate packets.

process threads
These threads process the **DHCPv6** client packets.

reconfigure thread
This thread manages client reconfiguration when the server is refreshed (with the `dadmin -x 6 -i` command, for example).

DHCPv6 configuration

By default, the **DHCP** server is configured by reading the `/etc/dhcpv6/dhcpsdv6.cnf` file, which specifies the initial database of options and addresses.

The server is started from SRC commands. If **dhcpsdv6** is intended to start across reboots, add an entry into the `/etc/rc.tcpip` file.

Configuring the **DHCP** server is usually the hardest part of using **DHCP** in your network. First, decide what networks you want to have **DHCP** clients on. Each subnet in your network represents a pool of addresses that the **DHCP** server must add to its database. For example:

```
subnet dead:dead:aaaa:: 48 {
        option 23 dead::beef beef:aaaa::bbbb:c aaaa:bbbb::cccc  #nameserver list
        option 24 austin.ibm.com ibm.com                        # domain list
 }
```

The example above shows a subnet, `dead:dead:aaaa::`, with a prefix of 48 bits. All addresses in this subnet, `dead:dead:aaaa::1` through `dead:dead:aaaa:ffff:ffff:ffff:ffff:ff7f`, are in the pool. Optionally, a range can be specified on the end of the line before the '{' or a range or exclude statement can be included in the subnet container.

Comments begin with a # (pound sign). Text from the initial #, to the end of the line, is ignored by the **DHCP** server. Each option line is used by the server to tell the client what to do.

If the server does not understand how to parse an option, it uses default methods to send the option to the client. This also allows the **DHCP** server to send site-specific options that are not RFC defined, but may be used by certain clients or client configurations.

DHCPv6 configuration file:

The configuration file has an address section and an option definition section. These sections use containers to hold options, modifiers, and, potentially, other containers.

A container (a method to group options) uses an identifier to classify clients into groups. The container types are **subnet**, **class**, **vendor**, **inoption**, and **client**. Currently, there is not a generic user-definable container. The identifier uniquely defines the client so that the client can be tracked if, for example, it moves between subnets. More than one container type can be used to define client access.

Options are identifiers that are returned to the client, such as DNS address or domain names.

After selecting modifiers, the next item to set up is logging. Logging parameters are specified in a container like the database, but the container keyword is **logging_info**. When learning to configure **DHCP**, it is advisable to turn logging to its highest level. Also, it is best to specify the logging configuration before any other configuration file data to ensure that configuration errors are logged after the logging subsystem is initialized. Use the **logitem** keyword to turn on a logging level or remove the **logitem** keyword to disable a logging level. Other keywords for logging allow the specification of the log filename, file size, and the number of rotating log files.

DHCPv6 containers:

When the **DHCP** server receives a request, the packet is parsed and identifying keys determine which containers, options, and addresses are extracted.

Each type of container uses a different option to identify a client:

- The **subnet** container uses the **hintlist** field or the interface address of the receiving interface to determine from which subnet the client belongs.
- The **class** container uses the value in option 15 (OPTION_USER_CLASS Identifier).
- The **vendor** uses the value in option 16 (OPTION_VENDOR_CLASS).
- The **client** container uses the option 1 (OPTION_CLIENTID) from the DHCP client's DUID.
- The **inoption** container matches the client's requested option.

Except for subnets, each container allows the specification of the value that matches it, including regular expression matching.

There is also an implicit container, the global container. Options and modifiers are placed in the global container unless overridden or denied. Most containers can be placed inside other containers implying a scope of visibility. Containers may or may not have address ranges associated with them. Subnets, by their nature, have ranges associated with them.

The basic rules for containers and subcontainers are:
- Only subnet containers are valid at the global level.
- Subnets cannot be placed inside other containers, including itself.
- Restricted containers cannot have regular containers of the same type within them. (For example, a container with an option that only allows a class of Accounting cannot include a container with an option that allows all classes that start with the letter a.)
- Restricted client containers cannot have subcontainers.
- Inoption containers cannot have subcontainers

Given the above rules, you can generate a hierarchy of containers that segment your options into groups for specific clients or sets of clients.

If a client matches multiple containers, The **DHCP** server passes the request to the database, and a container list is generated. The list is presented in order of depth and priority. Priority is defined as an implicit hierarchy in the containers. Strict containers are higher priority than regular containers. Clients, classes, vendors, and subnets are sorted, in that order, and within container type by depth. This generates a list ordered by most specific to least specific. For example:

```
Subnet 1
    --Class 1
    --Client 1
Subnet 2
    --Class 1
    ----Vendor 1
    ----Client 1
    --Client 1
```

The example shows two subnets, `Subnet 1` and `Subnet 2`. There is one class name, `Class 1`, one vendor name, `Vendor 1`, and one client name, `Client 1`. `Class 1` and `Client 1` are defined in multiple places. Because they are in different containers, their names can be the same but values inside them can be different. If `Client 1` sends a message to the **DHCP** server from `Subnet 1` with `Class 1` specified in its option list, the **DHCP** server would generate the following container path:

```
Subnet 1, Class 1, Client 1
```

The most specific container is listed last. To get an address, the list is examined in reverse hierarchy to find the first available address. Then, the list is examined in forward hierarchy to get the options. Options override previous values unless an option deny is present in the container. Also, because `Class 1` and `Client 1` are in `Subnet 1`, they are ordered according to the container priority. If the same client is in Subnet 2 and sends the same message, the container list generated is:

```
Subnet 2, Class 1, Client 1 (at the Subnet 2 level), Client 1 (at the Class 1 level)
```

`Subnet 2` is listed first, then `Class 1`, then the `Client 1` at the `Subnet 2` level (because this client statement is only one level down in the hierarchy). The hierarchy implies that a client matching the first client statement is less specific than the client matching `Client 1` of `Class 1` within `Subnet 2`.

Priority selected by depth within the hierarchy is not superseded by the priority of the containers themselves. For example, if the same client issues the same message and specifies a vendor identifier, the container list is:

```
Subnet 2, Class 1, Vendor 1, Client 1 (at Subnet 2 level), Client 1 (at Class 1 level)
```

Container priority improves search performance because it follows a general concept that client containers are the most specific way to define one or more clients. The class container holds less specific addresses than a client container; vendor is even less specific; and subnet is the least specific.

DHCPv6 addresses and address ranges:

Any container type can have associated address ranges; subnets must have associated address ranges.

Each range within a container must be a subset of the range and must not overlap with ranges of other containers. For example, if a class is defined within a subnet and the class has a range, the range must be a subset of the subnet range. Also, the range within that class container cannot overlap with any other ranges at its level.

Ranges can be expressed on the container line and modified by range and exclude statements to allow for disjoint address sets associated with a container. If you have the top ten addresses and the second ten addresses of a subnet available, the subnet can specify these addresses by range in the subnet clause to reduce both memory use and the chance of address collision with other clients not in the specified ranges.

After an address has been selected, any subsequent container in the list that contains address ranges is removed from the list along with its children. Network-specific options in removed containers are not valid if an address is not used from within that container.

DHCPv6 configuration file options:

After the list has been culled to determine addresses, a set of options is generated for the client.

In this selection process, options overwrite previously selected options unless a deny is encountered, in which case, the denied option is removed from the list being sent to the client. This method allows inheritance from parent containers to reduce the amount of data that must be specified.

DHCPv6 server-specific options:

The last set of parameters to specify are server-specific options that allow the user to control the number of packet processors, how often the garbage collection threads are run, and so on.

For example, two server-specific options are:

reservedTime
: Indicates how long an address stays in the reserved state after sending an ADVERTISE to the **DHCP** client

reservedTimeInterval
: Indicates how often the **DHCP** server scans through the addresses to see if there are any that have been in the reserved state longer than *reservedTime*.

These options are useful if you have several clients that multicast SOLICIT messages and, either they do not multicast their REQUEST message, or their REQUEST message gets lost in the network. Using these parameters keeps addresses from being reserved indefinitely for a noncompliant client.

Another particularly useful option is *SaveInterval*, which indicates how often saves occur.

/etc/dhcpv6/dhcpsdv6.cnf file:

The **DHCPv6** server is configured by editing the `/etc/dhcpv6/dhcpsdv6.cnf` file.

The keywords are case sensitive. When a '{' is listed, it must be on the same line as the keyword. A sample configuration file can be found in `/usr/samples/tcpip/dhcpv6`.

Following is the description of the `/etc/dhcpv6/dhcpsdv6.cnf` file. The following stanzas are allowed in this file:

- Logging
- Global keywords
- Non-nested container statements
- Nested container statements
- Options
- Common options

DHCPv6 logging:

The **DHCPv6** server keywords described here are for entries in the logging stanza.

This stanza is not required to exist but if present, must be at the top of the configuration file. It has the following format:

```
logging_info { log_options }
```

The *log_options* values can be any of the following:

Table 63. Keywords, values, and descriptions for entries in the logging stanza.

Keyword	Value	Description
logFileSize	*num*	Specifies the size of the log file. The *num* value is the maximum size of the log file in kilobytes. The log file will be rotated after this size is reached. Infinite size is assumed if `logFileSize` is not specified.
logFileName	"*filename*"	Specifies the name of the log file. The *filename* value will be the name of the log file. The default file name and location is `/var/tmp/dhcpsdv6.log`.
numLogFiles	*num*	Specifies the number of log files for file rotation. The default is 0.

Table 63. Keywords, values, and descriptions for entries in the logging stanza. (continued)

Keyword	Value	Description
logItem	*type*	Specifies the types of logging desired. The following types are valid: **SYSERR** System error, at the interface to the platform. **OBJERR** Object error, in between objects in the process. **PROTERR** Protocol error, between client and server. **WARNING** Warning, worth of attention from the user. **EVENT** Event occurred to the process. **ACTION** Action taken by the process. **INFO** Information that might be useful. **ACNTING** Who was served when. **TRACE** Code flow, for debugging.

DHCPv6 global keywords:

The keyword values described here are for entries in the global keyword stanza.

The global keywords are only valid outside a container. The following values are allowed:

Table 64. Keywords, values, and descriptions for entries in the global keywords stanza.

Keyword	Value	Description
UsedIpAddressExpiredInterval	num [*units*]	Specifies how often addresses placed in the BAD state are recouped and retested for validity. If a unit is not set, the system default is set to seconds. The default value is -1.
leaseExpiredInterval	num [*units*]	Specifies how often addresses in the BOUND state are checked to see if they have expired. If the address has expired, the state is moved to EXPIRED. If a unit is not set, the system default is set to seconds. The default value 900 seconds.
reservedTime	num [*units*]	Specifies how long addresses should sit in RESERVED state before being recouped into the FREE state. If unit is not set, the system default is set to seconds. The default value is -1.
reservedTimeInterval	num [*units*]	Specifies how often addresses in the RESERVE state are checked to see if they should be recouped into the FREE state. If unit not set, the system default is set to seconds. The default value is 900 seconds.
saveInterval	num [*units*]	Specifies how often the **DHCP** server should force a save of the open databases. For heavily loaded servers, this should be 60 or 120 seconds. If unit is not set, the system default is set to seconds. The default value is 3600 seconds.
clientpruneintv	num [*units*]	Specifies how often the **DHCP** server has the databases remove clients are not associated with any address (in the UNKNOWN state). This reduces the memory use of the **DHCP** server. If units is not set, the system default is set to seconds. The default value is 3600 seconds.
numprocessthreads	num	Specifies the number of packet processors threads to create. Minimum of one. Each processthread handles one client. By default it is 30.

Table 64. Keywords, values, and descriptions for entries in the global keywords stanza. (continued)

Keyword	Value	Description
numpacketthreads	num	Specifies the number of packet threads to create. This minimum is 1, but by default it is set to 5.
numloggingthreads	num	Specifies the number of logging threads. Default is 1.
numduidbuckets	num	This is used by the table manager, and has a direct correlation to the **numprocessthreads**. By default, it is set to 53.
numclientbuckets	num	How many buckets will be used to store the client records. By default it is 1021.
ignoreinterfacelist	*interface* [*interface*]	List of interfaces to ignore. It can be a single interface or multiple interfaces.
backupfile	"*filename*"	The file to use for database backups. The default file is /etc/dhcpv6/db_file6.crbk
checkpointfile	"*filename*"	Specifies the database checkpoint files. The first checkpoint file is the path. The second checkpoint file is path with the last character replaced with a 2. So, the checkpoint file should not end in 2.
clientrecorddb	"*filename*"	Specifies the database save file. The file contains all the client records the **DHCP** server has serviced. The default file is **/etc/dhcpv6/db_file6.cr**
duid	idtype *value* [*value*]	Used to identify the server. The following values are allowed: • duid 1 *interface* • duid 2 *interface* • duid 3 enterprise number *identifier* • duid number 0x*hexdigit*
preference-number	num	Allows the clients to identify the server it prefers to obtain information from. The higher the value, the greater the chance the client will use this server for its services. The default and maximum value is 255.
unicast-enable	*policy*	Unicast policy for the server, this allows the server to communicate using unicast. By default this is turned on.
tablemgr-policy	*policy*	Allows the server to have a table manager to better manage incoming clients. By default this is turned on.
auth	*policy*	Allows the server to support delayed authentication. By default this is turned off.
auth-keyfile	"*filename*"	The file that contains the delayed authentication keys for the clients. The default file is **/etc/dhcpv6/dhcpsdv6.keys**.

DHCPv6 non-nested container statements:

The **DHCPv6** server keyword **subnet** is for entries in the non-nested container statements.

Non-nested container statements can only exist as part of the global keywords.

Table 65. Keywords, values, and descriptions for entries in the non-nested container statements.

Item	Description	
subnet	*subnetid prefix-length* [*range*] {OPTIONS}	Specifies subnet to be used. The *subnetid* must be an IPv6 address. The *prefix-length* must be a positive integer less than 128.

DHCPv6 nested container statements:

Nested container statements can only exist as an option inside subnet.

All containers can have other containers nested within it unless otherwise stated. The maximum depth of nesting is seven, including the subnet and the global container (only five nested containers can exist under a subnet container).

Vendor and Inoption containers cannot have any other containers nested within it.

Table 66. Keywords, values, and descriptions for entries in the nested container statements.

Keyword	Value	Description
class	*name* [*range*] {OPTIONS|COMMON OPTIONS }	Class container. The *name* value is one string, space separated strings, regular expression, hex 0x*hexdigit*, 0x*hexdigit*
vendor	*name* [*range*] {OPTIONS|COMMON OPTIONS }	Vendor container. The *name* value is one string, space separated strings, regular expression, hex 0x*hexdigit*, 0x*hexdigit*
client	<*id* | 0 0xhexdigit | regular expression> <ip | *range* | none | any> {OPTIONA|COMMON OPTIONS }	Client container. *id*- 1-hexdigit, 2-hexdigit, 3-hexdigit <ip|range|none|any> - IP address to give to clients that match the ID
inoption	*icode keytomatch* [*range*] { OPTIONS|COMMON OPTIONS }	Inoption container *icode* - incoming option code or number to be specified by the client *keytomatch* - The option data to be matched against.

DHCPv6 cnf file options:

The cnf file options described here for **DHCPv6** can only exist inside a container.

Table 67. Keywords, values, and descriptions for entries in the options stanza.

Keyword	Value	Description
exclude	*range*	IP range to exclude from the current range, often used when a range is not specified as part of the container statement
exclude	ip	IP address to exclude from the current range
range	*range*	IP range use to extend the current range, often used when a range is not specified as part of the container statement
range	ip	IP address to add, used to extend the range
stealfromchildren	*policy*	Steal address from children containers if all addresses are exhausted. By default this is turned off.
stealfrompeer	*policy*	Steal addresses from peer containers if all addresses are exhausted. By default this is turned off.
stealfromparent	*policy*	Steal addresses from parent containers if all addresses are exhausted. By default this is turned off.
balance-option	{ balance-policy | <option | option option ...> }	Balance options container, options specified within this container will be given to the client base on the policy. This keyword can only exist under the subnet container.
balance-policy	b_policy	The b_policy value can be fill or rotate. The default is rotate.
fill-count	num	The number of times an option will be given out before handing out the next instance of the same option
interface-id	"*interface*"	This can only be listed under subnet. Clients requests received on this interface will be allowed to obtain addresses.

DHCPv6 common options:

These keywords are common **DHCPv6** options.

These can exist inside the containers or in the global section:

Table 68. Keywords, values, and descriptions for common options.

Keyword	Value	Description
reconfig-policy	*policy*	Allows the server to send reconfiguration message to the client. By default this is not set and is considered off.
rapid-commit	*policy*	Allows for the server to do rapid commit for the container or globally set. By default this is not set and is considered off.
prefered-lifetime	`num [units]`	The preferred lifetime of the IANA or IATA. The default is 43200 seconds.
valid-lifetime	`num [units]`	The valid lifetime of the IANA or IATA. The default is 86400 seconds.
rebind	`num`	The rebind time percent 0-100 for the address. The default value is 80 percent.
renew	`num`	The renew time percent 0-100 for the address. The default value is 50 percent.
unicast-option	`policy`	Allows containers to offer message exchange by unicasting, this can use used to turn on and off individual containers and subnets even if the server policy differs. By default this is not set and is considered off.
option	`num <string|stings| hex>`	For the list of options, see "DHCPv6 server file known options."
change-optiontable	`optiontable`	Only allowed within a vendor container.

DHCPv6 server file known options:

The known file options of the **DHCPv6** server are described here.

The following options are the **DHCPv6** server file known options. The options that have "No" in the **Can Specify** column cannot be specified in the configuration file; if they are specified, they will be ignored.

Option number	Default data type	Can specify?	Description
1	None	No	Solicit
2	None	No	Advertise
3	None	No	Request
4	None	No	Confirm
5	None	No	Address
6	None	No	Option Request
7	number	No	The preference number of the server
8	None	No	Elapse Time
9	None	No	Relay Message
11	None	No	Auth
12	ASCII string yes, no, true, false	Yes	Unicast
13	None	No	Status
14	ASCII string yes, no, true, false	Yes	Rapid Commit
15	None	No	User Class
16	None	No	Vendor Class
17	None	No	Vendor Option
18	None	No	Interface ID
19	None	No	Reconfiguration Message
20	ASCII string yes, no, true, false	Yes	Reconfiguration Accept
23	Space separated IPv6 addresses	Yes	DNS servers

Option number	Default data type	Can specify?	Description
24	ASCII string	Yes	Domain list

DHCPv6 parameter values:

These values can be used for the **DHCPv6** parameters.

units: second, seconds, minute, minutes, hour, hours, day, days, week, weeks, month, months, year, years

interface: en0, en1, tr0

identifier: numbers or characters

policy: yes, no, true, false

range: ipv6addresss-ipv6addresss

regular expression: "!epression to match$", "!epression to match^"

Example /etc/dhcpv6/dhcpsdv6.cnf file:

The sample `/etc/dhcpv6/dhcpsdv6.cnf` file shown here provides a glimpse of the file's contents.

```
logging_info{
        logFileSize 4000
        logItem     SYSERR
        logItem     PROTERR
        logItem     WARNING
        logItem     EVENT
        logItem     ACTION
        logItem     INFO
        logItem     ACNTING
        logItem     TRACE
        numLogFiles     3
        logFileName "/var/tmp/dhcpsdv6.log"
}
duid 1 en0
numprocessthreads 10
numpacketthreads  5
preference-number 255
reconfig-policy no
rapid-commit  no
unicast-option    yes
leaseExpiredInterval  3000 seconds
unicast-enable          yes
saveInterval            60 seconds
reservedTimeInterval    8000 seconds
reservedTime            10000 seconds
clientpruneintv         20  seconds

subnet bbbb:aaaa:: 40 bbbb:aaaa::0004-bbbb:aaaa::000f {
        balance-option {
               option 23 dead::beef
               option 23 beef::aaaa
               option 24 yahoo.com
        }

subnet dead:dead:aaaa:: 48 dead:dead:aaaa:aaaa::0006-dead:dead:aaaa:aaaa::000a {
        interface-id "en1"
        preferred-lifetime      100 seconds
        valid-lifetime          200 seconds
```

```
        rapid-commit  yes
        option 23 dead::beef beef:aaaa::bbbb:c aaaa:bbbb::cccc
        option 24 ibm.com austin.ibm.com
}
```

DHCPv6 client configuration

The `/etc/dhcpv6/dhcpc6.cnf` file is used to configure the **DHCPv6** clients.

The directives that can be specified in this file are included here. If **dhcpcd6** is intended to start across reboots, add an entry into the `/etc/rc.tcpip` file.

Logging keywords:

The valid logging keywords of the **DHCPv6** server are described here.

The following keywords are valid:

Table 69. Keywords and descriptions for logging keywords.

Keyword	Description
log-file-name	The path and file name of the most recent log file. The less recent filenames have number 1 to (n-1) appended to the filename; the larger the number the less recent the file.
log-file-size	Specifies the maximum size of a log file in KB. When the size of the most recent log file reaches this value, it is renamed and a new file is created.
log-file-num	Specifies the maximum number of log files maintained when the size of the most recent log file reaches the **log-file-size** value and the file is renamed to generate a new file.
log-item	Specifies the log items that need to be logged. **SYSERR** System error **OBJERR** Object error **PROTERR** Protocol error **WARNING** Warning **EVENT** Event occurred **ACTION** Action taken by the process **INFO** Additional information **ACNTING** Who was served when **TRACE** code flow, debugging

DUID keywords:

The following keyword values are for DUID entries.

The format of the DUID entries are as follows:

```
duid  <duid_type> <value>  <value> ...
```

DUID type can be a keyword or a number, leaving room for any DUID types that may be defined in future. There are three DUID types currently defined by RFC 3315 :

Table 70. Keywords and values for DUID entries.

Keyword	Description
LLT	DUID-LLT type (value 1)
LL	DUID-LL (value 2)
EN	DUID-EN type (value 3)

The specific format of the DUID entries depends on the keyword being used.

```
duid  LLT    <interface name>
duid  LL     <interface name>
duid  EN     <enterprise number>  <enterprise identifier>
duid  <number>  <hex data (prefixed with '0x')
```

Information-only keyword:

The information-only keyword is in the format `info-only` *`interface name`*.

Following is the information-only keyword:

Table 71. Keyword and description of the information-only keyword.

Keyword	Description
`info-only` *`interface name`*	This keyword specifies the interface name for which the client has to obtain only configuration information and not addresses from the server.

Lease renewal and rebind keywords:

The lease renewal and rebind keywords described here are for the **DHCPv6** server.

Table 72. Keywords and descriptions for lease renewal and rebind keywords

Keywords	Description
`rebind-time` *`value`*	In the event of the client failing to renew its lease (because the server does not respond), the rebind-time specifies the time at which the client contacts the other servers to rebind the lease.
`renew-time` *`value`*	The renew-time specifies the time at which the client contacts the server from which the client obtained lease information, in order to renew the lease.

Solicit retransmission keywords:

The solicit retransmission keywords include **solicit-maxcount** and **solicit-timeout**.

Table 73. Keywords and descriptions for solicit retransmission keywords

Keywords	Descriptions
solicit-maxcount	The solicit-maxcount keyword specifies the number of solicit messages that the client sends to the server before the client receives a response from the server.
solicit-timeout	The solicit-timeout keyword specifies the time until which the client attempts to send solicit message to the server before the client receives a response from the server.

Option keywords:

If the option keywords appear outside of 'interface' stanzas, then they are considered global. Such options apply to all interfaces. If the option keywords appear inside 'interface' stanzas, these options apply only to that interface.

The options stanza follows this format:

```
option <keyword | option code>
option <keyword | option code> exec "exec string"
option <keyword | option code> { option specific parameters }
option <keyword | option code> { option specific parameters } exec "exec string"
```

An option code can be specified using the IANA registered option code. However, some of the options can also be specified using keywords shown below:

Keyword	Option Code
ia-na	3
ia-ta	4
request-option	6
rapid-commit	14
user-class	15
vendor-class	16
vendor-opts	17
reconf-accept	20
dns-servers	23
domain-list	24

Additional explanation of each keyword follows:

Keyword	Purpose, format, and parameters
ia-na	**Purpose** Specifies option 3. If specified, the client requests for non-temporary addresses from the server. **Format** `option ia-na [ { parameters } ] [ exec "exec string" ]` **Parameters** Option **ia-na** takes the following parameters: ia-id *value* renew-time *value* rebind-time *value* These parameters specify the user-preferred values and are optional. The *value* specified can be a decimal number or a hex number prefixed with '0x'
ia-ta	**Purpose** Specifies option 4. If specified, the client requests for a temporary addresses from the server. **Format** `option ia-ta [ { parameters } ] [ exec "exec string" ]` **Parameters** option **ia-ta** takes the following parameters: ia-id *value* This parameter specifies the user's preferred values and is optional. The *value* specified can be a decimal number or a hex number prefixed with '0x'

Keyword	Purpose, format, and parameters
request-option	**Purpose** Specifies option 6. If specified, the client requests a list of options from the server. **Format** `option request-option { parameters } [ exec "exec string" ]` **Parameters** option **request-option** takes a space separated list of option codes (in decimal) as the argument
rapid-commit	**Purpose** Specifies option 14. If specified, the client indicates that the client is prepared to perform the Solicit-Reply message exchange. **Format** `option rapid-commit [exec "exec string"]` **Parameters** Does not take any parameters other than the optional exec statement
user-class	**Purpose** specifies option 15. If specified, the client indicates the type or category of user or applications it represents. **Format** `option user-class { parameters } [ exec "exec string" ]` **Parameters** Option **user-class** takes one or more instances of user class data. Each instance of user class data is a quoted or unquoted string of arbitrary length. If a string contains a blank space, it must be enclosed in quotes. The parameters are required. The format for the parameter is: class *value* class *value* where *value* is a quoted or unquoted string.
vendor-class	**Purpose** Specifies option 16. If specified, the client indicates the vendor that manufactured the hardware on which the client is running. **Format** option vendor-class { *parameters* } [exec "exec string"] **Parameters** Option **vendor-class** takes the vendor's registered Enterprise number and one or more instances of vendor class data. Each instance of vendor class data is a quoted or unquoted string of arbitrary length, each of which describes some characteristic of the client's hardware configuration. The parameters are *not* optional. The format is: vendor-id *value* class *value* class *value* where *value* is a quoted or unquoted string.
vendor-opts	**Purpose** Specifies option 17. If specified, the client indicates the vendor-specific information to the server. **Format** `option vendor-opts <enterprise-number> { parameters } [ exec "exec string" ]]` **Parameters** Option **vendor-opts** takes the vendor's registered Enterprise number and one or more instances of vendor option data. Each instance of vendor option data is a vendor option code followed by a option data in string or hex format. The parameters are *not* optional. The format is: vendor-id *value* option *opcode option-data* option *opcode option-data* where *option-data* is a quoted or non-quoted or a hex string (prefixed with '0x')

Keyword	Purpose, format, and parameters
reconf-accept	**Purpose** Specifies option 20. If specified, the client indicates to the server whether the client is willing to accept reconfigure message from the server. **Format** `option reconf-accept [ { exec "exec string" } ]` **Parameters** option **reconf-accept** takes no option specific parameters, other than the exec statement.
dns-servers	**Purpose** specifies option 23. If specified, the client indicates to the server the preferred set of DNS servers **Format** `option dns-servers [ { parameters } ] [ exec "exec string" ]` **Parameters** option **dns-servers** takes a space/line separated list of IPv6 addresses as argument.
domain-list	**Purpose** specifies option 24. If specified, the client indicates the preferred domain list. **Format** `option domain-list [ { parameters } ] [ exec "exec string" ]` **Parameters** option **domain-list** takes a space/line separated list of domain name strings.

Interface keywords:

The interface keyword is in the format `interface <interface name> [ { option declaration/s } ]`.

Table 74. Keyword and description for the interface keywords.

Keywords	Descriptions
`interface <interface name> [ { option declaration/s } ]`	The interface statement takes one or more option declarations as arguments. These options, specified within interface stanza are specific to this interface, unlike the options declared outside of the interface stanza, which apply to all the interfaces.

```
interface  en1  {
            option ia-na {
           ia-id  01
           renew-time 0x40
           rebind-time 0x60
 }

 option  request-option  { 3 23 24 }

       option  user-class {
          class ibm
               class "userclassA and B"
               class "userclassB"
       }

      option  vendor-class {
            vendor-id 1234
               class "vendorclassA"
               class "vendorclassB"
      }

      option  vendor-opts {
          vendor-id 2343
               option 89       vendoroption89
               option 90       vendoroption90
      }

 option reconf-accept
```

DHCP relay agent

The `/etc/dhcprd.cnf` file is the configuration file for the **DHCP** and **BOOTP** relay agent. The format of the file and the allowed directives and keywords are explained here.

The directives are specified in the following format:

<keyword> <value1> ... <valueN>

The presence and values of these parameters are used by the relay agent that is started or restarted.

This set of parameters specifies the log files that will be maintained by this server. Each parameter is identified by a keyword and followed by its value.

Keyword	Value	Definition
numLogFiles	0 to *n*	Number of log files. If 0 is specified, no log file will be maintained and no log message is display anywhere. *n* is the maximum number of log files maintained as the size of the most recent log file reaches its maximum size and a new log file is created.
logFileSize	In KB	Maximum size of a log file. When the size of the most recent log file reaches this value, it is renamed and a new log file is created.
logFileName	file path	Name of the most recent log file. Less recent log files have the number 1 to (*n* - 1) appended to their names; the larger the number, the less recent the file.
logItem	One item that will be logged.	**SYSERR** System error, at the interface to the platform. **OBJERR** Object error, in between objects in the process. **PROTERR** Protocol error, between client and server. **WARNING** Warning, worth of attention from the user. **EVENT** Event occurred to the process. **ACTION** Action taken by the process. **INFO** Information that might be useful. **ACNTING** Who was served when. **TRACE** Code flow, for debugging.

For example, an `/etc/dhcprd.cnf` file may have the following entries:

```
numLogFiles  4
logFileSize  1000
logFileName    /usr/tmp/dhcprd.log
logItem     SYSERR
logItem     OBJERR
logItem     PROTERR
```

```
logItem     WARNING
logItem     EVENT
logItem     ACTION
logItem     INFO
logItem     ACNTING
logItem     TRACE
```

Keyword	Value	Definition
relay	**IPv4**, **IPv6**, or **ALL**	Specifies the mode of packet relay. If **IPv4** is specified, the relay agent acts only as **DHCPv4** relay agent. This is the default mode of the relay agent. If **IPv6** is specified, the relay agent acts only as **DHCPv6** relay agent. If **ALL** is specified, the relay agent acts both as **DHCPv4** and **DHCPv6** relay agent.
server	IP address	Specifies the IP address of a **BOOTP** or **DHCP** server. The packet will be forwarded to the servers listed in this file.
server6	IPv6 address	Specifies the IPv6 address of the **DHCPv6** server. The packet will be forwarded to the servers listed here.
option6	*<option code> <option data>*	Specifies the **DHCPv6** relay agent options. The keyword is valid only if the relay mode is set to IPv6. The *option code* value is specified as a decimal number. The *option data* value is specified as quoted or non-quoted string or in hexadecimal format (prefixed with `0x`)
single-site		Specifies that the device on which the relay agent is running belongs to only one site.

Preboot Execution Environment Proxy DHCP daemon

The PXE Proxy **DHCP** server behaves much like a **DHCP** server by listening for ordinary **DHCP** client traffic and responding to certain client requests. However, unlike the **DHCP** server, the PXE Proxy **DHCP** server does not administer network addresses, and it only responds to clients that identify themselves as PXE clients.

The responses given by the PXE Proxy **DHCP** server contain the mechanism by which the client locates the boot servers or the network addresses and descriptions of the supported, compatible boot servers.

Using a PXE Proxy **DHCP** server in addition to a **DHCP** server provides three key features. First, you can separate the administration of network addresses from the administration of boot images. Using two different processes on the same system, you can configure the boot information managed by the PXE Proxy **DHCP** server without disturbing or requiring access to the **DHCP** server configuration. Second, you can define multiple boot servers and let the PXE client select a particular server during boot time. Each boot server can, for example, offer a different type of operating system or system configuration. Finally, using the proxy server offers the ability to configure the PXE client to use multicast IP addressing to discover the location of the compatible boot servers.

The PXE Proxy **DHCP** server can be configured to run on the same system that is running the **DHCP** server or on a different system. Also, it can be configured to run on the same system that is also running the boot server daemon or on a different system.

PXE Proxy DHCP server components

There are three components of the PXED server.

The PXED server is segmented into three main pieces, a database, a protocol engine, and a set of service threads, each with its own configuration information.

PXED database:

The `db_file.dhcpo` database is used to generate the options to be sent to the client when the client send an REQUEST packet.

The options returned by the database depend on the type of server chosen. This is set using the keyword **pxeservertype** in the `pxed.cnf` file.

Using the information in the configuration file, the database is primed and verified for consistency.

PXED protocol engine:

The protocol engine uses the database to determine what information should be returned to the client.

The PXED protocol engine is based on Intel Preboot Execution Environment (PXE) Specification Version 2.1 and is still compatible with Intel PXE Specification Version 1.1.

PXED threaded operations:

The last piece of the PXED server is actually a set of operations that are used to keep things running. Since the PXED server is threaded, these operations are actually set up as threads that occasionally do things to make sure everything is together.

The first thread, the *main* thread, handles the SRC requests (such as **startsrc**, **stopsrc**, **lssrc**, **traceson**, and **refresh**). This thread also coordinates all operations that affect all threads and handles signals. For example,

- A SIGHUP (-1) causes a refresh of all databases in the configuration file.
- A SIGTERM (-15) causes the server to gracefully stop.

The other thread processes packets. Depending on the server type, there can one or two threads. One thread listens on port 67 and the second one listens to port 4011. Each of these can handle a request from a client.

PXED server configuration

By default, the PXED server is configured by reading the `/etc/pxed.cnf` file, which specifies the server's initial database of options and addresses.

The server is started from SMIT, or through SRC commands.

Configuring the PXED server is usually the hardest part of using PXED in your network. First, figure out what networks you need to have PXE clients on. The following example configures the **pxed** daemon to run on the same machine as the DHCP server:

```
pxeservertype       proxy_on_dhcp_server

subnet default
{
    vendor pxe
    {
        option    6      2       # Disable Multicast boot server discovery
        option    8      1   2     9.3.4.5    9.3.4.6   2    1    9.3.149.29
                                 # The above option gives the list of bootservers
```

```
        option    9      0       "PXE bootstrap server" \
                         1       "Microsoft Windows NT Boot Server"  \
                         2       "DOS/UNDI Boot Server"
        option    10    20    "seconds left before the first item in the boot menu is auto-selected"
     }
}
```

The suboptions in the vendor container are sent to PXE clients only if the client's IP address is in the subnet's IP address range (for example, 9.3.149.0 through 9.3.149.255).

The following example configures the **pxed** daemon to run on a different machine than the **DHCP** server:

```
subnet default
{
    vendor pxe
    {
        option    6      10      # The bootfile name is present in the client's initial pxed
                                 # offer packet.
        option    8      1   2     9.3.4.5    9.3.4.6   2    1    9.3.149.29
                                 # The above option gives the list of bootservers
        option    9      0       "PXE bootstrap server" \
                         1       "Microsoft Windows NT Boot Server"  \
                         2       "DOS/UNDI Boot Server"
        option    10    20    "seconds left before the first item in the boot menu is auto-selected"
        bootstrapserver          9.3.148.65
        pxebootfile    1    2    1   window.one
        pxebootfile    2    2    1   linux.one
        pxebootfile    1    2    1   hello.one
        client 6 10005a8ad14d any
        {
           pxebootfile    1    2    1   aix.one
           pxebootfile    2    2    1   window.one
        }
    }

Vendor pxeserver
{
    option     7     224.234.202.202
}
```

The **pxeservertype** keyword is not set in the configuration file so the default value is taken, which is **pdhcp_only**, meaning the PXED server is running on a different machine than the **DHCP** server. Given this configuration, the PXED server listens on two ports (67 and 4011) for clients' BINLD REQUEST/INFORM packets. The option 7 is sent to the BINLD server when the PXED server receives a REQUEST/INFORM packet on port 67 from BINLD and option 60 is set to PXED server.

The `db_file` database clause indicates which database method to use for processing this part of the configuration file. Comments begin with a pound sign (#). From the # to the end of the line are ignored by the PXED server. Each `option` line is used by the server to tell the client what to do. “PXE vendor container suboptions” on page 288 describes the currently supported and known options. See “PXED server file syntax for general server operation” on page 289 for ways to specify options that the server does not know about.

PXED configuration file:

The configuration file has an address section and an option definition section, which are based on the concept of containers that hold options, modifiers, and, potentially, other containers.

A *container* (basically, a method to group options) uses an identifier to classify clients into groups. The container types are subnet, class, vendor, and client. Currently, there is not a generic user-definable container. The identifier uniquely defines the client so that the client can be tracked if, for example, it moves between subnets. More than one container type can be used to define client access.

Options are identifiers that are returned to the client, such as default gateway and DNS address.

PXED containers:

When the **DHCP** server receives a request, the packet is parsed and identifying keys determine which containers, options, and addresses are extracted.

The example in PXED server configuration shows a subnet container. Its identifying key is the client's position in the network. If the client is from that network, then it falls into that container.

Each type of container uses a different option to identify a client:

- The subnet container uses the `giaddr` field or the interface address of the receiving interface to determine which subnet the client came from.
- The class container uses the value in option 77 (User Site Class Identifier).
- The vendor uses the value in option 60 (Vendor Class Identifier).
- The client container uses the option 61 (Client Identifier) for PXE clients and the `chaddr` field in the **BOOTP** packet for **BOOTP** clients.

Except for subnets, each container allows the specification of the value that it will match including regular expression matching.

There is also an implicit container, the *global* container. Options and modifiers in the global container apply to all containers unless overridden or denied. Most containers can be placed inside other containers implying a scope of visibility. Containers might or might not have address ranges associated with them. Subnets, by their nature, have ranges associated with them.

The basic rules for containers and subcontainers are as follows:

- All containers are valid at the global level.
- Subnets can never be placed inside other containers.
- Restricted containers cannot have regular containers of the same type within them. (For example, a container with an option that only allows a class of `Accounting` cannot include a container with an option that allows all classes that start with the letter "a." This is illegal.)
- Restricted client containers cannot have subcontainers.

Given the above rules, you can generate a hierarchy of containers that segment your options into groups for specific clients or sets of clients.

If a client matches multiple containers, how are options and addresses handed out? The **DHCP** server receives messages, it passes the request to the database (`db_file` in this case), and a container list is generated. The list is presented in order of depth and priority. Priority is defined as an implicit hierarchy in the containers. Strict containers are higher priority than regular containers. Clients, classes, vendors, and finally subnets are sorted, in that order, and within container type by depth. This generates a list ordered by most specific to least specific. For example:

```
Subnet 1
--Class 1
--Client 1
Subnet 2
--Class 1
----Vendor 1
----Client 1
--Client 1
```

The above example shows two subnets, `Subnet 1` and `Subnet 2`. There is one class name, `Class 1`, one vendor name, `Vendor 1`, and one client name, `Client 1`. `Class 1` and `Client 1` are defined in multiple places. Because they are in different containers, their names can be the same but values inside them can

be different. If `Client 1` sends a message to the **DHCP** server from `Subnet 1` with `Class 1` specified in its option list, the **DHCP** server would generate the following container path:

`Subnet 1, Class 1, Client 1`

The most specific container is listed last. To get an address, the list is examined in reverse hierarchy to find the first available address. Then, the list is examined in forward hierarchy to get the options. Options override previous values unless an option **deny** is present in the container. Also, since `Class 1` and `Client 1` are in `Subnet 1`, they are ordered according to the container priority. If the same client is in `Subnet 2` and sends the same message, the container list generated is:

`Subnet 2, Class 1, Client 1` (at the `Subnet 2` level), `Client 1` (at the `Class 1` level)

`Subnet 2` is listed first, then `Class 1`, then the `Client 1` at the `Subnet 2` level (because this client statement is only one level down in the hierarchy). The hierarchy implies that a client matching the first client statement is less specific than the client matching `Client 1` of `Class 1` within `Subnet 2`.

Priority selected by depth within the hierarchy is not superseded by the priority of the containers themselves. For example, if the same client issues the same message and specifies a vendor identifier, the container list is:

`Subnet 2, Class 1, Vendor 1, Client 1` (at `Subnet 2` level), `Client 1` (at `Class 1` level)

Container priority improves search performance because it follows a general concept that client containers are the most specific way to define one or more clients. The class container holds less specific addresses than a client container; vendor is even less specific; and subnet is the least specific.

PXED addresses and address ranges:

Any container type can have associated addresses ranges; subnets must have them. Each range within a container must be a subset of the parent container's range and must not overlap with other containers' ranges.

For example, if a class is defined within a subnet and the class has a range, the range must be a subset of the subnet's range. Also, the range within that class container cannot overlap with any other ranges at its level.

Ranges can be expressed on the container line and modified by range and exclude statements to allow for disjoint address sets associated with a container. So, if you have the top ten addresses and the second ten addresses of a subnet available, the subnet could specify these addresses by range in the subnet clause to reduce both memory use and the chance of address collision with other clients not in the specified ranges.

After an address has been selected, any subsequent container in the list that contains address ranges is removed from the list along with its children. The reason for this is that network-specific options in removed containers are not valid if an address is not used from within that container.

PXED configuration file options:

After the list has been culled to determine addresses, a set of options is generated for the client.

In this selection process, options overwrite previously selected options unless a *deny* is encountered, in which case, the denied option is removed from the list being sent to the client. This method allows inheritance from parent containers to reduce the amount of data that must be specified.

PXED logging:

Logging parameters are specified in a container like the database, but the container keyword is **logging_info**.

When learning to configure PXED, it is advisable to turn logging to its highest level. Also, it is best to specify the logging configuration prior to any other configuration file data to ensure that configuration errors are logged after the logging subsystem is initialized. Use the **logitem** keyword to turn on a logging level or remove the **logitem** keyword to disable a logging level. Other keywords for logging allow the specification of the log filename, file size, and the number of rotating log files.

PXED performance considerations:

It is important to understand that certain configuration keywords and the structure of the configuration file have an effect on the memory use and performance of the PXED server.

First, excessive memory use can be avoided by understanding the inheritance model of options from parent to child containers. In an environment that supports no unlisted clients, the administrator must explicitly list each client in the file. When options are listed for any specific client, the server uses more memory storing that configuration tree than when options are inherited from a parent container (for example, the subnet, network, or global containers). Therefore, the administrator should verify whether any options are repeated at the client level within the configuration file and, if so, determine whether these options can be specified in the parent container and shared by the set of clients as a whole.

Also, when using the **logItem** entries INFO and TRACE, numerous messages are logged during the processing of every PXE client's message. Appending a line to the log file can be an expensive operation; therefore, limiting the amount of logging improves the performance of the PXED server. When an error with the PXED server is suspected, logging can be dynamically re-enabled using the SRC **traceson** command.

PXE vendor container suboptions

When supporting a PXE client, the **DHCP** server passes the following option to the BINLD server that BINLD uses to configure itself:

Opt Num	Default Data Type	Can Specify?	Description
6	Decimal number	Yes	PXE_DISCOVERY_CONTROL. Limit 0-16. This is a bit field. Bit 0 is the least significant bit. **bit 0** If set, disables broadcast discovery. **bit 1** If set, disables multicast discovery. **bit 2** If set, only uses/accepts servers in PXE_BOOT_ SERVERS. **bit 3** If set, and a bootfile name is present in the initial PXED offer packet, downloads the bootfile (does not prompt/menu/discover boot server). **bit 4-7** Must be 0. If this option is not supplied then client assumes all bits to be equal to 0.
7	One dotted quad	Yes	Multicast IP address. Boot server discovery multicast IP address. Boot servers capable of multicast discovery must listen on this multicast address. This option is required if the multicast discovery disable bit (bit 1) in the PXE_DISCOVERY_ CONTROL option is not set.

Opt Num	Default Data Type	Can Specify?	Description
8	*Boot server type(0-65535)*	Yes	PXE_BOOT_SERVERS *IP address count (0-256)* **Type 0** Miscrosoft Windows *IP address...IP address* NT Boot Server *Boot server type IP address* **Type 1** Intel LCM Boot Server *count IP address ...* **Type 3** DOS/UNDI Boot Server *IP address* **Type 4** NEC ESMPRO Boot Server **Type 5** IBM WSoD Boot Server **Type 6** IBM LCCM Boot Server **Type 7** CA Unicenter TNG Boot Server. **Type 8** HP OpenView Boot Server. **Type 9 through 32767** Reserved **Type 32768 through 65534** Vendor use **Type 65535** PXE API Test Server. If *IP address count* is zero for a server type then the client may accept offers from any boot server of that type. Boot Servers do not respond to discovery requests of types they do not support.
9	*Boot server type (0-65535)*	Yes	PXE_BOOT_MENU *"description"* Boot server boot "order" is implicit in the type. *"description"...menu order*.
10	*Timeout in seconds (0-255)*	Yes	PXE_MENU_PROMPT *"prompt"* The timeout is the number of seconds to wait before auto- selecting the first boot menu item. On the client system, the prompt is displayed followed by the number of seconds remaining before the first item in the boot menu is auto-selected. If the F8 key is pressed on the client system, then a menu is displayed. If this option is provided to the client, then the menu is displayed without prompt and timeout. If the timeout is 0, then the first item in the menu is auto-selected. If the timeout is 255, the menu and prompt is displayed without auto-selecting or timeout.

PXED server file syntax for general server operation

The PXED server file keywords of the **DHCPv6** server described here are for general server operation. Their forms, subcontainers, default values, and meanings are identified.

Note: Time Units (*time_units*) shown in the following table are optional and represent a modifier to the actual time. The default time unit is minutes. Valid values are seconds (1), minutes (60), hours (3600), days (86400), weeks (604800), months (2392000), and years (31536000). The number shown in parentheses is a multiplier applied to the specified value *n* to express the value in seconds.

Keyword	Form	Subcontainers?	Default Value	Meaning
database	database *db type*	Yes	None	The primary container that holds the definitions for the address pools, options, and client access statements. *db type* is the name of a module that is loaded to process this part of the file. The only value currently available is **db_file**.
logging_info	logging_info	Yes	None	The primary logging container that defines the logging parameters.
logitem	logitem NONE	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed
logitem	logitem SYSERR	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed
logitem	logitem OBJERR	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed
logitem	logitem PROTOCOL	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed
logitem	logitem PROTERR	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed
logitem	logitem WARN	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed
logitem	logitem WARNING	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed
logitem	logitem CONFIG	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed
logitem	logitem EVENT	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed
logitem	logitem PARSEERR	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed
logitem	logitem ACTION	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed
logitem	logitem ACNTING	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed
logitem	logitem STAT	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed
logitem	logitem TRACE	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed
logitem	logitem RTRACE	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed
logitem	logitem START	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed
numLogFiles	numLogFiles *n*	No	0	Specifies the number of log files to create. The log rotates when the first one fills. *n* is the number of files to create.
logFileSize	logFileSize *n*	No	0	Specifies the size of each log file in 1024-byte units.

Keyword	Form	Subcontainers?	Default Value	Meaning
logFileName	logFileName *path*	No	None	Specifies the path to the first log file. The original log file is named *filename* or *filename.extension*. The *filename* must be eight or fewer characters. When a file is rotated, it is renamed beginning with the base *filename*, then either appending a number or replacing the extension with a number. For example, if the original file name is `file`, the rotated file name becomes `file01`. If the original file name is `file.log`, it becomes `file.01`.
pxeservertype	pxeservertype *servertype*	No	dhcp_only	Indicates the type of **dhcpsd** server it is. *servertype* can be **proxy_on_dhcp_server**, which means that PXED is running on the same machine as the **DHCP** server and it is listening for PXE client requests on port 4011 only, or the default value of **pdhcp_only**, which means the PXED is running on a separate machine and it has to listen for client packets on port 67 and 4011.

PXED server file syntax for db_file database

The PXED server file syntax for the `db_file` database is described here. Forms, subcontainers, default values, and meanings are identified.

Note:

1. Time Units (*time_units*) shown in the following table are optional and represent a modifier to the actual time. The default time unit is minutes. Valid values are seconds (1), minutes (60), hours (3600), days (86400), weeks (604800), months (2392000), and years (31536000). The number shown in parentheses is a multiplier applied to the specified value *n* to express the value in seconds.
2. Items that are specified in one container can be overridden inside a subcontainer. For example, you could globally define **BOOTP** clients, but within a certain subnet allow **BOOTP** clients by specifying the supportBootp keywork in both containers.
3. The client, class, and vendor containers allow for regular expression support. For class and vendor, a quoted string with the first character after the quote being an exclamation point (!) indicates that the rest of the string should be treated as a regular expression. The client container allows for regular expressions on both the **hwtype** and the **hwaddr** fields. A single string is used to represent both fields with the following format:

   ```
   decimal_number-data
   ```

   If decimal_number is zero, then data is an ASCII string. If any other number, data is hex digits.

Keyword	Form	Subcontainers?	Default Value	Meaning
subnet	subnet default	Yes	None	Specifies a subnet that does not have any range. The subnet is used by the server only when it is responding to INFORM packet from the client.
subnet	subnet *subnet id netmask*			Specifies a subnet and a pool of addresses. All addresses are assumed to be in the pool unless a range is specified on the line or addresses are modified later in the container by a range or exclude statement. The optional range is a pair of IP addresses in dotted quad format separated by a dash. An optional label and priority can be specified. These are used by virtual subnets to identify and order the subnets in the virtual subnet. The label and priority are separated by a colon. These containers are only allowed at the global or database container level.
subnet	subnet *subnet id netmask range*			Specifies a subnet and a pool of addresses. All addresses are assumed to be in the pool unless a range is specified on the line or addresses are modified later in the container by a range or exclude statement. The optional range is a pair of IP addresses in dotted quad format separated by a dash. An optional label and priority can be specified. These are used by virtual subnets to identify and order the subnets in the virtual subnet. The label and priority are separated by a colon. These containers are only allowed at the global or database container level.

Keyword	Form	Subcontainers?	Default Value	Meaning
subnet	subnet *subnet id netmask label*:*priority*			Specifies a subnet and a pool of addresses. All addresses are assumed to be in the pool unless a range is specified on the line or addresses are modified later in the container by a range or exclude statement. The optional range is a pair of IP addresses in dotted quad format separated by a dash. An optional label and priority can be specified. These are used by virtual subnets to identify and order the subnets in the virtual subnet. The label and priority are separated by a colon. These containers are only allowed at the global or database container level.
subnet	subnet *subnet id netmask range label*:*priority*			Specifies a subnet and a pool of addresses. All addresses are assumed to be in the pool unless a range is specified on the line or addresses are modified later in the container by a range or exclude statement. The optional range is a pair of IP addresses in dotted quad format separated by a dash. An optional label and priority can be specified. These are used by virtual subnets to identify and order the subnets in the virtual subnet. The label and priority are separated by a colon. These containers are only allowed at the global or database container level.
subnet	subnet *subnet id range*	Yes	None	Specifies a subnet that goes within a network container. It defines a range of addresses that is the whole subnet unless the optional range part is specified. The netmask associated with the subnet is taken from the surrounding network container. **Note:** This method is deprecated in favor of the other subnet forms.

Keyword	Form	Subcontainers?	Default Value	Meaning
option	option *number data ...*	No	None	Specifies an option to send to a client or, in the case of deny, an option to prevent from being sent to the client. The optional `* deny` clause means all options not specified in the current container are not to be returned to the client. option *number*deny only denies the specified option. *number* is an unsigned 8-bit integer. *data* is specific to the option (see above) or can be specified as a quoted string (indicating ASCII text) or 0x*hexdigits* or hex"*hexdigits*" or hex "*hexdigits*". If the option is in a vendor container, the option will be encapsulated with other options in an option 43.
option	option *number*deny	No	None	Specifies an option to send to a client or, in the case of deny, an option to prevent from being sent to the client. The optional `* deny` clause means all options not specified in the current container are not to be returned to the client. option *number*deny only denies the specified option. *number* is an unsigned 8-bit integer. *data* is specific to the option (see above) or can be specified as a quoted string (indicating ASCII text) or 0x*hexdigits* or hex"*hexdigits*" or hex "*hexdigits*". If the option is in a vendor container, the option will be encapsulated with other options in an option 43.
option	option * deny	No	None	Specifies an option to send to a client or, in the case of deny, an option to prevent from being sent to the client. The optional `* deny` clause means all options not specified in the current container are not to be returned to the client. option *number*deny only denies the specified option. *number* is an unsigned 8-bit integer. *data* is specific to the option (see above) or can be specified as a quoted string (indicating ASCII text) or 0x*hexdigits* or hex"*hexdigits*" or hex "*hexdigits*". If the option is in a vendor container, the option will be encapsulated with other options in an option 43.

Keyword	Form	Subcontainers?	Default Value	Meaning
exclude	exclude *an IP address*	No	None	Modifies the range on the container in which the exclude statement is in. The exclude statement is not valid in the global or database container levels. The exclude statement removes the specified address or range from the current range on the container. The exclude statement allows you to create noncontiguous ranges for subnets or other containers.
exclude	exclude *dotted_quad-dotted_quad*	No	None	Modifies the range on the container in which the exclude statement is in. The exclude statement is not valid in the global or database container levels. The exclude statement removes the specified address or range from the current range on the container. The exclude statement allows you to create noncontiguous ranges for subnets or other containers.
range	range *IP_address*	No	None	Modifies the range on the container in which the range statement is in. The range statement is not valid in the global or database container levels. If the range is the first in the container that does not specify a range on the container definition line, then the range for the container becomes the range specified by the range statement. Any range statement after the first range or all range statements for a containers that specifies ranges in its definition are added to the current range. With the range statement, a single address or set of addresses can be added to the range. The range must fit inside the subnet container definition.

Keyword	Form	Subcontainers?	Default Value	Meaning
range	range *dotted_quad-dotted_quad*	No	None	Modifies the range on the container in which the range statement is in. The range statement is not valid in the global or database container levels. If the range is the first in the container that does not specify a range on the container definition line, then the range for the container becomes the range specified by the range statement. Any range statement after the first range or all range statements for a containers that specifies ranges in its definition are added to the current range. With the range statement, a single address or set of addresses can be added to the range. The range must fit inside the subnet container definition.
client	client *hwtype hwaddr* NONE	Yes	None	Specifies a client container that denies the client specified by the *hwaddr* and *hwtype* from getting an address. If *hwtype* is 0, then *hwaddr* is an ASCII string. Otherwise, *hwtype* is the hardware type for the client and *hwaddr* is the hardware address of the client. If the *hwaddr* is a string, then quotes are accepted around the string. If the *hwaddr* is a hexstring, then the address may be specified by 0x*hexdigits* or hex *digits*. *range* allows the client specified by the *hwaddr* and *hwtype* to get an address in the *range*. Must be regular expressions to match multiple clients.
client	client *hwtype hwaddr* ANY	Yes	None	Specifies a client container that denies the client specified by the *hwaddr* and *hwtype* from getting an address. If *hwtype* is 0, then *hwaddr* is an ASCII string. Otherwise, *hwtype* is the hardware type for the client and *hwaddr* is the hardware address of the client. If the *hwaddr* is a string, then quotes are accepted around the string. If the *hwaddr* is a hexstring, then the address may be specified by 0x*hexdigits* or hex *digits*. *range* allows the client specified by the *hwaddr* and *hwtype* to get an address in the *range*. Must be regular expressions to match multiple clients.

Keyword	Form	Subcontainers?	Default Value	Meaning
client	client *hwtype hwaddr dotted_quad*	Yes	None	Specifies a client container that denies the client specified by the *hwaddr* and *hwtype* from getting an address. If *hwtype* is 0, then *hwaddr* is an ASCII string. Otherwise, *hwtype* is the hardware type for the client and *hwaddr* is the hardware address of the client. If the *hwaddr* is a string, then quotes are accepted around the string. If the *hwaddr* is a hexstring, then the address may be specified by 0x*hexdigits* or hex *digits*. *range* allows the client specified by the *hwaddr* and *hwtype* to get an address in the *range*. Must be regular expressions to match multiple clients.
client	client *hwtype hwaddr range*	Yes	None	Specifies a client container that denies the client specified by the *hwaddr* and *hwtype* from getting an address. If *hwtype* is 0, then *hwaddr* is an ASCII string. Otherwise, *hwtype* is the hardware type for the client and *hwaddr* is the hardware address of the client. If the *hwaddr* is a string, then quotes are accepted around the string. If the *hwaddr* is a hexstring, then the address may be specified by 0x*hexdigits* or hex *digits*. *range* allows the client specified by the *hwaddr* and *hwtype* to get an address in the *range*. Must be regular expressions to match multiple clients.
class	class *string*	Yes	None	Specifies a class container with name *string*. String can be quoted or not. If quoted, the quotes are removed before comparison. Quotes are required for strings with spaces or tabs. This container is valid at any level. A range can be supplied to indicate a set of addresses to hand out to a client with this class. The range is either a single dotted quad IP address or two dotted quad IP addresses separated by a dash.

Keyword	Form	Subcontainers?	Default Value	Meaning
class	class *string range*	Yes	None	Specifies a class container with name *string*. String can be quoted or not. If quoted, the quotes are removed before comparison. Quotes are required for strings with spaces or tabs. This container is valid at any level. A range can be supplied to indicate a set of addresses to hand out to a client with this class. The range is either a single dotted quad IP address or two dotted quad IP addresses separated by a dash.
network	network *network id netmask*	Yes	None	Specifies a network ID using class information (for example, 9.3.149.0 with a netmask of 255.255.255.0 would be network 9.0.0.0 255.255.255.0). This version of the network container is used to hold subnets with the same network ID and netmask. When a range is provided, all the addresses in the range are in the pool. The range must be in the network ID's network. This uses class full addressing. This is only valid in the global or database container level. **Note:** The network keyword is deprecated in favor of the subnet container.
network	network *network id*	Yes	None	Specifies a network ID using class information (for example, 9.3.149.0 with a netmask of 255.255.255.0 would be network 9.0.0.0 255.255.255.0). This version of the network container is used to hold subnets with the same network ID and netmask. When a range is provided, all the addresses in the range are in the pool. The range must be in the network ID's network. This uses class full addressing. This is only valid in the global or database container level. **Note:** The network keyword is deprecated in favor of the subnet container.

Keyword	Form	Subcontainers?	Default Value	Meaning
network	network *network id range*	Yes	None	Specifies a network ID using class information (for example, 9.3.149.0 with a netmask of 255.255.255.0 would be network 9.0.0.0 255.255.255.0). This version of the network container is used to hold subnets with the same network ID and netmask. When a range is provided, all the addresses in the range are in the pool. The range must be in the network ID's network. This uses class full addressing. This is only valid in the global or database container level. **Note:** The network keyword is deprecated in favor of the subnet container.
vendor	vendor *vendor_id*	Yes	None	Specifies a vendor container. Vendor containers are used to return option 43 to the client. The vendor id may be specified in a quoted string or a binary string in the form 0x*hexdigits* or hex"*digits*". An optional range may be placed after the vendor id. The range is specified as two dotted quads separated by a dash. After the optional range, an optional hexstring or ASCII string can be specified as the first part of the option 43. If options are in the container, they are appended to the option 43 data. After all options are processed an End Of Option List Option is appended to the data. To return options outside of an option 43, use a regular expression client that matches all clients to specify normal options to return based on the vendor ID.

Keyword	Form	Subcontainers?	Default Value	Meaning
vendor	vendor *vendor_id* hex""			Specifies a vendor container. Vendor containers are used to return option 43 to the client. The vendor id may be specified in a quoted string or a binary string in the form 0x*hexdigits* or hex"*digits*". An optional range may be placed after the vendor id. The range is specified as two dotted quads separated by a dash. After the optional range, an optional hexstring or ASCII string can be specified as the first part of the option 43. If options are in the container, they are appended to the option 43 data. After all options are processed an End Of Option List Option is appended to the data. To return options outside of an option 43, use a regular expression client that matches all clients to specify normal options to return based on the vendor ID.
vendor	vendor *vendor_id* hex ""			Specifies a vendor container. Vendor containers are used to return option 43 to the client. The vendor id may be specified in a quoted string or a binary string in the form 0x*hexdigits* or hex"*digits*". An optional range may be placed after the vendor id. The range is specified as two dotted quads separated by a dash. After the optional range, an optional hexstring or ASCII string can be specified as the first part of the option 43. If options are in the container, they are appended to the option 43 data. After all options are processed an End Of Option List Option is appended to the data. To return options outside of an option 43, use a regular expression client that matches all clients to specify normal options to return based on the vendor ID.

Keyword	Form	Subcontainers?	Default Value	Meaning
vendor	vendor *vendor_id* 0xdata			Specifies a vendor container. Vendor containers are used to return option 43 to the client. The vendor id may be specified in a quoted string or a binary string in the form 0x*hexdigits* or hex"*digits*". An optional range may be placed after the vendor id. The range is specified as two dotted quads separated by a dash. After the optional range, an optional hexstring or ASCII string can be specified as the first part of the option 43. If options are in the container, they are appended to the option 43 data. After all options are processed an End Of Option List Option is appended to the data. To return options outside of an option 43, use a regular expression client that matches all clients to specify normal options to return based on the vendor ID.
vendor	vendor *vendor_id* ""			Specifies a vendor container. Vendor containers are used to return option 43 to the client. The vendor id may be specified in a quoted string or a binary string in the form 0x*hexdigits* or hex"*digits*". An optional range may be placed after the vendor id. The range is specified as two dotted quads separated by a dash. After the optional range, an optional hexstring or ASCII string can be specified as the first part of the option 43. If options are in the container, they are appended to the option 43 data. After all options are processed an End Of Option List Option is appended to the data. To return options outside of an option 43, use a regular expression client that matches all clients to specify normal options to return based on the vendor ID.

Keyword	Form	Subcontainers?	Default Value	Meaning
vendor	vendor *vendor_id range*			Specifies a vendor container. Vendor containers are used to return option 43 to the client. The vendor id may be specified in a quoted string or a binary string in the form 0x*hexdigits* or hex"*digits*". An optional range may be placed after the vendor id. The range is specified as two dotted quads separated by a dash. After the optional range, an optional hexstring or ASCII string can be specified as the first part of the option 43. If options are in the container, they are appended to the option 43 data. After all options are processed an End Of Option List Option is appended to the data. To return options outside of an option 43, use a regular expression client that matches all clients to specify normal options to return based on the vendor ID.
vendor	vendor *vendor_id range* hex""			Specifies a vendor container. Vendor containers are used to return option 43 to the client. The vendor id may be specified in a quoted string or a binary string in the form 0x*hexdigits* or hex"*digits*". An optional range may be placed after the vendor id. The range is specified as two dotted quads separated by a dash. After the optional range, an optional hexstring or ASCII string can be specified as the first part of the option 43. If options are in the container, they are appended to the option 43 data. After all options are processed an End Of Option List Option is appended to the data. To return options outside of an option 43, use a regular expression client that matches all clients to specify normal options to return based on the vendor ID.

Keyword	Form	Subcontainers?	Default Value	Meaning
vendor	vendor *vendor_id range* hex ""			Specifies a vendor container. Vendor containers are used to return option 43 to the client. The vendor id may be specified in a quoted string or a binary string in the form 0x*hexdigits* or hex"*digits*". An optional range may be placed after the vendor id. The range is specified as two dotted quads separated by a dash. After the optional range, an optional hexstring or ASCII string can be specified as the first part of the option 43. If options are in the container, they are appended to the option 43 data. After all options are processed an End Of Option List Option is appended to the data. To return options outside of an option 43, use a regular expression client that matches all clients to specify normal options to return based on the vendor ID.
vendor	vendor *vendor_id range* 0xdata			Specifies a vendor container. Vendor containers are used to return option 43 to the client. The vendor id may be specified in a quoted string or a binary string in the form 0x*hexdigits* or hex"*digits*". An optional range may be placed after the vendor id. The range is specified as two dotted quads separated by a dash. After the optional range, an optional hexstring or ASCII string can be specified as the first part of the option 43. If options are in the container, they are appended to the option 43 data. After all options are processed an End Of Option List Option is appended to the data. To return options outside of an option 43, use a regular expression client that matches all clients to specify normal options to return based on the vendor ID.

Keyword	Form	Subcontainers?	Default Value	Meaning
vendor	vendor *vendor_id range* ""			Specifies a vendor container. Vendor containers are used to return option 43 to the client. The vendor id may be specified in a quoted string or a binary string in the form 0x*hexdigits* or hex"*digits*". An optional range may be placed after the vendor id. The range is specified as two dotted quads separated by a dash. After the optional range, an optional hexstring or ASCII string can be specified as the first part of the option 43. If options are in the container, they are appended to the option 43 data. After all options are processed an End Of Option List Option is appended to the data. To return options outside of an option 43, use a regular expression client that matches all clients to specify normal options to return based on the vendor ID.
inoption	inoption *number option_data*	Yes	None	Specifies a container to be matched against any arbitrary incoming option specified by the client. *number* specifies the option number. *option_data* specifies the key to match for this container to be selected during address and option selection for the client. *option_data* is specified in expected form — quoted string, IP address, integer value — for well known options, or it can be optionally specified as a hexadecimal string of bytes if preceded by the characters `0x`. For options that are not well known to the server, a hexadecimal string of bytes can be specified in the same fashion. Additionally, the *option_data* can indicate a regular expression to be compared against the string representation of the client's option data. Regular expressions are specified in a quoted string beginning `"!` (double quote followed by an exclamation mark). The string form of options not well known to the server will be a hexadecimal string of bytes NOT preceded with the characters `0x`.

Keyword	Form	Subcontainers?	Default Value	Meaning
inoption	inoption *number option_data range*	Yes	None	Specifies a container to be matched against any arbitrary incoming option specified by the client. *number* specifies the option number. *option_data* specifies the key to match for this container to be selected during address and option selection for the client. *option_data* is specified in expected form — quoted string, IP address, integer value — for well known options, or it can be optionally specified as a hexadecimal string of bytes if preceded by the characters `0x`. For options that are not well known to the server, a hexadecimal string of bytes can be specified in the same fashion. Additionally, the *option_data* can indicate a regular expression to be compared against the string representation of the client's option data. Regular expressions are specified in a quoted string beginning `"!` (double quote followed by an exclamation mark). The string form of options not well known to the server will be a hexadecimal string of bytes NOT preceded with the characters `0x`.
virtual	virtual fill *id id* ...	No	None	Specifies a virtual subnet with a policy. `fill` means use all addresses in the container before going to the next container. `rotate` means select an address from the next pool in the list on each request. `sfill` and `srotate` are the same as `fill` and `rotate`, but a search is done to see if the client matches containers, vendors, or classes in the subnet. If a match is found that can supply an address, the address is taken from that container instead of following the policy. There can be as many IDs as needed. *id* is either the subnet ID from the subnet definition or the label from the subnet definition. The label is required if there are multiple subnets with the same subnet id.

Keyword	Form	Subcontainers?	Default Value	Meaning
virtual	virtual sfill *id id* ...	No	None	Specifies a virtual subnet with a policy. `fill` means use all addresses in the container before going to the next container. `rotate` means select an address from the next pool in the list on each request. `sfill` and `srotate` are the same as `fill` and `rotate`, but a search is done to see if the client matches containers, vendors, or classes in the subnet. If a match is found that can supply an address, the address is taken from that container instead of following the policy. There can be as many IDs as needed. *id* is either the subnet ID from the subnet definition or the label from the subnet definition. The label is required if there are multiple subnets with the same subnet id.
virtual	virtual rotate *id id* ...	No	None	Specifies a virtual subnet with a policy. `fill` means use all addresses in the container before going to the next container. `rotate` means select an address from the next pool in the list on each request. `sfill` and `srotate` are the same as `fill` and `rotate`, but a search is done to see if the client matches containers, vendors, or classes in the subnet. If a match is found that can supply an address, the address is taken from that container instead of following the policy. There can be as many IDs as needed. *id* is either the subnet ID from the subnet definition or the label from the subnet definition. The label is required if there are multiple subnets with the same subnet id.

Keyword	Form	Subcontainers?	Default Value	Meaning
virtual	virtual srotate *id id* ...	No	None	Specifies a virtual subnet with a policy. `fill` means use all addresses in the container before going to the next container. `rotate` means select an address from the next pool in the list on each request. `sfill` and `srotate` are the same as `fill` and `rotate`, but a search is done to see if the client matches containers, vendors, or classes in the subnet. If a match is found that can supply an address, the address is taken from that container instead of following the policy. There can be as many IDs as needed. *id* is either the subnet ID from the subnet definition or the label from the subnet definition. The label is required if there are multiple subnets with the same subnet id.
inorder:	inorder: *id id* ...	No	None	Specifies a virtual subnet with a policy of fill, which means use all addresses in the container before going to the next container. There can be as many IDs as needed. *id* is either the subnet ID from the subnet definition or the label from the subnet definition. The label is required if there are multiple subnets with the same subnet ID.
balance:	balance: *id id* ...	No	None	Specifies a virtual subnet with a policy of rotate, which means use the next address in the next container. There can be as many IDs as needed. *id* is either the subnet ID from the subnet definition or the label from the subnet definition. The label is required if there are multiple subnets with the same subnet ID.
bootstrapserver	bootstrapserver *IP address*	No	None	Specifies the server clients should use from which to **TFTP** files after receiving **BOOTP** or **DHCP** packets. This value fills in the **siaddr** field in the packet. This is valid at any container level.

Keyword	Form	Subcontainers?	Default Value	Meaning
giaddrfield	giaddrfield *IP address*	No	None	Specifies the giaddrfield for response packets. **Note:** This specification is illegal in the **BOOTP** and **DHCP** protocols, but some clients require the **giaddr** field to be the default gateway for the network. Because of this potential conflict, **giaddrfield** should only be used within a client container, although it can work at any level.
bootfile	bootfile *path*	No	None	Specifies the bootfile to use in the file section of the response packet. This can be specified at any container level. The bootfile policy defines how items specified in the file section of the incoming packet interact with the bootfile and the home directory statements.
pxebootfile	pxebootfile *System Arch MajorVer MinorVer Bootfilename*	No	None	Specifies the bootfile to be given to a client. The config file parser generates an error if the number of parameters after the keyword is less than 4 and ignore if more than 4. This keyword can be used only in a container.

For details about other options, see “DHCP server file known options” on page 213 and “Preboot execution environment vendor container suboption” on page 215.

Boot Image Negotiation Layer daemon

The Boot Image Negotiation Layer daemon (BINLD) server is the third stage of contact for preboot execution environment (PXE) clients.

After communicating with the DHCP server to obtain an IP address, and after communication with the PXE Proxy DHCP server to obtain the location of the boot server, the boot server is contacted to get the filename and location from which to download the boot image. The PXE client can return to communicate with the boot server multiple times in the course of booting if the client requires multiple files in its boot process.

The final stage in the PXE network boot is to download the boot image given by the boot server. The location of the TFTP server and the filename that is to be downloaded is given by the boot server to the PXE client.

BINLD server components

The three main components of the BINLD server are introduced here.

The BINLD server is segmented into three main pieces: a database, a protocol engine, and a set of service threads, each with its own configuration information.

BINLD database:

The `db_file.dhcpo` database is used to generate the options that respond to a client's REQUEST packet.

The options returned by the database depend on the type of server chosen. Options are set using the keyword **pxeservertype** in the `binld.cnf` file.

Using the information in the configuration file, the database is primed and verified for consistency.

BINLD protocol engine:

The protocol engine uses the database to determine what information should be returned to the client.

The PXED protocol engine is based on the Intel Preboot Execution Environment (PXE) Specification Version 2.1, but is still compatible with Intel PXE Specification Version 1.1.

BINLD threaded operations:

The last piece of the BINLD server is actually a set of operations that are used to keep things running.

Because the BINLD server is threaded, these operations are actually set up as threads that occasionally do things to make sure everything is together.

The first thread, the *main* thread, handles the SRC requests (such as **startsrc**, **stopsrc**, **lssrc**, **traceson**, and **refresh**). This thread also coordinates all operations that affect all threads and handles signals. For example,

- A SIGHUP (-1) causes a refresh of all databases in the configuration file.
- A SIGTERM (-15) causes the server to gracefully stop.

The other thread processes packets. Depending on the server type, there can one or two threads. One thread listens on port 67 and the second to port 4011. Each can handle a request from a client.

BINLD configuration

By default, the BINLD server is configured by reading the `/etc/binld.cnf` file, which specifies the server's initial database of options and addresses.

The server is started from SMIT, or through SRC commands.

Configuring the BINLD server is usually the hardest part of using BINLD in your network. First, figure out what networks you need to have PXE clients on. The following example configures a BINLD server to run on the same machine as the DHCP server:

```
pxeservertype       binld_on_dhcp_server

subnet default
{
    vendor pxe
    {
                bootstrapserver  9.3.149.6      #TFTP server IP address
                pxebootfile   1   2   1   window.one   1   0
                pxebootfile   2   2   1   linux.one    2   3
                pxebootfile   1   2   1   hello.one    3   4
                client 6 10005a8ad14d any
                {
                   pxebootfile   1   2   1   aix.one     5   6
                   pxebootfile   2   2   1   window.one  6   7
        }
     }
}
```

Given the above configuration, the BINLD server listens for client's unicast packets on port 4011 and Multicast packets on port 4011 if BINLD gets the Multicast Address from the dhcpsd/pxed. The BINLD server responds to client REQUEST/INFORM packets with the bootfile name and TFTP server's IP

address. If BINLD does not find the bootfile with a matching Layer specified by the client, then it tries to find a bootfile for the next layer. The BINLD does not respond when there is no boot file that matches the client requirements (*Type*, *SystemArch*, *MajorVers*, *MinorVers*, and *Layer*).

The following example configures BINLD to run on a separate machine (that is, DHCP / PXED is not running on the same machine).

```
subnet 9.3.149.0 255.255.255.0
{

     vendor pxe
     {

     bootstrapserver        9.3.149.6      # TFTP server ip address.
     pxebootfile    1    2    1   window.one    1   0
     pxebootfile    2    2    1   linux.one     2   3
     pxebootfile    1    2    1   hello.one     3   4
     client 6 10005a8ad14d any
       {
       pxebootfile    1    2    1   aix.one      5   6
       pxebootfile    2    2    1   window.one   6   7
       }
     }
}
```

In the above example, the *pxeservertype* is not set, so the default server type is **binld_only**. The BINLD server listens for client's unicast packets on port 4011, broadcast & unicast packets on port 67, and Multicast packets on port 4011 if BINLD gets the Multicast Address from the dhcpsd/pxed. The bootfile name and TFTP server IP address is sent to a PXE client only if the client's IP address is in the subnet's IP address range (9.3.149.0 through 9.3.149.255).

The following example configures BINLD to run on the same machine as the PXED server:

```
pxeservertype       binld_on_proxy_server
subnet default
{
     vendor
     {
     bootstrapserver        9.3.149.6          # TFTP server ip address.
     pxebootfile    1    2    1   window.one    1    0
     pxebootfile    2    2    1   linux.one     2    3
     pxebootfile    1    2    1   hello.one     3    4
     client 6 10005a8ad14d any
       {
       pxebootfile    1    2    1   aix.one     5   6
       pxebootfile    2    2    1   window.one  6   7
       }
     }
}
```

In this configuration, the BINLD server only listens on port 4011 for Multicast packets only if BINLD gets Multicast address from the dhcpsd/pxed. If it does not receive any multicast address, then BINLD exits and an error message is logged to the log file.

The database `db_file` clause indicates which database method to use for processing this part of the configuration file. Comments begin with a pound sign (#). From the # to the end of the line are ignored by the PXED server. Each `option` line is used by the server to tell the client what to do. “PXE vendor container suboptions” on page 288 describes the currently supported and known suboptions. See “BINLD server file syntax for general server operation” on page 313 for ways to specify options that the server does not know about.

BINLD configuration file:

The configuration file has an address section and an option definition section, which are based on the concept of containers that hold options, modifiers, and, potentially, other containers.

A *container* (basically, a method to group options) uses an identifier to classify clients into groups. The container types are subnet, class, vendor, and client. Currently, there is not a generic user-definable container. The identifier uniquely defines the client so that the client can be tracked if, for example, it moves between subnets. More than one container type can be used to define client access.

Options are identifiers that are returned to the client, such as default gateway and DNS address.

BINLD containers:

When the DHCP server receives a request, the packet is parsed and identifying keys determine which containers, options, and addresses are extracted.

The last example in BINLD configuration shows a subnet container. Its identifying key is the client's position in the network. If the client is from that network, then it falls into that container.

Each type of container uses a different option to identify a client:

- The subnet container uses the giaddr field or the interface address of the receiving interface to determine which subnet the client came from.
- The class container uses the value in option 77 (User Site Class Identifier).
- The vendor uses the value in option 60 (Vendor Class Identifier).
- The client container uses the option 61 (Client Identifier) for PXED clients and the chaddr field in the BOOTP packet for BOOTP clients.

Except for subnets, each container allows the specification of the value that it matches, including regular expression matching.

There is also an implicit container, the *global* container. Options and modifiers placed in the global container apply to all containers unless overridden or denied. Most containers can be placed inside other containers implying a scope of visibility. Containers may or may not have address ranges associated with them. Subnets, by their nature, have ranges associated with them.

The basic rules for containers and subcontainers are as follows:

- All containers are valid at the global level.
- Subnets can never be placed inside other containers.
- Restricted containers cannot have regular containers of the same type within them. (For example, a container with an option that only allows a class of `Accounting` cannot include a container with an option that allows all classes that start with the letter "a". This is illegal.)
- Restricted client containers cannot have subcontainers.

Given the above rules, you can generate a hierarchy of containers that segment your options into groups for specific clients or sets of clients.

If a client matches multiple containers, how are options and addresses handed out? The DHCP server receives messages, it passes the request to the database (`db_file` in this case), and a container list is generated. The list is presented in order of depth and priority. Priority is defined as an implicit hierarchy in the containers. Strict containers are higher priority than regular containers. Clients, classes, vendors, and finally subnets are sorted, in that order, and within container type by depth. This generates a list ordered by most specific to least specific. For example:

```
Subnet 1
--Class 1
--Client 1
Subnet 2
--Class 1
----Vendor 1
----Client 1
--Client 1
```

The example shows two subnets, `Subnet 1` and `Subnet 2`. There is one class name, `Class 1`, one vendor name, `Vendor 1`, and one client name, `Client 1`. `Class 1` and `Client 1` are defined in multiple places. Because they are in different containers, their names can be the same but values inside them may be different. If `Client 1` sends a message to the DHCP server from `Subnet 1` with `Class 1` specified in its option list, the DHCP server would generate the following container path:

`Subnet 1, Class 1, Client 1`

The most specific container is listed last. To get an address, the list is examined in reverse hierarchy to find the first available address. Then, the list is examined in forward hierarchy to get the options. Options override previous values unless an option *deny* is present in the container. Also, since `Class 1` and `Client 1` are in `Subnet 1`, they are ordered according to the container priority. If the same client is in `Subnet 2` and sends the same message, the container list generated is:

`Subnet 2, Class 1, Client 1` (at the `Subnet 2` level), `Client 1` (at the `Class 1` level)

`Subnet 2` is listed first, then `Class 1`, then the `Client 1` at the `Subnet 2` level (because this client statement is only one level down in the hierarchy). The hierarchy implies that a client matching the first client statement is less specific than the client matching `Client 1` of `Class 1` within `Subnet 2`.

Priority selected by depth within the hierarchy is not superseded by the priority of the containers themselves. For example, if the same client issues the same message and specifies a vendor identifier, the container list is:

`Subnet 2, Class 1, Vendor 1, Client 1` (at `Subnet 2` level), `Client 1` (at `Class 1` level)

Container priority improves search performance because it follows a general concept that client containers are the most specific way to define one or more clients. The class container holds less specific addresses than a client container; vendor is even less specific; and subnet is the least specific.

BINLD addresses and address ranges:

Any container type may have associated addresses ranges; subnets must have an associated address range.

Each range within a container must be a subset of the parent container's range and must not overlap with other containers' ranges. For example, if a class is defined within a subnet and the class has a range, the range must be a subset of the subnet's range. Also, the range within that class container cannot overlap with any other ranges at its level.

Ranges can be expressed on the container line and modified by range and exclude statements to allow for disjoint address sets associated with a container. So, if you have the top ten addresses and the second ten addresses of a subnet available, the subnet could specify these addresses by range in the subnet clause to reduce both memory use and the chance of address collision with other clients not in the specified ranges.

Once an address has been selected, any subsequent container in the list that contains address ranges is removed from the list along with its children. The reason for this is that network-specific options in removed containers are not valid if an address is not used from within that container.

BINLD configuration file options:

After the list has been culled to determine addresses, a set of options is generated for the client.

In this selection process, options overwrite previously selected options unless a *deny* is encountered, in which case, the denied option is removed from the list being sent to the client. This method allows inheritance from parent containers to reduce the amount of data that must be specified.

BINLD logging:

Logging parameters are specified in a container like the database, but the container keyword is **logging_info**.

When learning to configure PXED, it is advisable to turn logging to its highest level. Also, it is best to specify the logging configuration prior to any other configuration file data to ensure that configuration errors are logged after the logging subsystem is initialized. Use the **logitem** keyword to turn on a logging level or remove the **logitem** keyword to disable a logging level. Other keywords for logging allow the specification of the log filename, file size, and the number of rotating log files.

BINLD performance considerations:

It is important to understand that certain configuration keywords and the structure of the configuration file have an effect on the memory use and performance of the PXED server.

First, excessive memory use can be avoided by understanding the inheritance model of options from parent to child containers. In an environment that supports no unlisted clients, the administrator must explicitly list each client in the file. When options are listed for any specific client, the server uses more memory storing that configuration tree than when options are inherited from a parent container (for example, the subnet, network, or global containers). Therefore, the administrator should verify whether any options are repeated at the client level within the configuration file and, if so, determine whether these options can be specified in the parent container and shared by the set of clients as a whole.

Also, when using the **logItem** entries INFO and TRACE, numerous messages are logged during the processing of every PXE client's message. Appending a line to the log file can be an expensive operation; therefore, limiting the amount of logging improves the performance of the PXED server. When an error with the PXED server is suspected, logging can be dynamically re-enabled using the SRC traceson command.

BINLD server file syntax for general server operation

The BINLD server file syntax for general server operation is described here. Forms, subcontainers, default values, and meanings are identified.

Note: Time Units (*time_units*) shown in the following table are optional and represent a modifier to the actual time. The default time unit is minutes. Valid values are seconds (1), minutes (60), hours (3600), days (86400), weeks (604800), months (2392000), and years (31536000). The number shown in parentheses is a multiplier applied to the specified value *n* to express the value in seconds.

Keyword	Form	Subcontainers?	Default Value	Meaning
database	database *db type*	Yes	None	The primary container that holds the definitions for the address pools, options, and client access statements. *db type* is the name of a module that is loaded to process this part of the file. The only value currently available is **db_file**.
logging_info	logging_info	Yes	None	The primary logging container that defines the logging parameters.
logitem	logitem NONE	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed.
logitem	logitem SYSERR	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed.
logitem	logitem OBJERR	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed.
logitem	logitem PROTOCOL	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed.
logitem	logitem PROTERR	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed.
logitem	logitem WARN	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed.
logitem	logitem WARNING	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed.
logitem	logitem CONFIG	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed.
logitem	logitem EVENT	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed.
logitem	logitem PARSEERR	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed.
logitem	logitem ACTION	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed.
logitem	logitem ACNTING	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed.
logitem	logitem STAT	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed.
logitem	logitem TRACE	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed.
logitem	logitem RTRACE	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed.
logitem	logitem START	No	All default to not enabled.	Enables the logging level. Multiple lines are allowed.
numLogFiles	numLogFiles *n*	No	0	Specifies the number of log files to create. The log rotates when the first one fills. *n* is the number of files to create.
logFileSize	logFileSize *n*	No	0	Specifies the size of each log file in 1024-byte units.

Keyword	Form	Subcontainers?	Default Value	Meaning
logFileName	logFileName *path*	No	None	Specifies the path to the first log file. The original log file is named *filename* or *filename.extension*. When a file is rotated, it is renamed beginning with the base *filename*, then either appending a number or replacing the extension with a number. For example, if the original file name is `file`, the rotated file name becomes `file01`. If the original file name is `file.log`, it becomes `file.01`.
pxeservertype	pxeservertype *servertype*	No	dhcp_only	Indicate the type of dhcpsd server it is. *servertype* can be one of the following **binld_on_dhcp_server** This means that **BINLD** is running on the same machine as DHCP server and it is listening for PXE Client request on port 4011 and Multicast address if received from the DHCP / PXED. **binld_on_proxy_server** This means that **BINLD** is running on the same machine as PXED server and it is listening for PXE Client's request on Multicast address if received from the DHCP / PXED. The default value is **binld_only**, which means the BINLD is running on a separate machine and it has to listen for client's packets on port 67 , 4011 and Multicast address if received from the DHCP / PXED.
dhcp_or_proxy _address	dhcp_or_proxy_address *IP address*	No	None	This gives the IP address of dhcp or pxed server to which the BINLD server can send an Unicast packet of type REQUEST/INFORM to receive the Multicast Address. This keyword is defined only when the dhcp or pxed are on a different subnet than BINLD.

BINLD server file syntax for db_file database

The BINLD server file syntax for the `db_file` database is described here. Forms, subcontainers, default values, and meanings are identified.

Note:

1. Time Units (*time_units*) shown in the following table are optional and represent a modifier to the actual time. The default time unit is minutes. Valid values are seconds (1), minutes (60), hours (3600), days (86400), weeks (604800), months (2392000), and years (31536000). The number shown in parentheses is a multiplier applied to the specified value *n* to express the value in seconds.
2. Items that are specified in one container can be overridden inside a subcontainer. For example, you could globally define BOOTP clients, but within a certain subnet allow BOOTP clients by specifying the supportBootp keyword in both containers.
3. The client, class, and vendor containers allow for regular expression support. For class and vendor, a quoted string with the first character after the quote being an exclamation point (!) indicates that the

rest of the string should be treated as a regular expression. The client container allows for regular expressions on both the hwtype and the hwaddr fields. A single string is used to represent both fields with the following format:

```
decimal_number-data
```

If decimal_number is zero, then data is an ASCII string. If any other number, data is hex digits.

Keyword	Form	Subcontainers?	Default Value	Meaning
subnet	subnet default	Yes	None	Specifies a subnet that does not have any range. The subnet is used by a server only when it is responding to INFORM packet from the client and the client's address does not have another matching subnet container.
subnet	subnet *subnet id netmask*	Yes	None	Specifies a subnet and a pool of addresses. All addresses are assumed to be in the pool unless a range is specified on the line or addresses are modified later in the container by a range or exclude statement. The optional range is a pair of IP addresses in dotted quad format separated by a dash. An optional label and priority can be specified. These are used by virtual subnets to identify and order the subnets in the virtual subnet. The label and priority are separated by a colon. These containers are only allowed at the global or database container level.
subnet	subnet *subnet id netmask range*	Yes	None	Specifies a subnet and a pool of addresses. All addresses are assumed to be in the pool unless a range is specified on the line or addresses are modified later in the container by a range or exclude statement. The optional range is a pair of IP addresses in dotted quad format separated by a dash. An optional label and priority can be specified. These are used by virtual subnets to identify and order the subnets in the virtual subnet. The label and priority are separated by a colon. These containers are only allowed at the global or database container level.

Keyword	Form	Subcontainers?	Default Value	Meaning
subnet	subnet *subnet id netmask label:priority*	Yes	None	Specifies a subnet and a pool of addresses. All addresses are assumed to be in the pool unless a range is specified on the line or addresses are modified later in the container by a range or exclude statement. The optional range is a pair of IP addresses in dotted quad format separated by a dash. An optional label and priority can be specified. These are used by virtual subnets to identify and order the subnets in the virtual subnet. The label and priority are separated by a colon. These containers are only allowed at the global or database container level.
subnet	subnet *subnet id netmask range label:priority*			Specifies a subnet and a pool of addresses. All addresses are assumed to be in the pool unless a range is specified on the line or addresses are modified later in the container by a range or exclude statement. The optional range is a pair of IP addresses in dotted quad format separated by a dash. An optional label and priority can be specified. These are used by virtual subnets to identify and order the subnets in the virtual subnet. The label and priority are separated by a colon. These containers are only allowed at the global or database container level.
subnet	subnet *subnet id range*	Yes	None	Specifies a subnet that goes within a network container. It defines a range of addresses that is the whole subnet unless the optional range part is specified. The netmask associated with the subnet is taken from the surrounding network container. **Note:** This method is deprecated in favor of the other subnet forms
option	option *number data ...*	No	None	Specifies an option to send to a client or, in the case of deny, an option to prevent from being sent to the client. The option *** deny** clause means all options not specified in the current container are not to be returned to the client. option *number*deny only denies the specified option. *number* is an unsigned 8-bit integer. *data* is specific to the option (see above) or can be specified as a quoted string (indicating ASCII text) or 0x*hexdigits* or hex"*hexdigits*" or hex "*hexdigits*". If the option is in a vendor container, the option will be encapsulated with other options in an option 43.

Keyword	Form	Subcontainers?	Default Value	Meaning
option	option *number*deny	No	None	Specifies an option to send to a client or, in the case of deny, an option to prevent from being sent to the client. The option * **deny** clause means all options not specified in the current container are not to be returned to the client. option *number*deny only denies the specified option. *number* is an unsigned 8-bit integer. *data* is specific to the option (see above) or can be specified as a quoted string (indicating ASCII text) or 0x*hexdigits* or hex"*hexdigits*" or hex "*hexdigits*". If the option is in a vendor container, the option will be encapsulated with other options in an option 43.
option	option * deny	No	None	Specifies an option to send to a client or, in the case of deny, an option to prevent from being sent to the client. The option * **deny** clause means all options not specified in the current container are not to be returned to the client. option *number*deny only denies the specified option. *number* is an unsigned 8-bit integer. *data* is specific to the option (see above) or can be specified as a quoted string (indicating ASCII text) or 0x*hexdigits* or hex"*hexdigits*" or hex "*hexdigits*". If the option is in a vendor container, the option will be encapsulated with other options in an option 43.
exclude	exclude *an IP address*	No	None	Modifies the range on the container in which the exclude statement is in. The exclude statement is not valid in the global or database container levels. The exclude statement removes the specified address or range from the current range on the container. The exclude statement allows you to create noncontiguous ranges for subnets or other containers.
exclude	exclude *dotted_quad-dotted_quad*	No	None	Modifies the range on the container in which the exclude statement is in. The exclude statement is not valid in the global or database container levels. The exclude statement removes the specified address or range from the current range on the container. The exclude statement allows you to create noncontiguous ranges for subnets or other containers.

Keyword	Form	Subcontainers?	Default Value	Meaning
range	range *IP_address*	No	None	Modifies the range on the container in which the range statement is in. The range statement is not valid in the global or database container levels. If the range is the first in the container that does not specify a range on the container definition line, then the range for the container becomes the range specified by the range statement. Any range statement after the first range or all range statements for a containers that specifies ranges in its definition are added to the current range. With the range statement, a single address or set of addresses can be added to the range. The range must fit inside the subnet container definition.
range	range *dotted_quad-dotted_quad*	No	None	Modifies the range on the container in which the range statement is in. The range statement is not valid in the global or database container levels. If the range is the first in the container that does not specify a range on the container definition line, then the range for the container becomes the range specified by the range statement. Any range statement after the first range or all range statements for a containers that specifies ranges in its definition are added to the current range. With the range statement, a single address or set of addresses can be added to the range. The range must fit inside the subnet container definition.
client	client *hwtype hwaddr* NONE			Specifies a client container that denies the client specified by the *hwaddr* and *hwtype* from getting an address. If *hwtype* is 0, then *hwaddr* is an ASCII string. Otherwise, *hwtype* is the hardware type for the client and *hwaddr* is the hardware address of the client. If the *hwaddr* is a string, then quotes are accepted around the string. If the *hwaddr* is a hexstring, then the address may be specified by 0x*hexdigits* or hex *digits*. *range* allows the client specified by the *hwaddr* and *hwtype* to get an address in the *range*. Must be regular expressions to match multiple clients.

Keyword	Form	Subcontainers?	Default Value	Meaning
client	client *hwtype hwaddr* ANY			Specifies a client container that denies the client specified by the *hwaddr* and *hwtype* from getting an address. If *hwtype* is 0, then *hwaddr* is an ASCII string. Otherwise, *hwtype* is the hardware type for the client and *hwaddr* is the hardware address of the client. If the *hwaddr* is a string, then quotes are accepted around the string. If the *hwaddr* is a hexstring, then the address may be specified by 0x*hexdigits* or hex *digits*. *range* allows the client specified by the *hwaddr* and *hwtype* to get an address in the *range*. Must be regular expressions to match multiple clients.
client	client *hwtype hwaddr dotted_quad*			Specifies a client container that denies the client specified by the *hwaddr* and *hwtype* from getting an address. If *hwtype* is 0, then *hwaddr* is an ASCII string. Otherwise, *hwtype* is the hardware type for the client and *hwaddr* is the hardware address of the client. If the *hwaddr* is a string, then quotes are accepted around the string. If the *hwaddr* is a hexstring, then the address may be specified by 0x*hexdigits* or hex *digits*. *range* allows the client specified by the *hwaddr* and *hwtype* to get an address in the *range*. Must be regular expressions to match multiple clients.
client	client *hwtype hwaddr range*			Specifies a client container that denies the client specified by the *hwaddr* and *hwtype* from getting an address. If *hwtype* is 0, then *hwaddr* is an ASCII string. Otherwise, *hwtype* is the hardware type for the client and *hwaddr* is the hardware address of the client. If the *hwaddr* is a string, then quotes are accepted around the string. If the *hwaddr* is a hexstring, then the address may be specified by 0x*hexdigits* or hex *digits*. *range* allows the client specified by the *hwaddr* and *hwtype* to get an address in the *range*. Must be regular expressions to match multiple clients.

Keyword	Form	Subcontainers?	Default Value	Meaning
class	class *string*	Yes	None	Specifies a class container with name *string*. String can be quoted or not. If quoted, the quotes are removed before comparison. Quotes are required for strings with spaces or tabs. This container is valid at any level. A range can be supplied to indicate a set of addresses to hand out to a client with this class. The range is either a single dotted quad IP address or two dotted quad IP addresses separated by a dash.
class	class *string range*	Yes	None	Specifies a class container with name *string*. String can be quoted or not. If quoted, the quotes are removed before comparison. Quotes are required for strings with spaces or tabs. This container is valid at any level. A range can be supplied to indicate a set of addresses to hand out to a client with this class. The range is either a single dotted quad IP address or two dotted quad IP addresses separated by a dash.
network	network *network id netmask*	Yes	None	Specifies a network ID using class information (for example, 9.3.149.0 with a netmask of 255.255.255.0 would be network 9.0.0.0 255.255.255.0). This version of the network container is used to hold subnets with the same network ID and netmask. When a range is provided, all the addresses in the range are in the pool. The range must be in the network ID's network. This uses class full addressing. This is only valid in the global or database container level. **Note:** The network keyword is deprecated in favor of the subnet container.
network	network *network id*	Yes	None	Specifies a network ID using class information (for example, 9.3.149.0 with a netmask of 255.255.255.0 would be network 9.0.0.0 255.255.255.0). This version of the network container is used to hold subnets with the same network ID and netmask. When a range is provided, all the addresses in the range are in the pool. The range must be in the network ID's network. This uses class full addressing. This is only valid in the global or database container level. **Note:** The network keyword is deprecated in favor of the subnet container.

Keyword	Form	Subcontainers?	Default Value	Meaning
network	network *network id range*			Specifies a network ID using class information (for example, 9.3.149.0 with a netmask of 255.255.255.0 would be network 9.0.0.0 255.255.255.0). This version of the network container is used to hold subnets with the same network ID and netmask. When a range is provided, all the addresses in the range are in the pool. The range must be in the network ID's network. This uses class full addressing. This is only valid in the global or database container level. **Note:** The network keyword is deprecated in favor of the subnet container.
vendor	vendor *vendor_id*	Yes	None	Specifies a vendor container. Vendor containers are used to return option 43 to the client. The vendor id may be specified in a quoted string or a binary string in the form 0x*hexdigits* or hex"*digits*". An optional range may be placed after the vendor id. The range is specified as two dotted quads separated by a dash. After the optional range, an optional hexstring or ASCII string can be specified as the first part of the option 43. If options are in the container, they are appended to the option 43 data. After all options are processed an End Of Option List Option is appended to the data. To return options outside of an option 43, use a regular expression client that matches all clients to specify normal options to return based on the vendor ID. **pxe** after the keyword **vendor** will create a vendor container for PXEClient. **pxeserver** after the keyword **vendor** will create a vendor container for PXEServer.

Keyword	Form	Subcontainers?	Default Value	Meaning
vendor	vendor *vendor_id* hex""	Yes	None	Specifies a vendor container. Vendor containers are used to return option 43 to the client. The vendor id may be specified in a quoted string or a binary string in the form 0x*hexdigits* or hex"*digits*". An optional range may be placed after the vendor id. The range is specified as two dotted quads separated by a dash. After the optional range, an optional hexstring or ASCII string can be specified as the first part of the option 43. If options are in the container, they are appended to the option 43 data. After all options are processed an End Of Option List Option is appended to the data. To return options outside of an option 43, use a regular expression client that matches all clients to specify normal options to return based on the vendor ID. **pxe** after the keyword **vendor** will create a vendor container for PXEClient. **pxeserver** after the keyword **vendor** will create a vendor container for PXEServer.
vendor	vendor *vendor_id* hex ""	Yes	None	Specifies a vendor container. Vendor containers are used to return option 43 to the client. The vendor id may be specified in a quoted string or a binary string in the form 0x*hexdigits* or hex"*digits*". An optional range may be placed after the vendor id. The range is specified as two dotted quads separated by a dash. After the optional range, an optional hexstring or ASCII string can be specified as the first part of the option 43. If options are in the container, they are appended to the option 43 data. After all options are processed an End Of Option List Option is appended to the data. To return options outside of an option 43, use a regular expression client that matches all clients to specify normal options to return based on the vendor ID. **pxe** after the keyword **vendor** will create a vendor container for PXEClient. **pxeserver** after the keyword **vendor** will create a vendor container for PXEServer.

Keyword	Form	Subcontainers?	Default Value	Meaning
vendor	vendor *vendor_id* 0xdata	Yes	None	Specifies a vendor container. Vendor containers are used to return option 43 to the client. The vendor id may be specified in a quoted string or a binary string in the form 0x*hexdigits* or hex"*digits*". An optional range may be placed after the vendor id. The range is specified as two dotted quads separated by a dash. After the optional range, an optional hexstring or ASCII string can be specified as the first part of the option 43. If options are in the container, they are appended to the option 43 data. After all options are processed an End Of Option List Option is appended to the data. To return options outside of an option 43, use a regular expression client that matches all clients to specify normal options to return based on the vendor ID. **pxe** after the keyword **vendor** will create a vendor container for PXEClient. **pxeserver** after the keyword **vendor** will create a vendor container for PXEServer.
vendor	vendor *vendor_id* ""	Yes	None	Specifies a vendor container. Vendor containers are used to return option 43 to the client. The vendor id may be specified in a quoted string or a binary string in the form 0x*hexdigits* or hex"*digits*". An optional range may be placed after the vendor id. The range is specified as two dotted quads separated by a dash. After the optional range, an optional hexstring or ASCII string can be specified as the first part of the option 43. If options are in the container, they are appended to the option 43 data. After all options are processed an End Of Option List Option is appended to the data. To return options outside of an option 43, use a regular expression client that matches all clients to specify normal options to return based on the vendor ID. **pxe** after the keyword **vendor** will create a vendor container for PXEClient. **pxeserver** after the keyword **vendor** will create a vendor container for PXEServer.

Keyword	Form	Subcontainers?	Default Value	Meaning
vendor	vendor *vendor_id range*	Yes	None	Specifies a vendor container. Vendor containers are used to return option 43 to the client. The vendor id may be specified in a quoted string or a binary string in the form 0x*hexdigits* or hex"*digits*". An optional range may be placed after the vendor id. The range is specified as two dotted quads separated by a dash. After the optional range, an optional hexstring or ASCII string can be specified as the first part of the option 43. If options are in the container, they are appended to the option 43 data. After all options are processed an End Of Option List Option is appended to the data. To return options outside of an option 43, use a regular expression client that matches all clients to specify normal options to return based on the vendor ID. **pxe** after the keyword **vendor** will create a vendor container for PXEClient. **pxeserver** after the keyword **vendor** will create a vendor container for PXEServer.
vendor	vendor *vendor_id range* hex""	Yes	None	Specifies a vendor container. Vendor containers are used to return option 43 to the client. The vendor id may be specified in a quoted string or a binary string in the form 0x*hexdigits* or hex"*digits*". An optional range may be placed after the vendor id. The range is specified as two dotted quads separated by a dash. After the optional range, an optional hexstring or ASCII string can be specified as the first part of the option 43. If options are in the container, they are appended to the option 43 data. After all options are processed an End Of Option List Option is appended to the data. To return options outside of an option 43, use a regular expression client that matches all clients to specify normal options to return based on the vendor ID. **pxe** after the keyword **vendor** will create a vendor container for PXEClient. **pxeserver** after the keyword **vendor** will create a vendor container for PXEServer.

Keyword	Form	Subcontainers?	Default Value	Meaning
vendor	vendor *vendor_id range* hex ""	Yes	None	Specifies a vendor container. Vendor containers are used to return option 43 to the client. The vendor id may be specified in a quoted string or a binary string in the form 0x*hexdigits* or hex"*digits*". An optional range may be placed after the vendor id. The range is specified as two dotted quads separated by a dash. After the optional range, an optional hexstring or ASCII string can be specified as the first part of the option 43. If options are in the container, they are appended to the option 43 data. After all options are processed an End Of Option List Option is appended to the data. To return options outside of an option 43, use a regular expression client that matches all clients to specify normal options to return based on the vendor ID. **pxe** after the keyword **vendor** will create a vendor container for PXEClient. **pxeserver** after the keyword **vendor** will create a vendor container for PXEServer.
vendor	vendor *vendor_id range* 0xdata	Yes	None	Specifies a vendor container. Vendor containers are used to return option 43 to the client. The vendor id may be specified in a quoted string or a binary string in the form 0x*hexdigits* or hex"*digits*". An optional range may be placed after the vendor id. The range is specified as two dotted quads separated by a dash. After the optional range, an optional hexstring or ASCII string can be specified as the first part of the option 43. If options are in the container, they are appended to the option 43 data. After all options are processed an End Of Option List Option is appended to the data. To return options outside of an option 43, use a regular expression client that matches all clients to specify normal options to return based on the vendor ID. **pxe** after the keyword **vendor** will create a vendor container for PXEClient. **pxeserver** after the keyword **vendor** will create a vendor container for PXEServer.

Keyword	Form	Subcontainers?	Default Value	Meaning
vendor	vendor *vendor_id range* ""	Yes	None	Specifies a vendor container. Vendor containers are used to return option 43 to the client. The vendor id may be specified in a quoted string or a binary string in the form 0x*hexdigits* or hex"*digits*". An optional range may be placed after the vendor id. The range is specified as two dotted quads separated by a dash. After the optional range, an optional hexstring or ASCII string can be specified as the first part of the option 43. If options are in the container, they are appended to the option 43 data. After all options are processed an End Of Option List Option is appended to the data. To return options outside of an option 43, use a regular expression client that matches all clients to specify normal options to return based on the vendor ID. **pxe** after the keyword **vendor** will create a vendor container for PXEClient. **pxeserver** after the keyword **vendor** will create a vendor container for PXEServer.
vendor	vendor pxe	Yes	None	Specifies a vendor container. Vendor containers are used to return option 43 to the client. The vendor id may be specified in a quoted string or a binary string in the form 0x*hexdigits* or hex"*digits*". An optional range may be placed after the vendor id. The range is specified as two dotted quads separated by a dash. After the optional range, an optional hexstring or ASCII string can be specified as the first part of the option 43. If options are in the container, they are appended to the option 43 data. After all options are processed an End Of Option List Option is appended to the data. To return options outside of an option 43, use a regular expression client that matches all clients to specify normal options to return based on the vendor ID. **pxe** after the keyword **vendor** will create a vendor container for PXEClient. **pxeserver** after the keyword **vendor** will create a vendor container for PXEServer.

Keyword	Form	Subcontainers?	Default Value	Meaning
vendor	vendor pxeserver	Yes	None	Specifies a vendor container. Vendor containers are used to return option 43 to the client. The vendor id may be specified in a quoted string or a binary string in the form 0x*hexdigits* or hex"*digits*". An optional range may be placed after the vendor id. The range is specified as two dotted quads separated by a dash. After the optional range, an optional hexstring or ASCII string can be specified as the first part of the option 43. If options are in the container, they are appended to the option 43 data. After all options are processed an End Of Option List Option is appended to the data. To return options outside of an option 43, use a regular expression client that matches all clients to specify normal options to return based on the vendor ID. **pxe** after the keyword **vendor** will create a vendor container for PXEClient. **pxeserver** after the keyword **vendor** will create a vendor container for PXEServer.
inoption	inoption *number option_data*	Yes	None	Specifies a container to be matched against any arbitrary incoming option specified by the client. *number* specifies the option number. *option_data* specifies the key to match for this container to be selected during address and option selection for the client. *option_data* is specified in expected form — quoted string, IP address, integer value — for well known options, or it can be optionally specified as a hexadecimal string of bytes if preceded by the characters `0x`. For options that are not well known to the server, a hexadecimal string of bytes can be specified in the same fashion. Additionally, the *option_data* can indicate a regular expression to be compared against the string representation of the client's option data. Regular expressions are specified in a quoted string beginning `"!` (double quote followed by an exclamation mark). The string form of options not well known to the server will be a hexadecimal string of bytes NOT preceded with the characters `0x`.

Keyword	Form	Subcontainers?	Default Value	Meaning
inoption	inoption *number option_data range*	Yes	None	Specifies a container to be matched against any arbitrary incoming option specified by the client. *number* specifies the option number. *option_data* specifies the key to match for this container to be selected during address and option selection for the client. *option_data* is specified in expected form — quoted string, IP address, integer value — for well known options, or it can be optionally specified as a hexadecimal string of bytes if preceded by the characters `0x`. For options that are not well known to the server, a hexadecimal string of bytes can be specified in the same fashion. Additionally, the *option_data* can indicate a regular expression to be compared against the string representation of the client's option data. Regular expressions are specified in a quoted string beginning `"!` (double quote followed by an exclamation mark). The string form of options not well known to the server will be a hexadecimal string of bytes NOT preceded with the characters `0x`.
virtual	virtual fill *id id* ...	No	None	Specifies a virtual subnet with a policy. `fill` means use all addresses in the container before going to the next container. `rotate` means select an address from the next pool in the list on each request. `sfill` and `srotate` are the same as `fill` and `rotate`, but a search is done to see if the client matches containers, vendors, or classes in the subnet. If a match is found that can supply an address, the address is taken from that container instead of following the policy. There can be as many IDs as needed. *id* is either the subnet ID from the subnet definition or the label from the subnet definition. The label is required if there are multiple subnets with the same subnet id.

Keyword	Form	Subcontainers?	Default Value	Meaning
virtual	virtual sfill *id id* ...	No	None	Specifies a virtual subnet with a policy. `fill` means use all addresses in the container before going to the next container. `rotate` means select an address from the next pool in the list on each request. `sfill` and `srotate` are the same as `fill` and `rotate`, but a search is done to see if the client matches containers, vendors, or classes in the subnet. If a match is found that can supply an address, the address is taken from that container instead of following the policy. There can be as many IDs as needed. *id* is either the subnet ID from the subnet definition or the label from the subnet definition. The label is required if there are multiple subnets with the same subnet id.
virtual	virtual rotate *id id* ...	No	None	Specifies a virtual subnet with a policy. `fill` means use all addresses in the container before going to the next container. `rotate` means select an address from the next pool in the list on each request. `sfill` and `srotate` are the same as `fill` and `rotate`, but a search is done to see if the client matches containers, vendors, or classes in the subnet. If a match is found that can supply an address, the address is taken from that container instead of following the policy. There can be as many IDs as needed. *id* is either the subnet ID from the subnet definition or the label from the subnet definition. The label is required if there are multiple subnets with the same subnet id.
virtual	virtual srotate *id id* ...	No	None	Specifies a virtual subnet with a policy. `fill` means use all addresses in the container before going to the next container. `rotate` means select an address from the next pool in the list on each request. `sfill` and `srotate` are the same as `fill` and `rotate`, but a search is done to see if the client matches containers, vendors, or classes in the subnet. If a match is found that can supply an address, the address is taken from that container instead of following the policy. There can be as many IDs as needed. *id* is either the subnet ID from the subnet definition or the label from the subnet definition. The label is required if there are multiple subnets with the same subnet id.

Keyword	Form	Subcontainers?	Default Value	Meaning
virtual		No	None	Specifies a virtual subnet with a policy. `fill` means use all addresses in the container before going to the next container. `rotate` means select an address from the next pool in the list on each request. `sfill` and `srotate` are the same as `fill` and `rotate`, but a search is done to see if the client matches containers, vendors, or classes in the subnet. If a match is found that can supply an address, the address is taken from that container instead of following the policy. There can be as many IDs as needed. *id* is either the subnet ID from the subnet definition or the label from the subnet definition. The label is required if there are multiple subnets with the same subnet id.
inorder:	inorder: *id id* ...	No	None	Specifies a virtual subnet with a policy of fill, which means use all addresses in the container before going to the next container. There can be as many IDs as needed. *id* is either the subnet ID from the subnet definition or the label from the subnet definition. The label is required if there are multiple subnets with the same subnet ID.
balance:	balance: *id id* ...	No	None	Specifies a virtual subnet with a policy of rotate, which means use the next address in the next container. There can be as many IDs as needed. *id* is either the subnet ID from the subnet definition or the label from the subnet definition. The label is required if there are multiple subnets with the same subnet ID.
bootstrapserver	bootstrapserver *IP address*	No	None	Specifies the server clients should use from which to TFTP files after receiving BOOTP or DHCP packets. This value fills in the **siaddr** field in the packet. This is valid at any container level.
giaddrfield	giaddrfield *IP address*	No	None	Specifies the giaddrfield for response packets. **Note:** This specification is illegal in the BOOTP and DHCP protocols, but some clients require the **giaddr** field to be the default gateway for the network. Because of this potential conflict, giaddrfield should only be used within a client container, although it can work at any level.

Keyword	Form	Subcontainers?	Default Value	Meaning
bootfile	bootfile *path*	No	None	Specifies the bootfile to use in the file section of the response packet. This can be specified at any container level. The bootfile policy defines how items specified in the file section of the incoming packet interact with the bootfile and the home directory statements.
pxebootfile	pxebootfile *SystemArch MajorVer MinorVer Bootfilename Type Layer*	No	None	Specifies the bootfile to be given to a PXEClient. The config file parser generates an error if the number of parameters after the keyword is less than 4 , ignore if more than 7 and if 4 are there then it assume the value for Type = 0 and Layer = 0. This keyword can be used only in a container.

For details about other options, see “DHCP server file known options” on page 213 and “PXE vendor container suboptions” on page 288.

TCP/IP daemons

Daemons (also known as *servers*) are processes that run continuously in the background and perform functions required by other processes. **Transmission Control Protocol/Internet Protocol (TCP/IP)** provides daemons for implementing certain functions in the operating system.

These daemons are background processes that run without interrupting other processes (unless that is part of the daemon function).

Daemons are invoked by commands at the system management level, by other daemons, or by shell scripts. You can also control daemons with the **inetd** daemon, the **rc.tcpip** shell script, and the System Resource Controller (SRC).

Subsystems and subservers

A *subsystem* is a daemon, or server, that is controlled by the SRC. A *subserver* is a daemon that is controlled by a subsystem. (Daemon commands and daemon names are usually denoted by a **d** at the end of the name.)

The categories of subsystem and subserver are mutually exclusive. That is, daemons are not listed as both a subsystem and as a subserver. The only **TCP/IP** subsystem that controls other daemons is the **inetd** daemon. All **TCP/IP** subservers are also **inetd** subservers.

For a list of the **TCP/IP** daemons, see “TCP/IP daemons” on page 405.

System Resource Control

Among other functions, SRC allows you to start daemons, stop them, and trace their activity. In addition, SRC provides the ability to group daemons into subsystems and subservers.

System Resource Control is a tool designed to aid you in controlling daemons. SRC allows control beyond the flags and parameters available with each daemon command.

See the System Resource Controller in *Operating system and device management* for more information concerning the System Resource Controller.

For a list of the SRC commands, see “SRC commands” on page 403.

Configuring the inetd daemon

Use these steps to configure the **TCP/IP inetd** daemon.

To configure the **inetd** daemon:

1. Specify which subservers it will be invoked by adding an **inetd** daemon.
2. Specify the restart characteristics by changing the restart characteristics of the **inetd** daemon.

Table 75. Configuring the inetd daemon tasks

Task	SMIT fast path	Command or file
Starting the **inetd** Daemon	`smit mkinetd`	**startsrc -s inetd**
Changing Restart Characteristics of the **inetd** Daemon	`smit chinetd` or `smit lsinetd`	
Stopping the **inetd** Daemon	`smit rminetd`	**stopsrc -s inetd**
Listing All **inetd** Subservers	`smit inetdconf`	
Adding an **inetd** Subserver[1]	`smit mkinetdconf`	edit `/etc/inetd.conf` then run **refresh -s inetd** or **kill -1** *inetdPID*[2]
Change/Show Characteristics of an **inetd** Subserver	`smit inetdconf`	edit `/etc/inetd.conf` then run **refresh -s inetd** or **kill -1** *inetdPID*[2]
Removing an **inetd** Subserver	`smit rminetd`	edit `/etc/inetd.conf` then run **refresh -s inetd** or **kill -1** *inetdPID*[2]

Note:

1. Adding an **inetd** subserver configures the **inetd** daemon so that it invokes the subserver when it is needed.
2. Both the **refresh** and the **kill** commands inform the **inetd** daemon of changes to its configuration file.

Client network services

Client Network Services (accessible using the the SMIT fast path, `smit clientnet`) refers to the **TCP/IP** protocols available for use by this operating system.

Each protocol (or service) is known by the port number that it uses on the network, hence the term *well-known port*. As a convenience to programmers, the port numbers can be referred to by names as well as numbers. For example, the **TCP/IP** mail protocol uses port 25 and is known by the name **smtp**. If a protocol is listed (uncommented) in the `/etc/services` file, then a host can use that protocol.

By default, all the **TCP/IP** protocols are defined in the `/etc/services` file. You do not have to configure this file. If you write your own client/server programs, you might want to add your service to the `/etc/services` file, and reserve a specific port number and name for your service. If you do decide to add your service to `/etc/services`, note that port numbers 0 through 1024 are reserved for system use.

Table 76. Client network services tasks

Task	SMIT fast path	Command or file
Listing All Services	`smit lsservices`	view `/etc/services`
Adding a Service	`smit mkservices`	edit `/etc/services`
Change/Show Characteristics of a Service	`smit chservices`	edit `/etc/services`
Removing a Service	`smit rmservices`	edit `/etc/services`

Server network services

Server network services include controlling remote access, starting or stopping **TCP/IP**, and managing the **pty** device driver, as shown in this table.

The **pty** device driver is installed automatically with the system. By default, it is configured to support 16 BSD-style symbolic links, and it is available for use by the system at boot time.

Table 77. Server network services tasks

Task	SMIT fast path	Command or file
Controlling remote access		See "Remote Command Execution Access" and "Restricted File Transfer Program Users" in *Security*.
Start, restart, or stop TCP/IP subsystems	`smit otherserv`	See “System Resource Control” on page 332.
Change/show characteristics of the pty device driver	`smit chgpty`	**chdev -l pty0 -P -a num=***X* where *X* ranges from 0 to 64
Make the pty device driver unavailable for use	`smit pty` then select **Remove the PTY; Keep Definition**	No related command or file.
Make the pty device driver available for use	`smit pty` then select **Configure the Defined PTY**	No related command or file.
Generate an error report	`smit errpt`	No related command or file.
Trace the pty	`smit trace`	No related command or file.

TCP/IP routing

A *route* defines a path for sending packets through the Internet network to an address on another network.

A route does not define the complete path, only the path segment from one host to a gateway that can forward packets to a destination (or from one gateway to another). There are five types of routes:

Item	Description
host route	Defines a gateway that can forward packets to a specific host on another network.
network route	Defines a gateway that can forward packets to any of the hosts on a specific network.
default route	Defines a gateway to use when a host or network route to a destination is not otherwise defined.
loopback route	Default route for all packets sent to local network addresses. The loopback route IP is always 127.0.0.1.
broadcast route	Default route for all broadcast packets. Two broadcast routes are automatically assigned to each subnet on which the network has an IP (one to the subnet address and one to the broadcast address of the subnet).

Routes are defined in the kernel *routing table*. The route definitions include information on networks reachable from the local host and on gateways that can be used to reach remote networks. When a gateway receives a datagram, it checks the routing tables to find out where next to send the datagram along the path to its destination.

You can add multiple routes for the same destination in the kernel routing table. A routing lookup evaluates all routes that match the request then chooses the route with the lowest distance metric. If multiple matching routes have equal distance, a lookup chooses the most specific route. If both criteria are equal for multiple routes, routing lookups alternate choices of matching routes.

Static and dynamic routing

In **TCP/IP**, routing can be one of two types: *static* or *dynamic*.

With static routing, you maintain the routing table manually using the **route** command. Static routing is practical for a single network communicating with one or two other networks. However, as your network begins to communicate with more networks, the number of gateways increases, and so does the amount of time and effort required to maintain the routing table manually.

With dynamic routing, daemons update the routing table automatically. Routing daemons continuously receive information broadcast by other routing daemons, and so continuously update the routing table.

TCP/IP provides two daemons for use in dynamic routing, the **routed** and **gated** daemons. The **gated** daemon supports **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)**, and **Internet Control Message Protocol (ICMP and ICMPv6)/Router Discovery** routing protocols simultaneously. In addition, the **gated** daemon supports the **Simple Network Management Protocol (SNMP)**. The **routed** daemon only supports **Routing Information Protocol**.

Routing daemons can operate in one of two modes, *passive* or *active*, depending upon the options you use when starting the daemons. In active mode, routing daemons both broadcast routing information periodically about their local network to gateways and hosts, and receive routing information from hosts and gateways. In passive mode, routing daemons receive routing information from hosts and gateways, but do not attempt to keep remote gateways updated (they do not advertise their own routing information).

These two types of routing can be used not only for gateways, but for other hosts on a network as well. Static routing works the same for gateways as for other hosts. Dynamic routing daemons, however, must be run in the passive (quiet) mode when run on a host that is not a gateway.

TCP/IP routing gateways

Gateways are a type of router. *Routers* connect two or more networks and provide the routing function. Some routers, for example, route at the network interface level or at the physical level. *Gateways*, however, route at the network level.

Gateways receive IP datagrams from other gateways or hosts for delivery to hosts on the local network, and route IP datagrams from one network to another. For example, a gateway connecting two Token-Ring networks has two Token-Ring adapter cards, each with its own Token-Ring network interface. To pass on information, the gateway receives datagrams through one network interface and sends them out through the other network interface. Gateways periodically verify their network connections through interface status messages.

Gateways route packets according to the destination network, not according to the destination host. That is, a gateway machine is not required to keep track of every possible host destination for a packet. Instead, a gateway routes packets according to the network of the destination host. The destination network then takes care of sending the packet to the destination host. Thus, a typical gateway machine requires only limited disk storage capacity (if any) and limited main memory capacity.

The distance a message must travel from originating host to destination host depends upon the number of *gateway hops* it must make. A gateway is zero hops from a network to which it is directly attached, one hop from a network that is reachable through one gateway, and so on. Message distance is usually expressed in the number of gateway hops required, or *hop counts* (also called the *metric*).

Interior and exterior routing gateways:

Interior gateways are gateways that belong to the same autonomous system. They communicate with each other using the **Routing Information Protocol (RIP)**, **Routing Information Protocol Next Generation (RIPng)**, **Intermediate System to Intermediate System** protocol, **Open Shortest Path First (OSPF)** protocol, or the **HELLO Protocol (HELLO)**. Exterior gateways belong to different autonomous systems. They use the **Exterior Gateway Protocol (EGP)**, the **Border Gateway Protocol (BGP)**, or **BGP4+**.

For example, consider two autonomous systems. The first is all the networks administered by the Widget Company. The second is all the networks administered by the Gadget Company. The Widget Company has one machine, called apple, which is Widget's gateway to the Internet. The Gadget Company has one

machine, called orange, which is Gadget's gateway to the Internet. Both companies have several different networks internal to the companies. The gateways connecting the internal networks are interior gateways. But apple and orange are exterior gateways.

Each exterior gateway does not communicate with every other exterior gateway. Instead, the exterior gateway acquires a set of neighbors (other exterior gateways) with which it communicates. These neighbors are not defined by geographic proximity, but rather by their established communications with each other. The neighboring gateways, in turn, have other exterior gateway neighbors. In this way, the exterior gateway routing tables are updated and routing information is propagated among the exterior gateways.

The routing information is sent in a pair, (N,D), where N is a network and D is a distance reflecting the cost of reaching the specified network. Each gateway advertises the networks it can reach and the costs of reaching them. The receiving gateway calculates the shortest paths to other networks and passes this information along to its neighbors. Thus, each exterior gateway is continually receiving routing information, updating its routing table and then passing that information to its exterior neighbors.

Gateway protocols:

All gateways, whether interior or exterior, use protocols to communicate with each other. Here are brief descriptions of the more commonly used **TCP/IP** gateway protocols:

HELLO Protocol (HELLO)
HELLO is one protocol that the interior gateways use to communicate among themselves. **HELLO** calculates the shortest path to other networks by determining the path that has the least delay time.

Routing Information Protocol (RIP)
Routing Information Protocol is a protocol that the interior gateways use to communicate among themselves. Like the **HELLO Protocol**, **RIP** calculates the shortest path to other networks. Unlike **HELLO**, **RIP** estimates distance not by delay time, but by hop counts. Because the **gated** daemon stores all metrics internally as time delays, it converts **RIP** hop counts into time delays.

Routing Information Protocol Next Generation
RIPng is the **RIP** protocol that is enhanced to support **IPv6**.

Open Shortest Path First (OSPF)
OPSF is a protocol that the interior gateways use to communicate among themselves. It is a link-state protocol that is better suited than **RIP** for complex networks with many routers. It provides equal cost multipath routing.

Exterior Gateway Protocol (EGP)
The exterior gateways can use the **Exterior Gateway Protocol** to communicate among themselves. The **EGP** does not calculate the shortest path to other networks. Instead, it merely indicates whether a particular network is reachable or not.

Border Gateway Protocol (BGP)
The exterior gateways can use this protocol to communicate among themselves. It exchanges reachability information between autonomous systems, but provides more capabilities than **EGP**. **BGP** uses path attributes to provide more information about each route as an aid in selecting the best route.

Border Gateway Protocol 4+
BGP4+ is the **BGP** protocol version 4, which supports **IPv6** and has other enhancements over past versions of the protocol.

Intermediate System to Intermediate System (IS-IS)
Interior gateways use **IS-IS** protocol to communicate among themselves. It is a link-state protocol that can route IP and ISO/CLNP packets and, like **OSPF**, uses a "shorter path first" algorithm to determine routes.

Gateway considerations

Take these actions before configuring your gateway.

Before you configure the gateways for your network, you must first do the following:

1. Consider the number of gateways to use. The number of gateways you need to configure will depend upon:
   - The number of networks you want to connect.
   - How you want to connect the networks.
   - The level of activity on the connected networks.

   For example, suppose users on Network 1, Network 2, and Network 3 all need to communicate with each other.

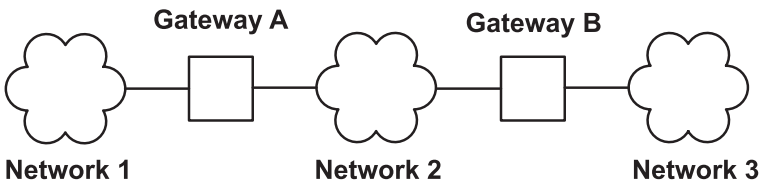


Figure 24. Simple gateway configuration

   This illustration contains three network clouds numbered one, two, and three. Networks one and two are connected with gateway A. Networks two and three are connected with gateway B.

   To connect Network 1 directly to Network 2, you would use a single gateway (Gateway A). To connect Network 2 directly to Network 3, you would use another gateway (Gateway B). Now, assuming the proper routes are defined, all the users on all three networks can communicate.

   However, if Network 2 is very busy, communication between Network 1 and Network 3 might suffer unacceptable delays. Furthermore, if most of the inter-network communication occurs between Network 1 and Network 3, you might want to connect Network 1 directly to Network 3. To do this, you could use an additional pair of gateways, Gateway C (on Network 1) and Gateway D (on Network 3), with a direct connection between these two additional gateways. This may be an inefficient solution, however, because one gateway can connect more than two networks.

   A more efficient solution would be to connect Gateway A to Gateway B directly, as well as to Network 2. This would require a second network adapter in both Gateway A and Gateway B. In general, the number of networks you connect through a single gateway is limited by the number of network adapter cards the gateway machine can support.

2. Decide on the type of routing to use.

   If your network is small, and its configuration rarely changes, you probably want to use static routing. But if you have a large network whose configuration changes frequently, you probably want to use dynamic routing. You might decide to use a combination of static and dynamic routing. That is, you might want to give static definitions to a few specific routes, while allowing other routes to be updated by the daemons. The static routes you create are not advertised to other gateways and are not updated by the routing daemons.

3. If you are using dynamic routing, choose the routing daemon according to the type of gateway you need and the protocols your gateway must support. If the gateway is an interior gateway, and only needs to support **RIP**, choose the **routed** daemon. If the gateway must support any other protocol, or is an exterior gateway, choose the **gated** daemon.

   Note: Unpredictable results can occur if the **gated** and **routed** daemons run on the same host at the same time.

Configuring a gateway

To configure a machine to act as a gateway, use these instructions.

For clarity, this procedure assumes that the gateway machine connects two networks, and that the gateway machine has already been minimally configured on one of the networks.

1. Install and configure the second network adapter, if you have not done so already. (See “Installing a network adapter” on page 155 and “Adapter management and configuration” on page 156.)
2. Choose an IP address for the second network interface, and then configure the network interface by following the instructions in “Network interface management” on page 161.
3. Add a route to the second network.
4. To use a machine as an internetwork router over **TCP/IP** networks, type:

   ```
   no -o ipforwarding=1
   ```

The gateway machine can now access both of the networks to which it is directly attached.

1. If you want to use static routing to communicate with hosts or networks beyond these two networks, add any other routes you want.
2. If you want to use dynamic routing, follow the instructions in either “Configuring the routed daemon” on page 340 or “Configuring the gated daemon” on page 341. If your internetwork is to join the Internet, you should also follow the instructions in “Autonomous system numbers” on page 343.

Table 78. Configuring gateway tasks

Task	SMIT fast path	Command file
Displaying the Routing Table	`smit lsroute`	**netstat -rn**[1]
Adding a Static Route	`smit mkroute`	**route add** *destination gateway*[2]
Removing a Static Route	`smit rmroute`	**route delete** *destination gateway*[2]
Flushing the Routing Table	`smit fshrttbl`	**route flush**

Note:

1. The table is divided into columns for destination address, gateway address, flags, reference count (hop count), and network interface. (For a detailed discussion of each of these columns, see the **netstat** command in the *Commands Reference, Volume 4*.) If frames are not reaching their destination and the routing tables indicate the correct route, one or more of the following conditions might exist:
   - Network is failing.
   - Remote host or gateway is failing.
   - Remote host or gateway is down or not ready to receive frames.
   - Remote host does not have a route back to the source network.
2. The *destination* value is the dotted decimal address or symbolic name of the destination host or network, and the *gateway* value is the dotted decimal address or symbolic name of the gateway. (A default route specifies 0 as the destination.)

Route use restrictions

Routes can be restricted so they can be used only by some users. The restrictions are based on the primary group IDs of users.

Using the **route** command, you can specify a list of up to 32 group IDs that are allowed or not allowed to use a route. If the list is of allowed groups, any user that belongs to any group on the list can use the route. If the list is of disallowed groups, only users that do not belong to any of the groups on the list can use the route. The root user can use any route.

Groups can also be associated with an interface using the **ifconfig** command. In this case, a forwardable packet can use any route allowed for the groups associated with its incoming interface.

If there are two or more routes to the same destination, any ICMP redirects that are received for that destination will be ignored and path MTU discovery will not be done on those routes.

Dead gateway detection

A host can be configured to detect whether a gateway it is using is down, and can adjust its routing table accordingly.

If the network option **-passive_dgd** is 1, passive dead gateway detection is enabled for the entire system. If no response is received for consecutive **dgd_packets_lost ARP** requests to a gateway, that gateway is assumed to be down and the distance metrics (also known as *hopcount* or *cost*) for all routes that use that gateway are raised to the maximum possible value. After **dgd_retry_time** minutes have passed, the route's costs are restored to their user-configured values. The host also takes action based on failing **TCP** connections. If consecutive **dgd_packets_lost TCP** packets are lost, the **ARP** entry for the gateway in use is deleted and the **TCP** connection tries the next-best route. The next time the gateway is used, the above actions take place if the gateway is actually down. The **passive_dgd**, **dgd_packets_lost**, and **dgd_retry_time** parameters can all be configured by using the **no** command.

Hosts can also be configured to use active dead gateway detection on a per-route basis with the **-active_dgd** flag of the **route** command. Active dead gateway detection pings all gateways used by routes for which it is enabled every **dgd_ping_time** second. If no response is received from a gateway, it is pinged more rapidly up to **dgd_packets_lost** times. If still no response is received, the costs of all routes that use that gateway are raised. The gateway continues to be pinged, and if a response is eventually received, the costs on the routes are restored to their user-configured values. The **dgd_ping_time** parameter can be configured by using the **no** command.

Dead gateway detection is most useful for hosts that use static rather than dynamic routing. Passive dead gateway detection results in less performance issues and is recommended for use on any network that has redundant gateways. However, passive dead gateway detection is done on a best-effort basis only. Some protocols, such as **UDP**, do not provide any feedback to the host if a data transmission is failing, and in this case no action can be taken by passive dead gateway detection.

Active dead gateway detection is most useful when a host must discover immediately when a gateway goes down. Since it queries each gateway for which it is enabled every few seconds, there is some excess network usage that is associated with its use. Active dead gateway detection is recommended only for hosts that provide critical services and on networks with a limited number of hosts.

Note: Dead gateway detection and the routing protocols that are used by the **gated** and **routed** daemons perform a similar function by discovering changes in the network configuration and adjusting the routing table accordingly. However, they use different mechanisms to do this, and if they are run at the same time, they might conflict with one another. For this reason, dead gateway detection must not be used on systems that run the **gated** or **routed** daemons.

When dead gateway detection detects that the primary route is back online and the **dgd_flush_cached_route** parameter is enabled, the current cached routes of all active connections are flushed. The routes of all the current active connections are validated again, to find the best route for sending data. The **dgd_flush_cached_route** parameter can be configured, by using the **no** command. By default, the **dgd_flush_cached_route** parameter is disabled.

Note: The **dgd_flush_cached_route** parameter must be enabled only in a stable network environment. Otherwise, there might be greater performance issues due to bad or unstable hardware routers, causing dead gateway detection to frequently update the routing table. Frequent flushing of the cached routes can also be expensive.

Route cloning

Route cloning allows a host route to be created for every host that a system communicates with.

When network traffic is about to be sent, a search is done of the routing table to find a route to that host. If a specific host route is found, it will be used. If a specific host route is not found, a network route or the default route may be found. If the route that is found has the cloning flag, 'c', set, a host route for the

destination will be created using the gateway from the route that is being cloned. Subsequent routing table searches for that destination will find the cloned host route. Cloned routes have the 'W' flag set. These routes will time out and be deleted from the routing table if they are unused for *route_expire* minutes. You can modify *route_expire* by using the **no** command.

The route cloning feature is used mainly by the path MTU discovery protocol within AIX operating system, to allow it to keep track of path MTU information for every destination it communicates with. If the network options **tcp_pmtu_discover** or **udp_pmtu_discover** (settable with the **no** command) are 1, the cloning flag is turned on for all network routes on the system. The path MTU discovery protocol is on by default.

Note: To manually add a cloning route entry, you can manipulate the routing table through the **route** command.

Related information:

route command

Dynamic route removal

If you are using the **routed** daemon, a manually deleted route is *not* replaced by incoming RIP information (because ioctl's are used).

If you are using the **gated** daemon, and the **-n** flag is not used, the manually deleted route *is* replaced by the route as discovered in incoming RIP information.

Configuring the routed daemon

Follow these steps to configure the **routed** daemon.

To configure the **routed** daemon:

1. Remove the comment symbol (#) and modify the routed clause in the `/etc/rc.tcpip` shell script. This automatically starts the **routed** daemon with each system startup.
   - Specify whether you want the gateway to run in active (**-s** flag) or passive (**-q** flag) mode.
   - Specify whether you want packet tracing on or off (**-t** flag). Packet tracing can also be turned on after the **routed** daemon is already started by using the **kill** command to send a **SIGUSR1** signal to the daemon. This signal can also be used to increment the level of tracing through four levels. In addition, packet tracing can be turned off while the **routed** daemon is running by using the **kill** command to send a **SIGUSR2** signal to the daemon. For more information, see the **routed** daemon and the **kill** command.
   - Specify whether you want debugging turned on or off (**-d** flag). If you use this flag, specify which log file you want debugging information stored in, or choose for it to be directed to the console display.
   - Specify whether you are running the **routed** daemon on a gateway (**-g** flag).

     Note: A host that is not a gateway can run the **routed** daemon, but it must be run in passive mode.
2. Identify any known networks by listing them in the `/etc/networks` file. See Networks File Format for TCP/IP in the *Files Reference* for more information. A sample `networks` file is located in the `/usr/samples/tcpip` directory.
3. Set up routes in the `/etc/gateways` file to any known gateways that are not directly connected to your network. Refer to Gateways File Format for TCP/IP in *Files Reference* for detailed examples of entries in the `/etc/gateways` file. A sample `gateways` file is located in the `/usr/samples/tcpip` directory.

Attention: Do not run the **routed** daemon and the **gated** daemon on the same machine. Unpredictable results can occur.

Configuring the gated daemon

When configuring the **gated** daemon, you must decide which gateway protocols are most appropriate for your system.

To configure the **gated** daemon:

1. Decide which gateway protocols are most appropriate for your system. The choices for routing protocols are **EGP**, **BGP**, **RIP**, **RIPng**, **HELLO**, **OSPF**, **ICMP/Router Discovery**, and **IS-IS**. You can also use **SNMP**, a protocol allowing you to change or show management information for a network element from a remote host.

   Note: Use **EGP**, **BGP**, or **BGP4**+ to advertise addresses of networks in an autonomous system to gateways in other autonomous systems. If you are on the Internet, **EGP**, **BGP**, or **BGP4+** must be used to advertise network reachability to the core gateway system. Use the interior routing protocols to advertise reachability information within an autonomous system.
2. Identify any known networks by listing them in the `/etc/networks` file. See Networks File Format for TCP/IP in *Files Reference* for more information. A sample `networks` file is located in the `/usr/samples/tcpip` directory.
3. Edit the `/etc/gated.conf` file to reflect the desired **gated** daemon configuration.

   Note: The **gated** version on AIX 4.3.2 and higher is 3.5.9. The syntax of the `/etc/gated.conf` file has changed. The examples given below are for the 3.5.9 version of **gated**. To configure the `/etc/gated.conf` file for versions prior to AIX 4.3.2, use the syntax provided in the `/etc/gated.conf` file itself.

   a. Specify the level of trace output you want. If tracing is needed before the `gated.conf` file is parsed, use the **-t** flag to turn tracing on when the daemon starts. See gated Daemon in *Commands Reference, Volume 2* for more information.
   b. Specify the routing protocols you want to use. Each protocol has its own protocol statement. Remove the comment symbols (#) and modify the statements corresponding to the protocols you want to use.
      - If using **EGP**:
        - Set up the **EGP** `autonomoussystem` clause. Obtain an autonomous system number from the Internet authority if you are on the Internet, or if not, assign an autonomous system number considering the autonomous system numbers of other systems on your network.
        - Set the **EGP** statement to `yes`.
        - Set up a `group` clause for each autonomous system.
        - Set up a `neighbor` clause for each neighbor in that autonomous system. For example:

```
autonomoussystem 283 ;

egp yes {
        group maxup 1 {
             neighbor nogendefault 192.9.201.1 ;
             neighbor nogendefault 192.9.201.2 ;
       } ;
       group {
             neighbor 192.10.201.1 ;
             neighbor 192.10.201.2 ;
       } ;
} ;
```

      - If using **RIP** or **HELLO**:
        - Set the **RIP** or **HELLO** statement to `yes`.
        - Specify `nobroadcast` in the **RIP** or **HELLO** statement if you want the gateway only to accept routing information, not broadcast information. Or specify `broadcast` in the **RIP** or **HELLO** statement if you want the gateway to broadcast routing information as well as accept routing information.

– If you want the gateway to send directly to source gateways, use the `sourcegateways` statement. Specify a gateway name or Internet address in dotted decimal in the `sourcegateways` clause. For example:

```
# Send directly to specific gateways

rip/hello yes {
          sourcegateways
                101.25.32.1
                101.25.32.2 ;
} ;
```

The following example shows the **RIP**/**HELLO** stanza in the `gated.conf` file of a machine that does not send **RIP** packets, and does not receive **RIP** packets on its tr0 interface.

```
rip/hello nobroadcast {
         interface tr0 noripin ;
} ;
```

- If using **BGP**:
  – Set up the **BGP** `autonomoussystem` clause. Obtain an autonomous system number from the Internet authority if you are on the Internet, or if not, assign an autonomous system number considering the autonomous system numbers of other systems on your network.
  – Set the **BGP** statement to `yes`.
  – Set up a `peer` clause for each neighbor in that autonomous system. For example:

```
# Perform all BGP operations

bgp yes {
        peer 192.9.201.1 ;
} ;
```

- If using **SNMP**:
  – Set the **SNMP** statement to `yes`.

```
snmp yes ;
```

Configuring the gated daemon to run IPv6:

Use this procedure to configure the **gated** daemon to run **Internet Protocol version 6 (IPv6)**.

To configure the **gated** daemon to run under **Internet Protocol version 6 (IPv6)**, first ensure that your system has been configured for **IPv6** and **IPv6** routing:

1. Run **autoconf6** to automatically configure your interfaces for **IPv6**.
2. Configure site local addresses for each **IPv6** interface on which you want to use **IPv6** routing using the following command:

   `ifconfig` *`interface`* `inet6 fec0:`*`n`*`::`*`address`*`/64 alias`

   where

   interface
   Is the name of the interface, such as `tr0` or `en0`.

   n Is any decimal number; for example, `11`

   address Is the portion of the **IPv6** interface address that follows the double colons; for example, given the **IPv6** address `fe80::204:acff:fe86:298d`, the *address* entry would be `204:acff:fe86:298d`.

   Note: You can use the command **netstat -i** to see what your **IPv6** address is for each configured interface.

   If token ring `tr0` has an **IPv6** address of `fe80::204:acff:fe86:298d`, you issue the following command:

   ```
   ifconfig tr0 inet6 fec0:13::204:acff:fe86:298d/64 alias
   ```

3. Turn on **IPv6** forwarding with the following command:

```
no -o ip6forwarding=1
```

4. Start **ndpd-router** with the following command:

   ```
   ndpd-router -g
   ```

   See **ndpd-router** in *Commands Reference, Volume 4* to determine which flags to use for your network configuration.

   Starting **ndpd-router** allows your system to act as a router for the **Neighbor Discovery Protocol**. **Neighbor Discovery Protocol** routers inform Neighbor Discovery hosts with routing information so hosts can route **IPv6** packets.

   Any hosts on the network that you want to be part of the **IPv6** network must run **ndpd-host**. Hosts on the network that run **ndpd-host** will recognize themselves as part of an **IPv6** network and use **Neighbor Discovery Protocol**, which allows them to determine and monitor link-layer addresses both to allow neighbor routing and to find neighboring routers for forwarding packets.

   See **ndpd-router** and **ndpd-host** in *Commands Reference, Volume 4*, or read RFC 1970, *Neighbor Discovery*, for more information.

5. Next, configure the **gated** daemon:
   a. Decide which **IPv6** gateway protocols are most appropriate for your system. The choices for **IPv6** routing protocols are **Border Gateway Protocol** enhanced for **IPv6** (**BGP4+**) and **Routing Information Protocol Next Generation** (**RIPng**).
   b. Edit the `/etc/gated.conf` file to reflect the desired **gated** daemon configuration.

      Note: AIX 4.3.2 and later run **gated** version 3.5.9. The syntax of the `gated.conf` file has changed slightly from earlier versions. Read the `gated.conf` documentation in in *Files Reference*, or use the sample file that is shipped in the `/usr/sample/tcpip` directory for correct syntax.

      When configuring **BGP4+** or **RIPng**, use **IPv6** addresses in which the syntax specifies an IP address.

      Note: By default, **RIPng** multicasts its packets.

After the `/etc/gated.conf` file has been modified, the **gated** daemon can be started.

Autonomous system numbers

If you use **EGP** or **BGP**, you should obtain an official *autonomous system number* for your gateway.

To obtain an official autonomous system number, contact the NIC at the following internet address:
`INFO@INTERNIC.NET`

Mobile IPv6

Mobile **IPv6** provides mobility support for **IPv6**. It allows you to keep the same internet address all over the world, and allows applications using that address to maintain transport and upper-layer connections when changing locations. It allows mobility across homogenous and heterogeneous media.

For example, Mobile **IPv6** facilitates node movement from an Ethernet segment to a wireless LAN cell while the mobile node's IP address remains unchanged.

In Mobile **IPv6**, each mobile node is identified by two IP addresses: its home address and its care-of address. The home address is a permanent IP address that identifies the mobile node regardless of its location. The care-of address changes at each new point of attachment and provides information about the mobile node's current situation. When a mobile node arrives to a visited network, it must acquire a care-of address, which will be used during the time that the mobile node is under this location in the visited network. It may use the methods of **IPv6** Neighborhood Discovery to get the care-of address (see "Neighbor discovery/stateless address autoconfiguration" on page 123). Both stateless and stateful autoconfiguration are possible. The care-of address can also be manually configured. How the care-of address is acquired is irrelevant to Mobile **IPv6**.

There must be at least one home agent configured on the home network, and the mobile node must be configured to know the IP address of its home agent. The mobile node sends a packet containing a binding update to the home agent. The home agent receives the packet and makes an association between the home address to the mobile node and the care-of address it received. The home agent responds with a packet containing a binding acknowledgment.

The home agent keeps a binding cache containing associations between the home addresses and the care-of addresses for the mobile nodes it serves. The home agent will intercept any packets destined for the home address and forward them to the mobile nodes. A mobile node will then send a binding update to the correspondent node informing it of its care-of address, and the correspondent node will create a binding cache entry so that it can send future traffic directly to the mobile node at its care-of address.

Mobility support in AIX provides the following basic functions:

As a **Home Agent** node:

- Maintain an entry in its binding cache for each mobile node for which it is serving.
- Intercept packets addressed to a mobile node for which it is currently serving as the home agent, on that mobile node's home link, while the mobile node is away from home.
- Encapsulate such intercepted packets in order to tunnel them to the primary care-of address for the mobile node indicated in its binding in the home agent's binding cache.
- Return a binding acknowledgment option in response to a binding update option received with the acknowledge bit set.
- Process the Duplicate Address Detection on the mobile node's care-of address to ensure the **IPv6** addresses are unique.
- Support Dynamic Home Agent Address Discovery to assist the mobile nodes is discovering the addresses of the home agents.
- Support the reception of Mobile Prefix Solicitation and the sending of Mobile Prefix Advertisement.

As a **Stationary Correspondent** node:

- Process a home address option received in any **IPv6** packet
- Process a binding update option received in a packet and to return a binding acknowledgement option if the acknowledge (A) bit is set in the received binding update
- Maintain a binding cache of the bindings received in accepted binding updates
- Send packets using a routing header when there is a binding cache entry for a mobile node that contains the mobile node's current care-of address

As a **Router** node in a Network visited by the mobile node:

- Send an advertisement interval option in its router advertisements to aid movement detection by mobile nodes. It is configurable by the **-m** parameter in the **ndpd-router** daemon.
- Support sending unsolicited multicast router advertisements at the faster rate described in RFC 2461. It is configurable by the **-m** and **-D** parameters in the **ndpd-router** daemon.
- Send a Home Agent Information option (home agent preference and lifetime) in its router advertisements to aid mobile nodes to choose their home agent. It is configurable by the **-H** parameter in the **ndpd-router** daemon.

Mobile IPv6 security

The binding update and binding acknowledgement messages exchanged between the mobile node and the home agent must be protected by IP Security using Encapsulating Security Payload (ESP) protection with a non-NULL payload authentication algorithm.

For more information about IP Security, see *Security*.

The binding establishment between the mobile node and the correspondent node is made secure by using the Return Routability procedure. In this procedure, the messages that are exchanged between the home agent node and the mobile nodes should also be protected by IP Security using ESP. Because the binding update and binding acknowledgement messages exchanged between a correspondent node and a mobile node are protected by the Return Routability procedure, there are no IP Security requirements for the correspondents. But, if a correspondent uses IP Security to restrict its access, the messages with protocol MH (135) must be permitted.

Tunnels can be defined manually or using IKE acting as responder (only aggressive mode is supported). At a minimum, the following IP Security tunnels will be defined on the home agent using the ESP header:

- a tunnel in transport mode with protocol MH (135) between the home agent IP address and the home address of each mobile node susceptible to be registered on this home agent.
- a tunnel in tunnel mode with protocol MH (135) between any IP address and the home address of each mobile node susceptible to be registered on this home agent.

Corresponding tunnels must be defined on the mobile nodes.

Note: The binding update and binding acknowledgement messages are sent using a Mobility Header and must be protected by IP Security using ESP.

In previous implementations of Mobile IPv6 in AIX, support was provided for mobile nodes using Destination Option packets to send binding update messages. These messages could be protected with IP Security using an Authentication Header.

For a home agent or a correspondent node to accept such binding update messages using a Destination Option, edit the `/etc/rc.mobip6` file and enable the **Enable_Draft13_Mobile** variable before starting Mobile **IPv6**. In this case, if you use IP Security to protect the binding update messages, you must define manual or IKE tunnels in transport mode on protocol 60, which will protect the Binding Update and Acknowledgement messages.

For a home agent or a correspondent node to accept binding update messages not protected by IP Security, edit the `/etc/rc.mobip6` file and disable the **Check_IPsec** variable. This method is not recommended because it presents a significant security vulnerability through the ability to affect the routing of packets addressed to a mobile node.

Mobile IPv6 configuration

This introduces information about configuring Mobile **IPv6**. In order to use Mobile **IPv6**, you must first install the `bos.net.mobip6.rte` fileset.

For information about installing filesets, see Installing optional software products and service updates in *Installation and migration*

Starting Mobile IPv6 as a home agent:

Use this procedure to start Mobile **IPv6** as a home agent.

1. Define either IKE tunnels (phases 1 and 2) as responder using the **ESP** protocol or manual ESP IP Security Association between the home agent IP address and each mobile home address the correspondent may communicate with.
2. Enable the system as a Mobile **IPv6** home agent and correspondent node. At the command line, type `smit enable_mobip6_home_agent`.
3. Select when you want it enabled.

Starting Mobile IPv6 as a correspondent:

Use this procedure to start Mobile IPv6 as a correspondent.

1. Define either IKE tunnels (phases 1 and 2) as responder using **ESP** protocol or manual ESP IP Security Association between the home agent IP address and each mobile home address the correspondent may communicate with.
2. Enable the system as a Mobile **IPv6** correspondent node. At the command line, type `smit enable_mobip6_correspondent`.
3. Select when you want it enabled.

Starting Mobile IPv6 as a router:

Use this procedure to start Mobile **IPv6** as a router.

Run the following command to facilitate movement detection:

```
ndpd-router -m
```

Stopping Mobile IPv6:

Use this procedure to stop Mobile **IPv6**.

1. Type `smit disable_mobip6` at the command line.
2. Select when you want Mobile **IPv6** stopped.
3. Select whether you want to stop the **ndpd-router** daemon.
4. Select whether you want to disable **IPv6** forwarding.

Troubleshooting Mobile IPv6

Use the **mobip6ctrl -b** command to troubleshoot Mobile **IPv6**.

1. Get the binding states by running the following:

   ```
   mobip6ctrl -b
   ```
2. See “TCP/IP troubleshooting” on page 393 for information on using the **TCP/IP** troubleshooting utilities.

Virtual IP address

A virtual IP address eliminates a host's dependency upon individual network interfaces.

Incoming packets are sent to the system's VIPA address, but all packets travel through the real network interfaces.

Previously, if an interface failed, any connections to that interface were lost. With VIPA on your system and routing protocols within the network providing automatic reroute, recovery from failures occurs without disruption to the existing user connections that are using the virtual interface as long packets can arrive through another physical interface. Systems running VIPA are more highly available because adapter outages no longer affect active connections. Because multiple physical adapters carry the system IP traffic, overall load is not concentrated on a single adapter and associated subnet.

The AIX VIPA function is transparent to the network equipment. No special network equipment or other hardware is needed. To implement VIPA, you need to have the following items:

- two or more existing IP interfaces of any physical type on different subnets that connect into the corporate network
- IP routing protocols running within the corporate network

Configuring VIPA

VIPA is configured, just as any IP network interface, in SMIT. In addition, you can specify a group of interfaces while configuring VIPA.

When configured this way, for all the outgoing connections initiated by the VIPA host via these interfaces, which are designated to use a VIPA, the virtual address becomes the source address placed in the **TCP/IP** packet header of the outgoing packets.

1. For an IPv4 VIPA, type `smit mkinetvi` on the command line. For an IPv6 VIPA, type `smit mkinetvi6` on the command line.
2. Fill in the required fields. For additional information, see the "Sample VIPA environment." Press Enter.

Adding an adapter to a VIPA

Use this procedure to add an adapter to a virtual IP address.

To add an adapter to your VIPA interface, follow these steps:

1. Type `smit chvi` on the command line.
2. Select the VIPA to which you want to add an adapter and press Enter.
3. Enter the adapter that you want to add in the **Interface Name(s)** field.
4. Type `ADD` in the **ADD/REMOVE interface(s)** field and press Enter.

Removing an adapter from a VIPA

Use this procedure to remove an adapter from a virtual IP address.

To remove an adapter from a VIPA, follow these steps:

1. Type `smit chvi` on the command line.
2. Select the VIPA from which you want to remove an adapter, and press Enter.
3. Enter the adapter that you want to remove in the **Interface Name(s)** field.
4. Type `REMOVE` in the **ADD/REMOVE interface(s)** field and press Enter.

Sample VIPA environment

The following sample VIPA environment with Ethernet connections involves a system with a virtual IP address and two physical connections.

A system has a virtual IP address, `vi0`, of `10.68.6.1` and two physical connections, `en1` with IP address `10.68.1.1` and `en5`, with IP address `10.68.5.1`. In this example, both physical connections are Ethernet, but any mixture of IP interfaces, such as token-ring or FDDI, would be supported as long as the subnets were ultimately attached to the larger corporate network and were known to the corporate routers.

Running the **lsattr -El vi0** command produces the following results:

```
netaddr          10.68.6.1      N/A                                      True
state            up             Standard Ethernet Network Interface      True
netmask          255.255.255.0  Maximum IP Packet Size for This Device    True
netaddr6                        Maximum IP Packet Size for REMOTE Networks True
alias6                          Internet Address                         True
prefixlen                       Current Interface Status                 True
alias4                          TRAILER Link-Level Encapsulation         True
interface_names  en1,en5        Interfaces using the Virtual Address     True
```

Running the **ifconfig vi0** command produces the following results:

```
vi0: flags=84000041<UP,RUNNING,64BIT>
       inet 10.68.6.1 netmask 0xffffff00
       iflist : en1 en5
```

Running the **netstat -rn** command produces the following results:

```
Routing tables
Destination       Gateway       Flags     Refs    Use    If     PMTU Exp Groups

Route Tree for Protocol Family 2 (Internet):
default           10.68.1.2     UG        3       1055   en1     -   -
10.68.1/24        10.68.1.1     U         0       665    en1     -   -
10.68.5/24        10.68.5.1     U         0       1216   en5     -   -
127/8             127.0.0.1     U         4       236    lo0     -   -
10.68.6.1         127.0.0.1     UH        0       0      lo0     -   -
```

The outgoing packets that do not have a source address set and that are routed via interfaces `en1` and `en5` will have the source address set to the virtual address (`10.68.6.1`). Incoming packets are routed to the VIPA address (`10.68.6.1`) advertised on the network. Because `vi0` is virtual (that is, not associated with any device) there should be no entries for it in the system-wide routing table displayed using the **netstat -rn** command. This means no interface route is added when the interface is configured in SMIT.

If one of the physical interfaces, a network attachment, or a network path fails, the network protocols route to the other physical interface on the same system. If a remote system telnets to the `vi0` address, packets to `vi0` can arrive using either `en1` or `en5`. If `en1` is down, for example, packets can still arrive on `en5`. Note that routing protocols might take time to propagate the routes.

When using the VIPA, the end systems and intervening routers must be able to route the packets destined for VIPA (`vi0`) to one of the physical interfaces (`en1` or `en5`).

VIPA versus alias

The VIPA concept is similar to IP aliases except that the addresses are not associated with a hardware interface.

VIPA offers several advantages that IP aliases does not:

- VIPA offers a virtual device that can be brought up and down independently without affecting the physical interfaces
- VIPA addresses can be changed while aliases can only be added or deleted

Access using the IP address of the real adapters

Individual interfaces are still accessible to other systems after VIPA is implemented. However, using the real IP addresses for ping and telnet sessions sidesteps the VIPA advantage of communicating independent of the physical adapters. VIPA hides physical adapter failures from the outlying clients. Using the real addresses reintroduces the dependency upon the physical adapters.

If the remote system contacts the VIPA system using the VIPA address or if an application on the VIPA system initiates the communication to another system, the VIPA address will be used as the source IP address in the packet. However, if the remote system initiates the session using the IP address of the real interface, that real IP address will be the source IP address in the responding packets. There is one exception. For applications that bind to a particular IP interface, the outgoing packets will carry the source address of the interface to which they are bound.

VIPA and routing protocols

The gated daemon was modified for VIPA so that it would not add the interface route or send advertisements over virtual interfaces.

The **OSPF** protocol, supported by gated, will advertise the virtual interface to neighboring routers. The other hosts on the network will be able to talk to the VIPA host through the first-hop router.

Multiple VIPA addresses

Multiple virtual interfaces can be configured. Multiple VIPA interfaces would be useful, for example, if network routers could give preferential treatment to packets sent to or from certain VIPA addresses.

Or, you might use multiple VIPA interfaces if they were binding applications to a specific VIPA interface. For example, to run multiple web servers for multiple companies on a single machine, you could configure the following:

- vi0 200.1.1.1 www.companyA.com
- vi1 200.1.1.2 www.companyB.com
- vi2 200.1.1.3 www.companyC.com

EtherChannel and IEEE 802.3ad Link Aggregation

EtherChannel and IEEE 802.3ad Link Aggregation are network port aggregation technologies that allow several Ethernet adapters to be aggregated together to form a single pseudo Ethernet device.

For example, `ent0` and `ent1` can be aggregated into an EtherChannel adapter called `en3`; interface `en3` would then be configured with an IP address. The system considers these aggregated adapters as one adapter. Therefore, IP is configured over them, as over any Ethernet adapter. In addition, all adapters in the EtherChannel or Link Aggregation are given the same hardware (Mac) address, so they are treated by remote systems as if they were one adapter. Both EtherChannel and IEEE 802.3ad Link Aggregation require support in the switch so that these two technologies are aware which switch ports must be treated as one.

Note: The EtherChannel driver assigns an invalid media access control (MAC) address, `02:00:00:00:00:00`, to the Host Ethernet Adapter (HEA) port of inactive channel of the EtherChannel configuration. This invalid MAC address is assigned when the EtherChannel is created or when the HEA ports are added to the inactive channel at run time. During the EtherChannel failover or recovery, the invalid MAC address is swapped with the valid MAC address, and the valid MAC address is swapped with the invalid MAC address at the run time.

The main benefit of EtherChannel and IEEE 802.3ad Link Aggregation is that they have the network bandwidth of all of their adapters in a single network presence. If an adapter fails, network traffic is automatically sent on the next available adapter without disruption to existing user connections. The adapter is automatically returned to service on the EtherChannel or Link Aggregation when it recovers.

There are some differences between EtherChannel and IEEE 802.3ad Link Aggregation. Consider the differences listed in Table 79 to determine which technology best suits your requirement.

Table 79. Differences between EtherChannel and IEEE 802.3ad Link Aggregation.

EtherChannel	IEEE 802.3ad Link Aggregation
Requires switch configuration.	Requires switch configuration for Link Aggregation Control Protocol Data Unit (LACPDU) exchange.
Heartbeats are not exchanged between the switch port and the adjacent system port.	Heartbeats (LACPDU) are exchanged at the interval that is defined by the IEEE 802.3ad standard. Heartbeats provide extra protection in case of a failure.

Dynamic Adapter Membership functionality is available in the AIX operating system. You can use this functionality to add or remove adapters from an EtherChannel without having to disrupt any user connections.

Related concepts:

"Dynamic Adapter Membership" on page 359
Prior to AIX 5L Version 5.2 with the 5200-03 Recommended Maintenance package, in order to add or remove an adapter from an EtherChannel, its interface first had to be detached, temporarily interrupting all user traffic. To overcome this limitation, Dynamic Adapter Membership (DAM) was added in AIX 5L Version 5.2 with the 5200-03 Recommended Maintenance package.

"EtherChannel" on page 350
The adapters that belong to an EtherChannel must be connected to the same EtherChannel-enabled switch. If the adapters are connected to different switches, those switches must be stacked and act as a

single switch.

"IEEE 802.3ad Link Aggregation configuration" on page 362
IEEE 802.3ad is a standard way of doing link aggregation. Conceptually, it works the same as EtherChannel in that several Ethernet adapters are aggregated into a single virtual adapter, providing greater bandwidth and protection against failures.

"Interoperability scenarios" on page 366
Consider the following interoperability scenarios when configuring your EtherChannel or IEEE 802.3ad Link Aggregation.

EtherChannel

The adapters that belong to an EtherChannel must be connected to the same EtherChannel-enabled switch. If the adapters are connected to different switches, those switches must be stacked and act as a single switch.

You must manually configure this switch to treat the ports that belong to the EtherChannel as an aggregated link. Your switch documentation might refer to this capability as *link aggregation* or *trunking*.

For EtherChannel to work correctly, the link polling mechanism that periodically verifies the status of the link must be enabled on each adapter before the EtherChannel is created. Traffic is distributed across the adapters in either the standard way (where the adapter over which the packets are sent is chosen depending on an algorithm) or on a round-robin basis (where packets are sent evenly across all adapters). Incoming traffic is distributed in accordance to the switch configuration and is not controlled by the EtherChannel operation mode.

You can configure multiple EtherChannels per system. If all links in one EtherChanel are attached to a single switch and if the switch is unplugged or fails, the entire EtherChannel is lost. To solve this problem, a backup option is available that keeps the service active when the main EtherChannel fails. The backup and EtherChannel adapters must be attached to different network switches, which must be interconnected for this setup to work properly. If all adapters in the EtherChannel fail, the backup adapter is used to send and receive all traffic. When any link in the EtherChannel is restored, the service is moved back to the EtherChannel.

For example, `ent0` and `ent1` can be configured as the main EtherChannel adapters, and `ent2` as the backup adapter, creating an EtherChannel called `en3`. Ideally, `ent0` and `ent1` are connected to the same EtherChannel-enabled switch, and `ent2` is connected to a different switch. In this example, all traffic sent over `en3` (the interface of EtherChannel) is sent over `ent0` or `ent1` by default (depending on the packet distribution scheme of EtherChannel), whereas `ent2` is idle. If at any time both `ent0` and `ent1` fail, all traffic is sent over the backup adapter `ent2`. When either `ent0` or `ent1` recover, they are again used for all traffic.

Network Interface Backup, a mode of operation available for EtherChannel, protects against a single point of Ethernet network failure. No special hardware is required to use Network Interface Backup, but the backup adapter must be connected to a separate switch for maximum reliability. In Network Interface Backup mode, only one adapter at a time is actively used for network traffic. The EtherChannel tests the currently active adapter and, optionally, the network path to a user-specified node. When a failure is detected, the next adapter will be used for all traffic. Network Interface Backup provides detection and failover with no disruption to user connections. Network Interface Backup was originally implemented as a mode in the EtherChannel system management interface tool (SMIT) menu. The backup adapter provides the equivalent function, so the mode was eliminated from the SMIT menu. To configure network interface backup, see "Network Interface Backup configuration" on page 355.

EtherChannel configuration considerations

Consult this list of recommendations before configuring EtherChannel.

- You can have up to eight primary Ethernet adapters and up to eight backup Ethernet adapters per EtherChannel.

- You can configure multiple EtherChannels on a single system, but each EtherChannel constitutes an additional Ethernet interface. The **ifsize** option of the **no** command might need to be increased to include not only the Ethernet interfaces for each adapter, but also any EtherChannels that are configured. In AIX 5.2 and earlier, the default **ifsize** is eight. The default size is 256.
- You can use any supported Ethernet adapter in an EtherChannel (see "Supported adapters" on page 366). However, the Ethernet adapters must be connected to a switch that supports EtherChannel. See the documentation that came with your switch to determine if it supports EtherChannel (your switch documentation may refer to this capability also as link aggregation or trunking).
- All adapters in the EtherChannel should be configured for the same speed (100 Mbps, for example) and should be full duplex.
- The adapters used in the EtherChannel cannot be accessed by the system after the EtherChannel is configured. To modify any of their attributes, such as media speed, transmit or receive queue sizes, and so forth, you must do so before including them in the EtherChannel.
- The adapters that you plan to use for your EtherChannel must not have an IP address configured on them before you start this procedure. When configuring an EtherChannel with adapters that were previously configured with an IP address, make sure that their interfaces are in the `detach` state. The adapters to be added to the EtherChannel cannot have interfaces configured in the `up` state in the Object Data Manager (ODM), which will happen if their IP addresses were configured using SMIT. This may cause problems bringing up the EtherChannel when the machine is rebooted because the underlying interface is configured before the EtherChannel with the information found in ODM. Therefore, when the EtherChannel is configured, it finds that one of its adapters is already being used. To change this, before creating the EtherChannel, type `smitty chinet`, select each of the interfaces of the adapters to be included in the EtherChannel, and change its **state** value to `detach`. This will ensure that when the machine is rebooted the EtherChannel can be configured without errors.

  For more information about ODM, see Object Data Manager (ODM) in *General Programming Concepts: Writing and Debugging Programs*.
- If you will be using 10/100 Ethernet adapters in the EtherChannel for AIX versions prior to AIX 5L Version 5.2 with the 5200-03 Recommended Maintenance package, you may need to enable link polling on those adapters before you add them to the EtherChannel. Type `smitty chgenet` at the command line. Change the **Enable Link Polling** value to `yes`, and press Enter.

  Note: In AIX 5L Version 5.2 with the 5200-03 Recommended Maintenance package and later, enabling the link polling mechanism is not necessary. The link poller will be started automatically.
- If you plan to use jumbo frames, you may need to enable this feature in every adapter before creating the EtherChannel and in the EtherChannel itself. Type `smitty chgenet` at the command line. Change the **Enable Jumbo Frames** value to `yes` and press Enter. Do this for every adapter for which you want to enable Jumbo Frames. You will enable jumbo frames in the EtherChannel itself later.

  Note: Enabling jumbo frames in every underlying adapter is not necessary once it is enabled in the EtherChannel itself. The feature will be enabled automatically if you set the **Enable Jumbo Frames** attribute to `yes`.
- AIX 5.3 and AIX 6.1 levels support the following configurations for Host Ethernet Adapters (HEA).
  - Link aggregation between dedicated HEA port and PCI/PCI-E adapter is supported for both manual aggregation and LACP aggregation.
  - EtherChannel configuration that includes non-dedicated HEA port support, EtherChannel with backup adapter configured as PCI/PCI-E, or virtual Ethernet.

    Note: For non-dedicated HEA port in EtherChannel configuration, limitations about Link aggregation apply.
- AIX Version 6.1 with the 6100-06 Technology Level and later supports EtherChannel on stacked switches.
- Network boot or network installation over EtherChannel on Network Installation Management (NIM) client is not supported.

Configuring an EtherChannel

Use this procedure to configure an EtherChannel.

1. Type `smitty etherchannel` at the command line.
2. Select **Add an EtherChannel / Link Aggregation** from the list and press Enter.
3. Select the primary Ethernet adapters that you want on your EtherChannel and press Enter. If you are planning to use EtherChannel backup, do not select the adapter that you plan to use for the backup at this point.

   Note: The **Available Network Adapters** displays all Ethernet adapters. If you select an Ethernet adapter that is already being used (has an interface defined), you will get an error message. You first need to detach this interface if you want to use it.
4. Enter the information in the fields according to the following guidelines:
   - **Parent Adapter:** Provides information of an EtherChannel's parent device (for example, when an EtherChannel belongs to a Shared Ethernet Adapter). This field displays a value of `NONE` if the EtherChannel is not contained within another adapter (the default). If the EtherChannel is contained within another adapter, this field displays the parent adapter's name (for example, `ent6`). This field is informational only and cannot be modified. The parent adapter option is available in AIX 5.3 and later.
   - **EtherChannel / Link Aggregation Adapters:** You should see all primary adapters that you are using in your EtherChannel. You selected these adapters in the previous step.
   - **Enable Alternate Address:** This field is optional. Setting this to `yes` will enable you to specify a MAC address that you want the EtherChannel to use. If you set this option to `no`, the EtherChannel will use the MAC address of the first adapter.
   - **Alternate Address:** If you set **Enable Alternate Address** to `yes`, specify the MAC address that you want to use here. The address you specify must start with `0x` and be a 12-digit hexadecimal address (for example, 0x001122334455).
   - **Enable Gigabit Ethernet Jumbo Frames:** This field is optional. In order to use this, your switch must support jumbo frames. This will only work with a Standard Ethernet (en) interface, not an IEEE 802.3 (et) interface. Set this to `yes` if you want to enable it.
   - **Mode:** You can choose from the following modes:
     - **standard**: In this mode the EtherChannel uses an algorithm to choose which adapter it will send the packets out on. The algorithm consists of taking a data value, dividing it by the number of adapters in the EtherChannel, and using the remainder (using the modulus operator) to identify the outgoing link. The Hash Mode value determines which data value is fed into this algorithm (see the Hash Mode attribute for an explanation of the different hash modes). For example, if the Hash Mode is `standard`, it will use the packet's destination IP address. If this is 10.10.10.11 and there are 2 adapters in the EtherChannel, (1 / 2) = 0 with remainder 1, so the second adapter is used (the adapters are numbered starting from 0). The adapters are numbered in the order they are listed in the SMIT menu. This is the default operation mode.
     - **round_robin**: In this mode the EtherChannel will rotate through the adapters, giving each adapter one packet before repeating. The packets may be sent out in a slightly different order than they were given to the EtherChannel, but it will make the best use of its bandwidth. It is an invalid combination to select this mode with a Hash Mode other than `default`. If you choose the round-robin mode, leave the Hash Mode value as `default`.
     - **netif_backup**: To enable Network Interface Backup Mode, you can configure one or more adapters in the primary EtherChannel and the backup EtherChannel. For more information, see "Configuring Network Interface Backup" on page 356.
     - **8023ad**: This option enables the use of the IEEE 802.3ad Link Aggregation Control Protocol (LACP) for automatic link aggregation. For more details about this feature, see "IEEE 802.3ad Link Aggregation configuration" on page 362.
   - **IEEE 802.3ad Interval**: You can choose from the following values:

- **long**: This is the default value of the interval. When selected, the EtherChannel will request LACP packets from its partner at long interval value as specified by the protocol
- **short**: When selected, the EtherChannel will request LACP packets from its partner at short interval value as specified by the protocol.

  Note: The interval value is only used when EtherChannel is operating in IEEE 802.3ad mode. Otherwise, the value is ignored.

  Note: AIX honors both long and short interval request from its partner.

- **Hash Mode**: Choose from the following hash modes, which will determine the data value that will be used by the algorithm to determine the outgoing adapter:
  - **default**: The destination IP address of the packet is used to determine the outgoing adapter. For non-IP traffic (such as ARP), the last byte of the destination MAC address is used to do the calculation. This mode guarantees packets are sent out over the EtherChannel in the order they were received, but it may not make full use of the bandwidth.
  - **src_port**: The source UDP or TCP port value of the packet is used to determine the outgoing adapter. If the packet is not UDP or TCP traffic, the last byte of the destination IP address will be used. If the packet is not IP traffic, the last byte of the destination MAC address will be used.
  - **dst_port**: The destination UDP or TCP port value of the packet is used to determine the outgoing adapter. If the packet is not UDP or TCP traffic, the last byte of the destination IP will be used. If the packet is not IP traffic, the last byte of the destination MAC address is used.
  - **src_dst_port**: The source and destination UDP or TCP port values of the packet is used to determine the outgoing adapter (specifically, the source and destination ports are added and then divided by two before being fed into the algorithm). If the packet is not UDP or TCP traffic, the last byte of the destination IP is used. If the packet is not IP traffic, the last byte of the destination MAC address will be used. This mode can give good packet distribution in most situations, both for clients and servers.

    Note: It is an invalid combination to select a Hash Mode other than `default` with a Mode of `round_robin`.

  To learn more about packet distribution and load balancing, see "EtherChannel load-balancing options" on page 357.

- **Backup Adapter:** This field is optional. Enter a list of adapters that you want to use as EtherChannel backup.
- **Internet Address to Ping:** This field is optional and takes effect only if you are running **Network Interface Backup** mode or if you have one or more adapters in EtherChannel and one or more adapters in the backup list. EtherChannel ping the IP address or host name that you specify. If EtherChannel is unable to ping the IP address for the number of times specified in the **Number of Retries** field and within the time interval that is specified in the **Retry Timeout** field, EtherChannel changes over to the other adapters in the backup list.
- **Number of Retries:** Enter the number of ping response failures that are allowed before the EtherChannel switches adapters. The default is three. This field is optional and valid only if you have set an **Internet Address to Ping**.
- **Retry Timeout:** Enter the number of seconds between the times when the EtherChannel will ping the **Internet Address to Ping**. The default is one second. This field is optional and valid only if you have set an **Internet Address to Ping**.

5. Press Enter after changing the desired fields to create the EtherChannel.
6. Configure IP over the newly-created EtherChannel device by typing `smitty chinet` at the command line.
7. Select your new EtherChannel interface from the list.
8. Fill in all of the required fields and press Enter.

For additional tasks that can be performed after the EtherChannel is configured, see “Listing EtherChannels or Link Aggregations” on page 359.

Recovery and failover options

Recovery and failover features are available for EtherChannel or IEEE 802.3ad Link Aggregation.

With these features, the following improvements are available:

- packet loss can be prevented during recovery
- failovers can be set to occur instantaneously
- automatic recovery can be turned off so that the backup adapter continues operations
- Link Aggregations can be forced to failover from the primary channel to the backup, or the other way around

Lossless recovery:

The lossless recovery feature ensures that recovery from the backup adapter to the primary channel loses as few packets as possible.

Before lossless recovery, EtherChannel or IEEE 802.3ad would recover to the primary channel at the same instant it detected the recovery of one of the primary adapters. In some cases, the adapter's switch would not be in a state in which it could send or receive data, and some packets would get lost immediately after a recovery.

With lossless recovery, the EtherChannel or IEEE 802.3ad adapter recovers to the primary channel only when it has been able to actually receive traffic on it. This ensures that the switch port is fully initialized and no packets get lost.

Lossless failover:

The lossless failover feature modifies the behavior of the lossless recovery feature.

When ping failures cause a failover, lossless recovery is observed by default. This involves a period of waiting until the inactive adapter's switch receives traffic before finalizing the failover. If the **noloss_failover** attribute is set to `no`, however, ping failovers occur immediately.

Automatic recovery:

After a failover from the primary channel to the backup adapter, EtherChannel and IEEE 802.3ad Link Aggregation automatically starts a recovery to the primary channel that failed when at least one of its adapters recovers.

This recovery option is not supported in IEEE 802.3ad mode and failover to the backup adapter is because of the Link Aggregation Control Protocol (LACP) failure. The LACP failure occurs when all adapters in the primary channel do not receive the LACP data units (LACPDU) within the timeout period. The timeout period is determined by the IEEE standard, which is based on the interval that is configured for IEEE 802.3ad mode.

This default behavior can be modified by setting the **auto_recovery** attribute to `no`. With this setting, the EtherChannel or IEEE 802.3ad Link Aggregation continues operating on the backup adapter after the failover. Operations on the backup adapter continue until one of the following events occur:

- A failover is forced.
- The backup adapter fails.
- A ping failure is detected on the backup adapter.

Forced failovers:

EtherChannel or IEEE 802.3ad Link Aggregation can be forced to fail over from the primary channel to the backup adapter, or from the backup adapter to the primary channel.

Forced failovers work only if there is a backup adapter defined, and if the inactive channel is up and running. For example, to force a failover from the primary channel to the backup adapter, the backup adapter must be running.

To use this feature, enter `smitty etherchannel` and select the **Force A Failover In An EtherChannel / Link Aggregation** option from the screen. Then select the EtherChannel or IEEE 802.3ad Link Aggregation where the failover needs to be forced.

Network Interface Backup configuration

Network Interface Backup protects against a single point of network failure by providing failure detection and failover with no disruption to user connections. When operating in this mode, only one adapter is active at any given time.

If the active adapter fails, another adapter in the EtherChannel will be used for all traffic. When operating in Network Interface Backup mode, it is not necessary to connect to EtherChannel-enabled switches.

The Network Interface Backup setup is most effective when the adapters are connected to different network switches, as this provides greater redundancy than connecting all adapters to one switch. When connecting to different switches, make sure there is a connection between the switches. This provides failover capabilities from one adapter to another by ensuring that there is always a route to the currently-active adapter.

Priority is given to the adapter configured in the primary EtherChannel, over the backup adapter. As long as the primary adapter is functional, it is used. This contrasts from the behavior of Network Interface Backup mode in earlier releases, where the backup adapter was used until it also failed, regardless of whether the primary adapter had already recovered.

For example, `ent0` could be configured as the main adapter, and `ent2` as the backup adapter, creating an EtherChannel called `ent3`. Ideally, `ent0` and `ent2` would be connected to two different switches. In this example, all traffic sent over `ent3` (the EtherChannel's interface) is sent over `ent0` by default, whereas `ent2` is idle. If at any time `ent0` fails, all traffic is sent over the backup adapter, `ent2`. When `ent0` recovers, it is again used for all traffic.

It is now possible to configure the EtherChannel to detect link failure and network unreachability for multiple EtherChannels with a backup adapter. To do this, use the **netaddr** attribute to specify the IP address or host name of a remote host where connectivity should always be present. The EtherChannel will periodically ping this host to determine whether there is still a network path to it. If a specified number of ping attempts go unanswered, the EtherChannel will fail over to the other adapter in the hope that there is a network path to the remote host through the other adapter. In this setup, not only should every adapter be connected to a different switch, but each switch should also have a different route to the host that is pinged.

This ping feature is available for one or more EtherChannels with a backup adapter. However, if there is a failover due to unanswered pings on the primary adapter, the backup adapter remains the active channel as long as it is working. There is no way of knowing, while operating on the backup adapter, whether it is possible to reach the host being pinged from the primary adapter. To avoid failing over back and forth between the primary and the backup, it keeps operating on the backup (unless the pings go unanswered on the backup adapter as well, or if the backup adapter itself fails, in which case it would fail over to the primary adapter). However, if the failover occurred because the primary adapter failed (not because the pings went unanswered), the EtherChannel will then come back to the primary adapter as soon it has come back up, as usual.

To configure Network Interface Backup in the newer versions, see "Configuring Network Interface Backup."

Configuring Network Interface Backup:

Use this procedure to configure a network interface backup in the newer versions.

1. With root authority, type `smitty etherchannel` on the command line.
2. Select **Add an EtherChannel / Link Aggregation** from the list and press Enter.
3. Select the primary Ethernet adapter and press Enter. This is the adapter that will be used until it fails.

   Note: The **Available Network Adapters** field displays all of the Ethernet adapters. If you select an Ethernet adapter that is already being used, you will get an error message and will need to detach this interface before you can use it. Refer to "Making changes to an EtherChannel with 5200-01 and earlier" on page 361 for information on how to detach an interface.
4. Enter the information in the following fields according to the following guidelines:
   - **Parent Adapter:** This field provides information of an EtherChannel's parent device (for example, when an EtherChannel belongs to a Shared Ethernet Adapter). This field displays a value of `NONE` if the EtherChannel is not contained within another adapter (the default). If the EtherChannel is contained within another adapter, this field displays the parent adapter's name (for example, `ent6`). This field is informational only and cannot be modified. The backup adapter option is available in the AIX operating system.
   - **EtherChannel / Link Aggregation Adapters**: You should see the primary adapter you selected in the previous step.
   - **Enable Alternate Address**: This field is optional. Setting this to `yes` will enable you to specify a MAC address that you want the EtherChannel to use. If you set this option to `no`, the EtherChannel will use the MAC address of the primary adapter.
   - **Alternate Address**: If you set **Enable Alternate Address** to `yes`, specify the MAC address that you want to use here. The address you specify must start with `0x` and be a 12-digit hexadecimal address (for example 0x001122334455).
   - **Enable Gigabit Ethernet Jumbo Frames**: This field is optional. In order to use this, your switch must support jumbo frames. This only works with a Standard Ethernet (en) interface, not an IEEE 802.3 (et) interface. Set this to `yes` if you want to use it.
   - **Mode**: It is irrelevant which mode of operation you select because there is only one adapter in the main EtherChannel. All packets are sent over that adapter until it fails. There is no `netif_backup` mode because that mode can be emulated using a backup adapter.
   - **Hash Mode**: It is irrelevant which hash mode you select because there is only one adapter in the main EtherChannel. All packets are sent over that adapter until it fails.
   - **Backup Adapter**: Enter a list of one or more adapters that you want to include in the EtherChannel backup group. After a failover event due to the loss of the primary EtherChannel group, the backup adapters are used until the primary EtherChannel group recovers.
   - **Internet Address to Ping**: This field is optional. The EtherChannel pings the IP address or host name that you specify here. If the EtherChannel is unable to ping this address for the number of times specified in the **Number of Retries** field and in the intervals specified in the **Retry Timeout** field, the EtherChannel switches adapters.
   - **Number of Retries**: Enter the number of ping response failures that are allowed before the EtherChannel switches adapters. The default is three. This field is optional and valid only if you have set an **Internet Address to Ping**.
   - **Retry Timeout**: Enter the number of seconds between the times when the EtherChannel pings the **Internet Address to Ping**. The default is one second. This field is optional and valid only if you have set an **Internet Address to Ping**.
5. Press Enter after changing the desired fields to create the EtherChannel.
6. Configure IP over the newly-created interface by typing `smitty chinet` at the command line.

7. Select your new EtherChannel interface from the list.
8. Fill in all of the required fields and press Enter.

Your network interface backup is now configured.

EtherChannel load-balancing options

There are two load balancing methods for outgoing traffic in EtherChannel, as follows: round-robin, which spreads the outgoing traffic evenly across all of the adapters in the EtherChannel; and standard, which selects the adapter using an algorithm.

The Hash Mode parameter determines the numerical value that is fed to the algorithm.

The following table summarizes the valid load-balancing option combinations offered.

Table 80. Mode and Hash Mode combinations and the outgoing traffic distributions each produces.

Mode	Hash Mode	Outgoing Traffic Distribution
standard or 8023ad	default	The traditional AIX behavior. The adapter selection algorithm uses the last byte of the destination IP address (for **TCP/IP** traffic) or MAC address (for ARP and other non-IP traffic). This mode is typically a good initial choice for a server with a large number of clients.
standard or 8023ad	src_dst_port	The outgoing adapter path is selected by an algorithm using the combined source and destination TCP or UDP port values. Because each connection has a unique TCP or UDP port, the three port-based hash modes provide additional adapter distribution flexibility when there are several, separate TCP or UDP connections between an IP address pair.
standard or 8023ad	src_port	The adapter selection algorithm uses the source TCP or UDP port value. In the **netstat -an** command output, the port is the TCP/IP address suffix value in the `Local` column.
standard or 8023ad	dst_port	The outgoing adapter path is selected by the algorithm using the destination system port value. In the **netstat -an** command output, the TCP/IP address suffix in the `Foreign` column is the TCP or UDP destination port value.
round-robin	default	Outgoing traffic is spread evenly across all of the adapter ports in the EtherChannel. This mode is the typical choice for two hosts connected back-to-back (without an intervening switch).

Round-Robin distribution:

All outgoing traffic is spread evenly across all of the adapters in the EtherChannel. It provides the highest bandwidth optimization for the AIX server system. While round-robin distribution is the ideal way to use all of the links equally, consider that it also introduces the potential for out-of-order packets at the receiving system.

In general, round-robin mode is ideal for back-to-back connections running jumbo frames. In this environment, there is no intervening switch, so there is no chance that processing at the switch could alter the packet delivery time, order, or adapter path. On this direct cable network path, packets are

received exactly as sent. Jumbo frames (9000 byte MTU) always yield better file transfer performance than traditional 1500 byte MTUs. In this case, however, they add another benefit. These larger packets take longer to send so it is less likely that the receiving host would be continuously interrupted with out-of-order packets.

Round-robin mode can be implemented in other environments but at increased risk of out-of-order packets at the receiving system. This risk is particularly high when there are few, long-lived, streaming TCP connections. When there are many such connections between a host pair, packets from different connections could be intermingled, thereby decreasing the chance of packets for the same connection arriving out-of-order. Check for out-of-order packet statistics in the `tcp` section of the **netstat -s** command output. A steadily-increasing value indicates a potential problem in traffic sent from an EtherChannel.

If out-of-order packets are a problem on a system that must use traditional Ethernet MTUs and must be connected through a switch, try the various hash modes offered in standard mode operation. Each mode has a particular strength, but the default and src_dst_port modes are the logical starting points as they are more widely applicable.

Standard or 802.3ad algorithm:

There are advantages to using the EtherChannel standard algorithm.

The standard algorithm is used for both standard and IEEE 802.3ad-style link aggregations. AIX divides the last byte of the "numerical value" by the number of adapters in the EtherChannel and uses the remainder to identify the outgoing link. If the remainder is zero, the first adapter in the EtherChannel is selected; a remainder of one means the second adapter is selected, and so on (the adapters are selected in the order that they are listed in the **adapter_names** attribute).

The Hash Mode selection determines the numerical value used in the calculation. By default, the last byte of the destination IP address or MAC address is used in the calculation, but the source and destination TCP or UDP port values can also be used. These alternatives allow you to fine-tune the distribution of outgoing traffic across the real adapters in the EtherChannel.

In default hash mode, the adapter selection algorithm is applied to the last byte of the destination IP address for IP traffic. For ARP and other non-IP traffic, the same formula is applied on the last byte of the destination MAC address. Unless there is an adapter failure that causes a failover, all traffic between a host pair in default standard mode goes out over the same adapter. The default hash mode may be ideal when the local host establishes connections to many different IP addresses.

If the local host establishes lengthy connections to few IP addresses, however, you will notice that some adapters carry a greater load than others, because all of the traffic sent to a specific destination is sent over the same adapter. While this prevents packets from arriving out-of-order, it may not use bandwidth in the most effective fashion in all cases. The port-based hash modes still send packets in order, but they allow packets belonging to different UDP or TCP connections, even if they are sent to the same destination, to be sent over different adapters, thus better using the bandwidth of all of the adapters.

In **src_dst_port** hash mode, the TCP or UDP source and destination port values of the outgoing packet are added, then divided by two. The resultant whole number (no decimals) is plugged into the standard algorithm. TCP or UDP traffic is sent on the adapter selected by the standard algorithm and selected hash mode value. Non-TCP or UDP traffic will fall back to the default hash mode, meaning the last byte of either the destination IP address or MAC address. The **src_dst_port** hash mode option considers both the source and the destination TCP or UDP port values. In this mode, all of the packets in one TCP or UDP connection are sent over a single adapter so they are guaranteed to arrive in order, but the traffic is still spread out because connections (even to the same host) may be sent over different adapters. The results of this hash mode are not skewed by the connection establishment direction because it uses both the source and destination TCP or UDP port values.

In **src_port** hash mode, the source TCP or UDP port value of the outgoing packet is used. In **dst_port** hash mode, the destination TCP or UDP port value of the outgoing packet is used. Use the **src_port** or **dst_port** hash mode options if port values change from one connection to another and if the **src_dst_port** option is not yielding a desirable distribution.

Listing EtherChannels or Link Aggregations

Use this procedure to list EtherChannels or Link Aggregations.

1. On the command line, type `smitty etherchannel`.
2. Select **List All EtherChannels / Link Aggregations** and press Enter.

Changing the alternate address

To specify a MAC address for your EtherChannel or Link Aggregation, follow these steps.

1. Depending on which version of AIX you are running, you might have to detach the interface:
   - On AIX 5.2 with 5200-01 and earlier, type `smitty chinet` and select the interface belonging to your EtherChannel. Change the **Current STATE** attribute to **detach**, and press Enter.
   - On AIX 5L Version 5.2 with the 5200-03 Recommended Maintenance package and later, you can change the alternate address of the EtherChannel without detaching its interface.
2. On the command line, type `smitty etherchannel`.
3. Select **Change / Show Characteristics of an EtherChannel** and press Enter.
4. If you have multiple EtherChannels, select the EtherChannel for which you want to create an alternate address.
5. Change the value in **Enable Alternate EtherChannel Address** to `yes`.
6. Enter the alternate address in the **Alternate EtherChannel Address** field. The address must start with `0x` and be a 12-digit hexadecimal address (for example, 0x001122334455).
7. Press Enter to complete the process.

   Note: Changing the EtherChannel's MAC address at runtime can cause a temporary loss of connectivity. This is because the adapters need to be reset so they learn the new hardware address, and some adapters take a few seconds to be initialized.

Dynamic Adapter Membership

Prior to AIX 5L Version 5.2 with the 5200-03 Recommended Maintenance package, in order to add or remove an adapter from an EtherChannel, its interface first had to be detached, temporarily interrupting all user traffic. To overcome this limitation, Dynamic Adapter Membership (DAM) was added in AIX 5L Version 5.2 with the 5200-03 Recommended Maintenance package.

It allows adapters to be added or removed from an EtherChannel without having to disrupt any user connections. A backup adapter can also be added or removed; an EtherChannel can be initially created without a backup adapter, and one can be added at a later date if the need arises.

Not only can adapters be added or removed without disrupting user connections, it is also possible to modify most of the EtherChannel attributes at runtime. For example, you may begin using the "ping" feature of Network Interface Backup while the EtherChannel is in use, or change the remote host being pinged at any point.

You can also turn a regular EtherChannel into an IEEE 802.3ad Link Aggregation (or the other way around), allowing users to experiment with this feature without having to remove and recreate the EtherChannel.

Furthermore, with DAM, you may choose to create a one-adapter EtherChannel. A one-adapter EtherChannel behaves exactly like a regular adapter; however, should this adapter ever fail, it is possible to replace it at runtime without ever losing connectivity. To accomplish this, you add a temporary adapter to the EtherChannel, remove the defective adapter from the EtherChannel, replace the defective

adapter with a working one using Hot Plug, add the new adapter to the EtherChannel, and then remove the temporary adapter. During this process you never notice a loss in connectivity. If the adapter had been working as a standalone adapter, however, it must be detached before being removed using Hot Plug, and during that time any traffic going over it is lost.

Adapter additions, removals, or changes in an EtherChannel or Link Aggregation

There are two ways to add, remove, or change an adapter in an EtherChannel or Link Aggregation.

One method requires the EtherChannel or Link Aggregation interface to be detached, while the other does not (using Dynamic Adapter Membership, which is available in AIX 5L Version 5.2 with the 5200-03 Recommended Maintenance package and later).

Making changes to an EtherChannel using Dynamic Adapter Membership:

Making changes using Dynamic Adapter Membership does not require you to stop all traffic going over the EtherChannel by detaching its interface.

Consider the following before proceeding:

1. When adding an adapter at runtime, note that different Ethernet adapters support different capabilities (for example, the ability to do checksum offload, to use private segments, to do large sends, and so forth). If different types of adapters are used in the same EtherChannel, the capabilities reported to the interface layer are those supported by all of the adapters (for example, if all but one adapter supports the use of private segments, the EtherChannel will state it does not support private segments; if all adapters do support large send, the channel will state it supports large send). When adding an adapter to an EtherChannel at runtime, be sure that it supports at least the same capabilities as the other adapters already in the EtherChannel. If you attempt to add an adapter that does not support all of the capabilities the EtherChannel supports, the addition will fail. Note, however, that if the EtherChannel's interface is detached, you may add any adapter (regardless of the capabilities it supports), and when the interface is reactivated the EtherChannel will recalculate the capabilities it supports based on the new list of adapters.
2. If you are not using an alternate address and you plan to delete the adapter whose MAC address was used for the EtherChannel (the MAC address used for the EtherChannel is "owned" by one of the adapters), the EtherChannel will use the MAC address of the next adapter available. In other words, the one that becomes the first adapter after the deletion, or the backup adapter in case all main adapters are deleted. For example, if an EtherChannel has main adapters `ent0` and `ent1` and backup adapter `ent2`, it uses by default `ent0`'s MAC address (it is then said that `ent0` "owns" the MAC address). If `ent0` is deleted, the EtherChannel then uses `ent1`'s MAC address. If `ent1` is then deleted, the EtherChannel uses `ent2`'s MAC address. If `ent0` is later re-added to the EtherChannel, it continues to use `ent2`'s MAC address because `ent2` is now the owner of the MAC address. If `ent2` were then deleted from the EtherChannel, it starts using `ent0`'s MAC address again.

   Deleting the adapter whose MAC address was used for the EtherChannel may cause a temporary loss of connectivity, because all of the adapters in the EtherChannel need to be reset so they learn of their new hardware address. Some adapters take a few seconds to be initialized.

   If your EtherChannel is using an alternate address (a MAC address you specified), it keeps using this MAC address regardless of which adapters are added or deleted. Furthermore, it means that there is no temporary loss of connectivity when adding or deleting adapters because none of the adapters "owns" the EtherChannel's MAC address.
3. Almost all EtherChannel attributes can now be modified at runtime. The only exception is **Enable Gigabit Ethernet Jumbo Frames**. To modify the **Enable Gigabit Ethernet Jumbo Frames** attribute, you must first detach the EtherChannel's interface before attempting to modify this value.
4. For any attribute that cannot be changed at runtime (currently, only **Enable Gigabit Ethernet Jumbo Frames**), there is a field called **Apply change to DATABASE only**. If this attribute is set to `yes`, it is possible to change, at runtime, the value of an attribute that usually cannot be modified at runtime. With the **Apply change to DATABASE only** field set to `yes` the attribute will only be changed in the ODM and will not be reflected in the running EtherChannel until it is reloaded into memory (by

detaching its interface, using `rmdev -l EtherChannel_device` and then `mkdev -l EtherChannel_device` commands), or until the machine is rebooted. This is a convenient way of making sure that the attribute is modified the next time the machine boots, without having to disrupt the running EtherChannel.

5. In a logical partition, when you remove an adapter from an EtherChannel, you must also remove the associated switch port from the EtherChannel in the switch. Otherwise, the connectivity might be lost because the switch might be using the same switch port for communication.

To make changes to the EtherChannel or Link Aggregation using Dynamic Adapter Membership, follow these steps:

1. At the command line, type `smitty etherchannel`.
2. Select **Change / Show Characteristics of an EtherChannel / Link Aggregation**.
3. Select the EtherChannel or Link Aggregation that you want to modify.
4. Fill in the required fields according to the following guidelines:
   - In the **Add adapter** or **Remove adapter** field, select the Ethernet adapter you want to add or remove.
   - In the **Add backup adapter** or **Remove backup adapter** fields, select the Ethernet adapter you want to start or stop using as a backup.
   - Almost all of the EtherChannel attributes can be modified at runtime, although the **Enable Gigabit Ethernet Jumbo Frames** attribute cannot.
   - To turn a regular EtherChannel into an IEEE 802.3ad Link Aggregation, change the **Mode** attribute to `8023ad`. To turn an IEEE 802.3ad Link Aggregation into an EtherChannel, change the **Mode** attribute to `standard` or `round_robin`.
5. Fill in the necessary data, and press Enter.

Making changes to an EtherChannel with 5200-01 and earlier:

Use this procedure to detach the interface and make changes to an EtherChannel with 5200-01 and earlier.

1. Type `smitty chinet` and select the interface belonging to your EtherChannel. Change the **Current STATE** attribute to **detach**, and press Enter.
2. On the command line type, `smitty etherchannel`.
3. Select **Change / Show Characteristics of an EtherChannel / Link Aggregation** and press Enter.
4. Select the EtherChannel or Link Aggregation that you want to modify.
5. Modify the attributes you want to change in your EtherChannel or Link Aggregation and press Enter.
6. Fill in the necessary fields and press Enter.

Removing an EtherChannel or Link Aggregation:

Use this procedure to remove an EtherChannel or Link Aggregation.

1. Type `smitty chinet` and select the interface belonging to your EtherChannel. Change the **Current STATE** attribute to **detach**, and press Enter.
2. On the command line type `smitty etherchannel`.
3. Select **Remove an EtherChannel** and press Enter.
4. Select the EtherChannel that you want to remove and press Enter.

Configuring or removing a backup adapter on an existing EtherChannel or Link Aggregation:

The following procedure configures or removes a backup adapter on an EtherChannel or Link Aggregation.

1. Type `smitty chinet` and select the interface belonging to your EtherChannel. Change the **Current STATE** attribute to **detach**, and press Enter.

2. On the command line, type `smitty etherchannel`.
3. Select **Change / Show Characteristics of an EtherChannel / Link Aggregation**.
4. Select the EtherChannel or Link Aggregation that you are adding or modifying the backup adapter on.
5. Enter the adapter that you want to use as your backup adapter in the **Backup Adapter** field, or select **NONE** if you wish to stop using the backup adapter.

IEEE 802.3ad Link Aggregation configuration

IEEE 802.3ad is a standard way of doing link aggregation. Conceptually, it works the same as EtherChannel in that several Ethernet adapters are aggregated into a single virtual adapter, providing greater bandwidth and protection against failures.

For example, `ent0` and `ent1` can be aggregated into an IEEE 802.3ad Link Aggregation called `ent3`; interface `ent3` would then be configured with an IP address. The system considers these aggregated adapters as one adapter. Therefore, IP is configured over them as over any Ethernet adapter.

IEEE 802.3ad requires support in the switch.

The advantages of using IEEE 802.3ad Link Aggregation instead of EtherChannel are that you can use switches that support the IEEE 802.3ad standard but do not support EtherChannel and it provides protection against adapter failures.

When an IEEE 802.3ad aggregation is configured, link aggregation control protocol data units (LACPDUs) are exchanged between the server machine (host system) and the adjacent switch. Only the active channel, which could be either the primary channel or the backup adapter, exchanges LACPDU with the adjacent switch.

To be able to aggregate adapters (meaning that the switch allows them to belong to the same aggregation) they must be of the same line speed (for example, all 100 Mbps, or all 1 Gbps) and they must all be full duplex. If you attempt to place adapters of different line speeds or different duplex modes, the creation of the aggregation on the AIX system succeeds, but the switch might not aggregate the adapters together. If the switch does not successfully aggregate the adapters together, you might notice a decrease in network performance. For information on how to determine whether an aggregation on a switch was successful, see “IEEE 802.3ad Link Aggregation troubleshooting” on page 364.

According to the IEEE 802.3ad specification, packets that are being sent to the same IP address are sent over the same adapter. Thus, when being operated in `802.3ad` mode, the packets are always distributed in the standard fashion, never in a round-robin fashion.

The backup adapter feature is available for IEEE 802.3ad Link Aggregations just as it is for EtherChannel. The backup adapter also follows the IEEE 802.3ad LACP. The switch port connected to the backup adapter must also have IEEE 802.3ad enabled.

Note: The steps to enable the use of IEEE 802.3ad varies from switch to switch. You must consult the documentation for your switch to determine what initial steps, if any, must be performed to enable LACP in the switch.

For information in how to configure an IEEE 802.3ad aggregation, see “Configuring IEEE 802.3ad Link Aggregation” on page 363.

Consider the following before you configure an IEEE 802.3ad Link Aggregation:

- Although not officially supported, the AIX implementation of IEEE 802.3ad allows the Link Aggregation to contain adapters of different line speeds. However, you must aggregate only those adapters that are set to the same line speed and are set to full duplex. This helps to avoid potential

problems in configuring the Link Aggregation on the switch. For more information on the types of aggregation allowed by your switch, see the documentation of your switch.

- If you are using 10/100 Ethernet adapters in the Link Aggregation, you must enable link polling on those adapters before you add them to the aggregation. Type `smitty chgenet` at the command line. Change the **Enable Link Polling** value to `yes`, and press Enter. Perform this action for every 10/100 Ethernet adapter that you are adding to your Link Aggregation.

  Note: In AIX 5L Version 5.2 with the 5200-03 Recommended Maintenance package and later, enabling the link polling mechanism is not necessary. The link poller is started automatically.

Configuring IEEE 802.3ad Link Aggregation:

Follow these steps to configure an IEEE 802.3ad Link Aggregation.

1. Type `smitty etherchannel` at the command line.
2. Select **Add an EtherChannel / Link Aggregation** from the list and press Enter.
3. Select the primary Ethernet adapters that you want on your Link Aggregation and press Enter. If you are planning to use a backup adapter, do not select the adapter that you plan to use for the backup at this point.

   Note: The **Available Network Adapters** displays all Ethernet adapters. If you select an Ethernet adapter that is already being used (has an interface defined), you will get an error message. You first need to detach these interfaces if you want to use them.
4. Enter the information in the fields according to the following guidelines:
   - **Parent Adapter:** Provides information about an EtherChannel's parent device (for example, when an EtherChannel belongs to a Shared Ethernet Adapter). This field displays a value of `NONE` if the EtherChannel is not contained within another adapter (the default). If the EtherChannel is contained within another adapter, this field displays the parent adapter's name (for example, `ent6`). This field is informational only and cannot be modified. The parent adapter option is available in AIX 5.3 and later.
   - **EtherChannel / Link Aggregation Adapters:** You should see all primary adapters that you are using in your Link Aggregation. You selected these adapters in the previous step.
   - **Enable Alternate Address:** This field is optional. Setting this to `yes` enables you to specify a MAC address that you want the Link Aggregation to use. If you set this option to `no`, the Link Aggregation uses the MAC address of the first adapter.
   - **Alternate Address:** If you set **Enable Alternate Address** to `yes`, specify the MAC address that you want to use here. The address you specify must start with `0x` and be a 12-digit hexadecimal address (for example, 0x001122334455).
   - **Enable Gigabit Ethernet Jumbo Frames:** This field is optional. In order to use this, your switch must support jumbo frames. This only works with a Standard Ethernet (en) interface, not an IEEE 802.3 (et) interface. Set this to `yes` if you want to enable it.
   - **Mode:** Enter `8023ad`.
   - **Hash Mode**: You can choose from the following hash modes, which determines the data value to be used by the algorithm to determine the outgoing adapter:
     - **default**: In this hash mode the destination IP address of the packet will be used to determine the outgoing adapter. For non-IP traffic (such as ARP), the last byte of the destination MAC address is used to do the calculation. This mode will guarantee packets are sent out over the EtherChannel in the order they were received, but it may not make full use of the bandwidth.
     - **src_port**: The source UDP or TCP port value of the packet is used to determine the outgoing adapter. If the packet is not UDP or TCP traffic, the last byte of the destination IP address is used. If the packet is not IP traffic, the last byte of the destination MAC address is used.

– **dst_port**: The destination UDP or TCP port value of the packet is used to determine the outgoing adapter. If the packet is not UDP or TCP traffic, the last byte of the destination IP is used. If the packet is not IP traffic, the last byte of the destination MAC address is used.
– **src_dst_port**: Both the source and destination UDP or TCP port values of the packet are used to determine the outgoing adapter (specifically, the source and destination ports are added and then divided by two before being fed into the algorithm). If the packet is not UDP or TCP traffic, the last byte of the destination IP is used. If the packet is not IP traffic, the last byte of the destination MAC address is used. This mode can give good packet distribution in most situations, both for clients and servers.

To learn more about packet distribution and load balancing, see "EtherChannel load-balancing options" on page 357.

- **Backup Adapter:** This field is optional. Enter the adapter that you want to use as your backup.
- **Internet Address to Ping:** This field is optional and is available if you have one or more adapters in the main aggregation and a backup adapter. The Link Aggregation pings the IP address or host name that you specify here. If the Link Aggregation is unable to ping this address for the number of times specified by the **Number of Retries** field and in the intervals specified by the **Retry Timeout** field, the Link Aggregation switches adapters.
- **Number of Retries:** Enter the number of ping response failures that are allowed before the Link Aggregation switches adapters. The default is three. This field is optional and valid only if you have set an **Internet Address to Ping**.
- **Retry Timeout:** Enter the number of seconds between the times when the Link Aggregation pings the **Internet Address to Ping**. The default is one second. This field is optional and valid only if you have set an **Internet Address to Ping**.

5. Press Enter after changing the desired fields to create the Link Aggregation.
6. Configure IP over the newly-created Link Aggregation device by typing `smitty chinet` at the command line.
7. Select your new Link Aggregation interface from the list.
8. Fill in all the required fields and press Enter.

IEEE 802.3ad Link Aggregation troubleshooting:

Use the **entstat** command to troubleshoot IEEE 802.3ad Link Aggregation.

If you are having trouble with your IEEE 802.3ad Link Aggregation, use the following command to verify the mode of operation of the Link Aggregation:

```
entstat -d device
```

where *device* is the Link Aggregation device.

This will also make a best-effort determination of the status of the progress of LACP based on the LACPDUs received from the switch. The following status values are possible:

- `Inactive`: LACP has not been initiated. This is the status when a Link Aggregation has not yet been configured, either because it has not yet been assigned an IP address or because its interface has been detached.
- `Negotiating`: LACP is in progress, but the switch has not yet aggregated the adapters. If the Link Aggregation remains on this status for longer than one minute, verify that the switch is correctly configured. For instance, you should verify that LACP is enabled on the ports.
- `Aggregated`: LACP has succeeded and the switch has aggregated the adapters together.
- `Failed`: LACP has failed. Some possible causes are that the adapters in the aggregation are set to different line speeds or duplex modes or that they are plugged into different switches. Verify the adapters' configuration.

In addition, some switches allow only contiguous ports to be aggregated and may have a limitation on the number of adapters that can be aggregated. Consult the switch documentation to determine any limitations that the switch may have, then verify the switch configuration.

Note: The Link Aggregation status is a diagnostic value and does not affect the AIX side of the configuration. This status value was derived using a best-effort attempt. To debug any aggregation problems, it is best to verify the switch's configuration.

The following IEEE 802.3ad Link Aggregation statistics represent the status of the LACP on each port of the aggregation.

They are shown for both the Actor (the IEEE 802.3ad Link Aggregation) and for the Partner (the switch port).

`System Priority:` priority value for this system
`System:` value that uniquely identifies this system
`Operational Key:` value that denotes which ports may be
aggregated together
`Port Priority:` priority value for this port
`Port:` unique value that identifies this port in the aggregation
State:
`LACP activity:` Active or Passive - whether to initiate
sending LACPDUs always, or only in response to
another LACPDU: the IEEE 802.3ad Link Aggregation
will always operate in Active mode
`LACP timeout:` Long or Short - time to wait before
sending LACPDUs: the IEEE 802.3ad Link Aggregation
will always use the long timeout
`Aggregation:` Individual or Aggregatable - whether this
port can form an aggregation with other ports, or
if it can only form an aggregation with itself:
the port in a one-adapter IEEE 802.3ad Link Aggregation
will be marked as Individual, or Aggregatable
if there are more than one ports
`Synchronization:` IN_SYNC or OUT_OF_SYNC - whether the
aggregation has determined it has reached
synchronization with the partner
`Collecting:` Enabled or Disabled - whether the IEEE 802.3ad
Link Aggregation is collecting (receiving) packets
`Distributing:` Enabled or Disabled - whether the IEEE 802.3ad
Link Aggregation is distributing (sending) packets
`Defaulted:` True or False - whether the IEEE 802.3ad
Link Aggregation is using default values for
the partner's information
`Expired:` True or False - whether the IEEE 802.3ad
Link Aggregation is operated in expired mode

The following statistics are shown on both a port-by-port and aggregate basis:

`Received LACPDUs:` LACPDU packets received
`Transmitted LACPDUs:` LACPDU packets sent out
`Received marker PDUs:` marker PDUs received
`Transmitted marker PDUs:` marker PDUs sent:
this version of the protocol does not implement the
marker protocol, so this statistic will always be zero
`Received marker response PDUs:` marker PDUs received

`Transmitted marker response PDUs:` marker response PDUs sent:
this version of the protocol does not implement the marker protocol, so this statistic will always be zero
`Received unknown PDUs:` PDUs received of unknown type
`Received illegal PDUs:` PDUs received of known type, but which were malformed, of unexpected length, or unknown subtype

Interoperability scenarios

Consider the following interoperability scenarios when configuring your EtherChannel or IEEE 802.3ad Link Aggregation.

Additional explanation of each scenario is given after the table.

Table 81. Different AIX and switch configuration combinations and the results each combination will produce.

EtherChannel mode	Switch configuration	Result
8023ad	IEEE 802.3ad LACP	OK - AIX initiates LACPDUs, which triggers an IEEE 802.3ad Link Aggregation on the switch.
standard or round_robin	EtherChannel	OK - Results in traditional EtherChannel behavior.
8023ad	EtherChannel	Undesirable - AIX and Switch cannot aggregate. AIX initates LACPDUs, but the switch ignores them and do not send LACPDUs to AIX. Due to missing LACPDU, AIX does not distribute packets on the link/port. The result is loss of network connectivity.
standard or round_robin	IEEE 802.3ad LACP	Undesirable - Switch cannot aggregate. The result might be from poor performance as the switch moves the MAC address between switch ports

A brief description of each configuration combination follows:

- 8023ad with EtherChannel :

  In this case, AIX will send LACPDUs, but they will go unanswered because the switch is operating as an EtherChannel. As a result, due to missing LACPDU, AIX will not use the link/port for packets distribution. This will cause loss of network connectivity.

  Note: In this case, the `entstat -d` command will always report the aggregation is in the Negotiating state. Also, in the entstat output, the section IEEE 802.3ad Port Statistics will show the **Distributing** is disabled for the **Actor**.
- standard or round_robin with EtherChannel:

  This is the most common EtherChannel configuration.
- standard or round_robin with IEEE 802.3ad LACP:

  This setup is invalid. If the switch is using LACP to create an aggregation, the aggregation will never happen because AIX will never reply to LACPDUs. For this to work correctly, `8023ad` should be the mode set on AIX.

Supported adapters

EtherChannel and IEEE 802.3ad Link Aggregation are supported on IBM Power Systems™ Peripheral Component Interconnect-X (PCI-X) and PCI Express (PCIe) Ethernet adapters.

Additional considerations are as follows:

- Virtual I/O Ethernet Adapter

  Virtual I/O Ethernet Adapters are supported in only two possible EtherChannel configurations:

- One Virtual I/O Ethernet Adapter as the primary, one Virtual I/O Ethernet Adapter as the backup. In this configuration, the **Internet Address to Ping** attribute must be enabled so that the EtherChannel can detect remote connectivity failures. For Virtual I/O Server (VIOS) 2.2.3.0, or later, and AIX Version 7.1 with Technology Level 3, or later, you can use the virtual Ethernet uplink status feature to detect the serving VIOS or Shared Ethernet Adapter (SEA) failure by setting the **poll_uplink** attribute of the virtual Ethernet device to `yes`.
- One supported physical Ethernet adapter as the primary, one Virtual I/O Ethernet Adapter as the backup. In this configuration, the **Internet Address to Ping** attribute must be enabled so that the EtherChannel can detect remote connectivity failures.

- Host Ethernet Adapter (HEA)

  HEA logical ports are supported under EtherChannel if all adapters within the EtherChannel are HEA logical ports. For dedicated HEA port, link aggregation with the PCI/PCI-E adapter is supported. In addition, a PCI/PCI-E and virtual Ethernet adapter as a backup adapter is also supported (when the primary adapter contain HEA).

  When using multiple HEA logical ports as primary adapters in an EtherChannel, the physical ports associated with the HEA logical ports must also be placed in an EtherChannel in the Ethernet switch. Consequently, all partitions that use HEA logical ports going to the same HEA physical ports must also be placed in an EtherChannel.

  For example, assume that Partition 1 is configured as follows:
  - A logical HEA port out of physical HEA port 0
  - A logical HEA port out of physical HEA port 1
  - An EtherChannel created using the logical HEA ports listed above

  If another partition on the same system needs to use a logical HEA port out of physical HEA port 0 or out of physical HEA port 1, you must create an EtherChannel for the partition over both of the logical HEA ports, similar to the configuration of Partition 1. Attempting to use either of those logical HEA ports as stand-alone ports in other partitions might cause connectivity problems because packets might not be delivered to the correct logical HEA port.

  The restriction does not exist when using logical HEA ports in a Network interface backup configuration (1 primary and 1 backup), because the physical HEA ports do not require specific configuration on the Ethernet switch.

  Note: If logical ports from physical HEA ports are configured as part of LACP aggregation (802.3ad), then those physical ports must be exclusive to that LPAR. HMC does not prevent the ports from being assigned to other LPARs, but does not support the configuration.

- Fibre Channel over Ethernet converged network adapters

  Link aggregation between a shared port (a port that is used for both Ethernet and Fiber Channel traffic) and other supported adapters is only supported if the switch that is connected to the shared port can support the link aggregation without impacting the Fiber Channel traffic.

- Single Root I/O Virtualization (SR-IOV) adapters

  Link aggregation with SR-IOV logical ports can be approached by using one of the following methods:
  - IEEE 802.3ad Link Aggregation, also known as Link Aggregation Control Protocol (LACP)
  - Network Interface Backup (NIB)
  - Both LACP and NIB

  For network applications where bandwidth of more than a single port is required, IEEE 802.3ad Link Aggregation can be used to aggregate multiple SR-IOV logical ports. An SR-IOV logical port that is aggregated by using IEEE 802.3ad Link Aggregation must be the only logical port that is configured for the physical port. Multiple SR-IOV logical ports that are configured for the same physical port, where one of the multiple SR-IOV logical ports is configured to be part of an IEEE 802.3ad Link Aggregation configuration, might not be managed properly by the switch because more than one LACP partner might be communicating across the physical port. To prevent configuration of a second SR-IOV logical

port on the same physical port as SR-IOV logical port in an IEEE 802.3ad Link Aggregation configuration, the capacity value of the logical port must be set to 100 (100%) when the logical port is configured.

For network applications where bandwidth of less than a single port is required along with protection against a single network failure, SR-IOV logical ports can be a part of NIB configuration. When an SR-IOV logical port is configured as either a primary or backup adapter in an NIB configuration, the physical port can be shared by other SR-IOV logical ports. In this configuration, the **Internet Address to Ping** attribute can be enabled to detect remote connectivity failures. SR-IOV logical ports can be the primary or backup adapter for another SR-IOV logical port, a virtual Ethernet adapter, or a physical adapter port.

For additional release information about new adapters, see the AIX Release Notes that correspond to your level of AIX.

Important: Mixing adapters of different speeds in the same EtherChannel, even if one of them is operating as the backup adapter, is not supported. This does *not* mean that such configurations will not work. The EtherChannel driver makes every reasonable attempt to work even in a mixed-speed scenario.

Related information:

Single root I/O virtualization

EtherChannel troubleshooting

If you are having trouble with your EtherChannel, there are many scenarios to consider.

You can use tracing and statistics to help diagnose the problem, which can involve issues with failover and jumbo frames.

EtherChannel tracing:

Use **tcpdump** and **iptrace** to troubleshoot the EtherChannel.

The trace hook ID for the transmission packets is 2FA and for other events is 2FB. You cannot trace receive packets on the EtherChannel as a whole, but you can trace each adapter's receive trace hooks.

EtherChannel statistics:

Use the **entstat** command to get the aggregate statistics of all of the adapters in the EtherChannel.

For example, `entstat ent3` will display the aggregate statistics of ent3. Adding the **-d** flag will also display the statistics of each adapter individually. For example, typing `entstat -d ent3` will show you the aggregate statistics of the EtherChannel as well as the statistics of each individual adapter in the EtherChannel.

Note: In the `General Statistics` section, the number shown in `Adapter Reset Count` is the number of failovers. In EtherChannel backup, coming back to the main EtherChannel from the backup adapter is not counted as a failover. Only failing over from the main channel to the backup is counted.

In the `Number of Adapters` field, the backup adapter is counted in the number displayed.

Slow failover:

If the failover time when you are using network interface backup mode or EtherChannel backup is slow, verify that your switch is not running the Spanning Tree Protocol (STP).

When the switch detects a change in its mapping of switch port to MAC address, it runs the spanning tree algorithm to see if there are any loops in the network. Network Interface Backup and EtherChannel backup may cause a change in the port to MAC address mapping.

Switch ports have a forwarding delay counter that determines how soon after initialization each port should begin forwarding or sending packets. For this reason, when the main channel is re-enabled, there is a delay before the connection is reestablished, whereas the failover to the backup adapter is faster. Check the forwarding delay counter on your switch and make it as small as possible so that coming back to the main channel occurs as fast as possible.

For the EtherChannel backup function to work correctly, the forwarding delay counter must not be more than 10 seconds, or coming back to the main EtherChannel might not work correctly. Setting the forwarding delay counter to the lowest value allowed by the switch is recommended.

Adapters not failing over:

If adapter failures are not triggering failovers and you are running AIX 5.2 with 5200-01 or earlier, check to see if your adapter card needs to have link polling enabled to detect link failure.

Some adapters cannot automatically detect their link status. To detect this condition, these adapters must enable a link polling mechanism that starts a timer that periodically verifies the status of the link. Link polling is disabled by default. For EtherChannel to work correctly with these adapters, however, the link polling mechanism must be enabled on each adapter before the EtherChannel is created. If you are running AIX 5L Version 5.2 with the 5200-03 Recommended Maintenance package and later, the link polling is started automatically and this cannot be an issue.

Adapters that have a link polling mechanism have an ODM attribute called **poll_link**, which must be set to `yes` for the link polling to be enabled. Before creating the EtherChannel, use the following command on every adapter to be included in the channel:

```
smitty chgenet
```

Change the **Enable Link Polling** value to `yes` and press Enter.

Jumbo frames:

Aside from enabling the **use_jumbo_frame** attribute on the EtherChannel, you must also enable jumbo frames on each adapter before creating the EtherChannel.

To do this, run the following command:

```
smitty chgenet
```

Jumbo frames are enabled automatically in every underlying adapter when the **use_jumbo_frame** attribute of an EtherChannel is set to yes.

Remote dump:

Remote dump is not supported over an EtherChannel.

Internet Protocol over InfiniBand (IPoIB)

Internet protocol (IP) packets can be sent over an InfiniBand (IB) interface. This transport is accomplished by encapsulating IP packets of IB packets using a network interface.

In order to use IP over IB, you must install and configure the InfiniBand connection manager (ICM) driver and at least one IB device in the system. To see if an IB device is already installed, run the **lsdev -C | grep iba** command. The name of the fileset containing the IB interface is: `devices.common.IBM.ib`. The `devices.chrp.IBM.lhca` fileset is an example of a currently supported adapter fileset.

To configure an ICM driver, refer to “Configuring an InfiniBand Communication Manager driver” on page 372.

In order to create the InfiniBand interface (IB IF), the IB IF must be able to join an existing broadcast-multicast group with a user-provided PKEY (or a default PKEY = 0xFFFF is used if the user did not provide one) and a user provided Q_Key (or a default Q_Key = 0x1E is used if the user did not provide one). A broadcast-multicast group is a multicast group that the interface must join in order to send broadcast and **ARP** packets. If such a broadcast-multicast group does not exist, or can not be created by the interface, the creation of the IB IF will fail.

You can create or change an IB IF using the command-line interface or the SMIT user interface. The parameters required to create an IB IF are the following:

- *interface name*
- *adapter name*
- *port number*
- *interface IP address*

The following parameters are for changing the IB IF:

- *internet address*
- *network mask*
- *MTU size* (equal to the MTU desired, less 4 bytes for the IB header)
- *state*
- *Send and Receive queue size (default is 4000)*
- *Multicast Queue Key*
- *Super packet on or off*

The following is an example of the command used to create an IB IF from the command line:

```
$ /usr/sbin/mkiba -i ib0 -p 1 -A iba0 -a 1.2.3.8 [-P -1 -S "up" -m "255.255.254.0" -M 2044]
```

where:

Item	Description
`-M 2044`	Maximum transmit unit.
`-m "255.255.254.0"`	Net mask.
`-p 1`	Port number (defaults to *1* if not provided).
`-A iba0`	IB device name.
`-a 1.2.3.8`	IF IP address.
`-i ib0`	Interface name.
`-P -1`	Partition key (defaults to *PKEY* if not provided. After the interface is created, PKEY cannot be changed; the user must obtain a non-default PKEY from the network administrator.)
`-S "up"`	Interface state.
`-q 8000`	Receive and Transmit Queues size (each).
`-Q 0x1E`	Multicast Queue Key assigned to the Multicast group (defaults to Q_KEY = 0x1E if not provided).
`-k "on"`	Superpacket will allow the interface TCP/IP MTU to be 64K. It has to be enabled in the remote host also to be able to work.

The following is an example of the command used to create an IB IF from the SMIT user interface:

```
$ smitty inet
```

After the Network Interface Selection menu is displayed, follow this procedure:

1. Select **Add a Network Interface** or **Change / Show Characteristics of a Network Interface.** The Add a Network Interface menu displays.
2. In the Add a Network Interface menu, select **Add an IB Network Interface**. The Add an IB Network Interface menu displays.
3. In the Add an IB Network Interface menu, make the necessary changes and press Enter.

Creating, displaying, adding, deleting ARP entries and modifying ARP timers

An **Address Resolution Protocol (ARP)** entry allows an interface to communicate with another interface even if they are not in the same multicast group.

An **ARP** entry can be created manually using the **arp -t ib** command.

To display all **ARP** entries, run the**$ arp -t ib -a** command. If you want to display a specific number of **ARP** entries, you can specify the number. For example, `$ arp -t ib -a 5` displays 5 **ARP** entries.

The following command adds an **ARP** entry:

```
$ arp -t ib -s IB interface name dlid <16 bits DLID> dqp
16 bits hex Destination Queue Pair Number
ipaddr <Destination IP Address>
```

where:

Item	Description
DLID	is the Destination Local ID.
DGID	is the Destination Global ID.

The following command removes an **ARP** entry:

```
$ arp -t ib -d IP Address
```

The following modify the ARP entry's timer values for complete and incomplete ARP entries. These values are used to remove the ARP entries after a period of time:

```
arp –t ib –i <number in complete minutes to remove incomplete ARP entries>
-c <number in complete minutes to remove complete ARP entries>
```

The current default time for incomplete ARP entries to be removed is 3 minutes. For complete ARP entries, the default time is 24 hours. If the values need to be changed, the execution of the command will change only the values for all the current interfaces configured (or in defined state). If new interfaces are configured, the command must be executed again. The values are also changed in ODM.

The values can be changed dynamically to one specific interface by running the **ifconfig** command:

```
To change the incomplete ARP entry timer
ifconfig ib0 inc_timer 4
ifconfig ib0 com_timer 60
```

Changing the parameters of an InfiniBand interface

To change the parameters of an IB IF, use the SMIT user interface or command-line commands.

To change the parameters of an IB IF using SMIT:

1. Run the **$ smitty inet** command. The Network Interface Selection menu displays.
2. In the Network Interface Selection menu, select **Change / Show Characteristics of a Network Interface**. The Available Network Interfaces menu displays.
3. In the Available Network Interfaces menu, select **InfiniBand Interface**. The Change / Show an IB Interface menu displays.
4. Change the desired parameters.

To change the IB IF parameters at the command line, run the **$ ifconfig** command. The following command changes the IB IF parameters from the command line:

```
$ ifconfig ib0 [ib_port port number mtu maximum transmission unit p_key
16 bits hex partition key  ib_adapter InfiniBand adapter name netmask
dotted decimals]
$ ifconfig ib0 inc_timer  3  com_timer 60
```

- *inc_timer* is the time in minutes that an incomplete ARP entry will expire. Default is 2 mintues.
- *com_timer* is the time in minutes that a complete ARP entry will expire. Default is 24 hours.

Configuring an InfiniBand Communication Manager driver

Use this procedure to configure an InfiniBand Communication Manager.

1. Run the **$ smitty icm** command. The InfiniBand Communication Manager menu displays.
2. In the InfiniBand Communication Manager menu, select **Add an InfiniBand Communication Manager**.
3. In the Add an InfiniBand Communication Manager menu, select **Add an InfiniBand Communication Manager**. The Name of IB Communication Manager to Add menu displays.
4. In the Name of IB Communication Manager to Add menu, select **management icm InfiniBand**.
5. Use the defaults or change any necessary parameters, and then press Enter.

iSCSI software initiator and software target

The iSCSI software initiator enables AIX to access storage devices using TCP/IP on Ethernet network adapters. The iSCSI software target enables AIX to export local storage to be accessed by other iSCSI initiators using the iSCSI protocol that is defined in the RFC 3720.

The use of iSCSI technology, often referred to as SAN over IP technology, allows the deployment of storage area networking over an IP network. iSCSI is an open, standards-based approach by which SCSI information is encapsulated by **TCP/IP** to allow its transport over Ethernet and gigabit Ethernet networks. iSCSI allows an existing Ethernet network to transfer SCSI commands and data with total location independence. iSCSI solutions use the following different, but integrally related, components:

- Initiators

  These are the device drivers that reside on the client. They encapsulate SCSI commands and route them over the IP network to the target device.
- Target software

  The software receives the encapsulated SCSI commands over the IP network. The software can also provide configuration support and storage-management support.
- Target hardware

  The hardware can be a storage appliance that contains embedded storage. The hardware can also be a gateway or bridge product that contains no internal storage of its own.

Configuring iSCSI software initiator

The software initiator is configured using SMIT as shown in this procedure.

1. Select **Devices**.
2. Select **iSCSI**.
3. Select **Configure iSCSI Protocol Device**.
4. Select **Change / Show Characteristics of an iSCSI Protocol Device**
5. Verify that the **Initiator Name** value is correct. The **Initiator Name** value is used by the iSCSI Target during login.

   Note: A default initiator name is assigned when the software is installed. This initiator name can be changed by the user to match local network naming conventions.
6. The **Maximum Targets Allowed** field corresponds to the maximum number of iSCSI targets that can be configured. If you reduce this number, you also reduce the amount of network memory pre-allocated for the iSCSI protocol driver during configuration.
7. Configure the iSCSI discovery method by using the **Discovery Policy** field to discover the iSCSI targets. The iSCSI initiator software supports the following 4 discovery methods:

   file The information about targets is stored in a configuration file.

odm The information about targets is stored in the Object Data Manager (ODM) objects. When using an iSCSI disk as a boot disk or as part of the **rootvg** boot, the **odm** discovery method must be used. See Adding a statically-discovered iSCSI target into ODM.

isns The information about targets is stored on an Internet Storage Name Service (iSNS) server and automatically retrieved during the iSCSI initiator configuration.

slp The information about targets is stored on a Service Location Protocol (SLP) service agent or directory agent and automatically retrieved during the iSCSI initiator configuration.

After the software initiator is configured, do the following:

1. If the discovery policy is **file**, edit the `/etc/iscsi/targets` file to include the iSCSI targets needed during device configuration.

   Each uncommented line in the file represents an iSCSI target. For more information, see targets File in *Files Reference*.

   If the discovery policy is **odm**, use the **mkiscsi** command or smit panels to create the target definitions in ODM. For more information, see Adding a statically-discovered iSCSI target into ODM.

   If the discovery policy is **isns** or **slp**, ensure that the iSNS or SLP server is properly configured and accessible by the iSCSI initiator.

   iSCSI device configuration requires that the iSCSI targets can be reached through a properly configured network interface. Although the iSCSI software initiator can work using a 10/100 Ethernet LAN, it is designed for use with a gigabit Ethernet network that is separate from other network traffic.
2. After defining the targets, type the following command:

   ```
   cfgmgr -l iscsi0
   ```

   This reconfigures the software initiator driver.

   This command causes the driver to attempt to communicate with the targets listed in the `/etc/iscsi/targets` file, and to define a new hdisk for each LUN on the targets that are found. For more information, see the **cfgmgr** command description in *Commands Reference, Volume 1*.

   Note: If the appropriate disks are not defined, review the configuration of the initiator, the target, and any iSCSI gateways to ensure correctness, and then rerun the **cfgmgr** command.

If you want to further configure parameters for iSCSI software initiator devices, use SMIT as follows:

1. Select **Devices**.
2. Select **Fixed Disk**.

A typical software initiator device look similar to the following:

```
hdisk2   Available             Other iSCSI Disk Drive
```

If the iSCSI disk supports command tag queuing and NACA=1 in the control byte, consider changing the queue depth setting of the disk to a larger value. A larger value may help improve device performance. The optimal queue depth setting cannot exceed the actual queue size on the drive. Setting the queue depth to a value larger then the drive's queue size would possibly degrade performance. To determine the drive's queue size consult the drive's documentation.

Configuring the iSCSI software target

The iSCSI software target driver enables AIX to act as one iSCSI target device or as several iSCSI target devices. The iSCSI target driver exports local disks, logical volumes, or local files to iSCSI initiators that connect to AIX using the iSCSI protocol and TCP/IP.

Each target device has an iSCSI Qualified Name and a set of logical unit numbers (LUNs) that are available to initiators that connect to the virtual iSCSI target. For each target device, you can specify which network interface and which TCP/IP port numbers the target driver can use to accept incoming connections.

Note: You need to have the iSCSI target fileset installed. The fileset name is devices.tmiscsw.rte and the fileset is contained on the AIX Expansion pack.

To configure an iSCSI target driver, complete the following steps:

1. Create a single instance of the iSCSI target driver using the following SMIT path. This instance acts as a container for the other iSCSI objects.

   Devices > **iSCSI** > **iSCSI Target Device** > **iSCSI Target Protocol Device** > **Add an iSCSI Target Protocol Device**
2. Create one iSCSI target device for each virtual iSCSI target that is allocated by the iSCSI target driver. Use the following SMIT path to create each iSCSI target device:

   Devices > **iSCSI** > **iSCSI Target Device** > **iSCSI Targets** > **Add an iSCSI Targets**
3. Define one or more LUNs for each target device using the following SMIT path:

   Note: The LUNs are accessible by initiators that connect to a virtual target. On the iSCSI target, each LUN can be associated with either a previously defined logical volume, with a physical volume, or with a file previously created on a local file system. Any physical volume that is associated with an iSCSI target logical unit cannot be used in any other way by the AIX system that is running the iSCSI target driver.

   Devices > **iSCSI** > **iSCSI Target Device** > **iSCSI Target LUNs**

   This step generally completes the configuration. However, if you are using Challenge Handshake Authentication Protocol (CHAP), or if you are using Access Control Lists (ACLs) to indicate which initiators can access which LUNs, an additional step is required to complete the target configuration.

   - If you are using CHAP authentication of the initiators, edit the `/etc/tmiscsi/autosecrets` file and add the secrets used by the initiators to log in. The `/etc/tmiscsi/autosecrets` file contains one entry per target. Each entry has the following format:

     target_name chap_name chap_secret
   - If you are using ACLs to indicate which initiators can access which LUNs, edit the `/etc/tmiscsi/access_lists` file and add one entry per target. Each entry has the following format:

     target_name|*lun_name iSCSI_name*, *iSCSI_name,...*

Related information:

/etc/tmiscsi/autosecrets

/etc/tmiscsi/access_lists

/etc/tmiscsi/isns_servers

iSCSI software initiator considerations

Consider the following when dealing with iSCSI software initiators.

- Target discovery

  The iSCSI software initiator supports the following 4 forms of target discovery:

  file A text file is used to configure each target.

  odm ODM objects are used to configure each target. When using an iSCSI disk as a boot disk or as part of the rootvg boot, the **odm** discovery method must be used.

  isns Each target is registered in one or more Internet Storage Name Service (iSNS) servers.

  slp Each target is registered in one or more Service Location Protocol (SLP) service agents or directory agents.
- iSCSI Authentication

Only CHAP (MD5) can be used to configure Initiator authentication. Target authentication is not implemented.

- Number of configured LUNs

  The maximum number of configured LUNs tested using the iSCSI software initiator is 128 per iSCSI target. The software initiator uses a single TCP connection for each iSCSI target (one connection per iSCSI session). This TCP connection is shared among all LUNs that are configured for a target. The software initiator's TCP socket send and receive space are both set to the system socket buffer maximum. The maximum is set by the **sb_max** network option. The default is 1 MB.

- Volumes Groups

  To avoid configuration problems and error log entries when you create volume groups using iSCSI devices, follow these guidelines:

  – Configure volume groups that are created using iSCSI devices to be in an inactive state after reboot. After the iSCSI devices are configured, manually activate the iSCSI-backed volume groups. Then, mount any associated file systems.

    Volume groups are activated during a different boot phase than the iSCSI software driver. For this reason, it is not possible to activate iSCSI volume groups during the boot process.

  – Do not span volume groups across non-iSCSI devices.

- I/O Failures

  If connectivity to iSCSI target devices is lost, I/O failures occur. To prevent I/O failures and file system corruption, stop all I/O activity and unmount iSCSI backed file systems before doing anything that will cause long term loss of connectivity to active iSCSI targets.

  If a loss of connectivity to iSCSI targets occurs while applications are attempting I/O activities with iSCSI devices, I/O errors will eventually occur. It might not be possible to unmount iSCSI backed file systems because the underlying iSCSI device stays busy.

  File system maintenance must be performed if I/O failures occur due to loss of connectivity to active iSCSI targets. To do file system maintenance, run the **fsck** command.

- Do not use the AIX iSCSI software initiator or the AIX iSCSI software target with the loopback interface (`lo0`). The processing of the loopback interface interrupt differs from the processing of the physical or virtual Ethernet adapter network interface interrupt. The AIX operating system might stop operations if the loopback interface is used with the iSCSI software drivers.

Related information:

Adding a statically-discovered iSCSI target into ODM

iSCSI security considerations:

The `/etc/iscsi` directory, the `/etc/tmiscsi` directory, and the files in those directories are protected from non-privileged users through file permission and ownership.

CHAP secrets are saved in the `/etc/iscsi/targets` file and the `/etc/tmiscsi/autosecrets` file in clear text.

Note: Do not to change the original file permission and ownership of these files.

iSCSI performance considerations:

Set the following configurations to get the best performance from iSCSI.

To ensure the best performance:

- Enable the TCP Large Send, TCP send and receive flow control, and Jumbo Frame features of the AIX Gigabit Ethernet Adapter and the iSCSI Target interface.
- Tune network options and interface parameters for maximum iSCSI I/O throughput on the AIX system as follows:

- Enable the RFC 1323 network option.
- Set up the **tcp_sendspace**, **tcp_recvspace**, **sb_max**, and **mtu_size** network options and network interface options to appropriate values.

  The iSCSI Software Initiator's maximum transfer size is 256 KB. Assuming that the system maximums for **tcp_sendspace** and **tcp_recvspace** are set to 262144 bytes, an **ifconfig** command used to configure a gigabit Ethernet interface might look like the following:

  ```
  ifconfig en2 10.1.2.216 mtu 9000 tcp_sendspace 262144 tcp_recvspace 262144
  ```
- Set the **sb_max** network option to at least 524288, and preferably 1048576.
- Set the **mtu_size** to 9000.
- For some iSCSI targets, the TCP Nagle algoritm must be disabled for best performance. Use the **no** command to set the **tcp_nagle_limit** parameter to `0`, which will disable the Nagle algorithm.

Note: For information on setting network options, see the **no** command description in *Commands Reference, Volume 4*.

For more information and additional tuning parameters, see TCP and UDP performance tuning.

iSCSI software target considerations

Consider the following when you are defining an iSCSI software target and exporting logical unit numbers (LUNs):

- The iSCSI Qualified Name (IQN) of each virtual target is specified in SMIT when a software target is defined. The SMIT panel does not restrict the format of the name. However, some iSCSI initiators require that the IQN is specified in the format defined by the iSCSI protocol. Using an incorrect name format might prevent the initiator from logging to the target and accessing the disks exported by the target.

  To display the current name of an iSCSI target device, complete the following steps:

  1. Run a command similar to the following. For this example, assume the iSCSI target device is `target0`.

     ```
     lsattr -E -l target0
     ```
  2. Check the `iscsi_name` attribute.
- The inquiry data returned for an exported LUN has the following values:
  - `Vendor ID: AIX`
  - `Product ID: iSCSI_VDASD`
  - `ANSI version number: 3`
- Do not use the AIX iSCSI software initiator or the AIX iSCSI software target with the loopback interface (`lo0`). The processing of the loopback interface interrupt differs from the processing of the physical or virtual Ethernet adapter network interface interrupt. The AIX operating system might stop operations if the loopback interface is used with the iSCSI software drivers.

Stream Control Transmission Protocol

Stream Transmission Control Protocol (SCTP) is a connection-oriented protocol, similar to TCP, but provides message-oriented data transfer, similar to **UDP**. The AIX operating system is compliant with RFC 4960.

The following table highlights the general differences in behavior between SCTP and existing transport protocols, TCP and UDP.

Table 82. Differences between TCP, UDP, and SCTP

Attribute	TCP	UDP	SCTP
Reliability	Reliable	Unreliable	Reliable
Connection Management	Connection-oriented	Connectionless	Connection-oriented
Transmission	Byte-oriented	Message-oriented	Message-oriented
Flow Control	Yes	No	Yes
Congestion Control	Yes	No	Yes
Fault Tolerance	No	No	Yes
Data Delivery	Strictly Ordered	Unordered	Partially ordered
Security	Yes	Yes	Improved

In general, **SCTP** may provide more flexibility for certain applications, like **Voice over IP (VoIP)**, that require the reliable but message-oriented data transfer. For this category of applications, **SCTP** is most likely better-suited than **TCP** or **UDP**.

- **TCP** provides reliable and strict order-of-transmission data delivery. For applications that need reliability, but can tolerate unordered or partially ordered data delivery, **TCP** may cause some unnecessary delay because of head-of-line blocking. With the concept of multiple streams within a single connection, **SCTP** can provide strictly ordered delivery within a stream while logically isolating data from different streams.
- **SCTP** is message-oriented, unlike **TCP**, which is byte-oriented. Because of the byte-oriented nature of **TCP**, the application has to add its own record marking to maintain message boundaries.
- **SCTP** provides some degree of fault tolerance by using the Multihoming feature. A host is considered multihomed when it has more than one network interface attached, either on the same or different networks. An **SCTP** association can be established between two multihomed hosts. In this case, all IP addresses of both endpoints are exchanged at association startup; this allows each endpoint to use any of these addresses over the life of the connection if one of the interfaces is down for any reason, as long as the peer is reachable through the alternate interfaces.
- **SCTP** provides additional security features that **TCP** and **UDP** do not. In **SCTP**, resource allocation during association setup is delayed until the client's identity can be verified using a cookie exchange mechanism, thus reducing the possibility of Denial of Service attacks.

SCTP association startup and shutdown

SCTP association startup and shutdown guidelines are described here.

SCTP association is comprised of a four way handshake that takes place in the following order:

1. The client sends an **INIT** signal to the server to initiate an association.
2. On receipt of the **INIT** signal, the server sends an **INIT-ACK** response to the client. This **INIT-ACK** signal contains a state cookie. This state cookie must contain a Message Authentication Code (MAC), along with a time stamp corresponding to the creation of the cookie, the life span of the state cookie, and the information necessary to establish the association. The MAC is computed by the server based on a secret key only known to it.
3. On receipt of this **INIT-ACK** signal, the client sends a **COOKIE-ECHO** response, which just echoes the state cookie.
4. After verifying the authenticity of the state cookie using the secret key, the server then allocates the resources for the association, sends a **COOKIE-ACK** response acknowledging the **COOKIE-ECHO** signal, and moves the association to **ESTABLISHED** state.

SCTP supports also graceful close of an active association upon request from the **SCTP** user. The following sequence of events occurs:

1. The client sends a **SHUTDOWN** signal to the server, which tells the server that the client is ready to close the connection.

2. The server responds by sending a **SHUTDOWN-ACK** acknowledgement.
3. The client then sends a **SHUTDOWN-COMPLETE** signal back to the server.

SCTP also supports abrupt close (**ABORT** signal) of an active association upon the request from the **SCTP** client or due to an error in the **SCTP** stack. However, **SCTP** does not support half open connections. More information about the protocol and its internals can be found in RFC 4960.

In addition to the differences specified above between **SCTP** and existing transport protocols, **SCTP** provides the following features:

- **Sequenced delivery within streams**: A stream in **SCTP** context refers to a sequence of user messages that are transferred between endpoints. An **SCTP** association can support multiple streams. At the time of association setup, the user can specify the number of streams. The effective value of number of stream is fixed after negotiating with the peer. Within each stream the order of data delivery is strictly maintained. However across the streams data delivery is independent. Thus, the loss of data from one stream does not prevent data from being delivered in another stream. This allows a user application to use different streams for logically independent data. Data can also be delivered in an unordered fashion using a special option. This can be useful to send urgent data.
- **User data fragmentation**: **SCTP** can fragment user messages to ensure that the packet size passed to the lower layer does not exceed the path MTU. At the time of receipt the fragments are reassembled into a complete message and passed to the user. Although fragmentation can also be performed at the network level, transport-layer fragmentation provides various advantages over IP-layer fragmentation. Some of these advantages including not having to re-send entire messages when fragments are lost in the network and reducing the burden on routers, which would otherwise possibly have to perform IP fragmentation.
- **Acknowledgment and Congestion Control**: Packet acknowledgment is necessary for reliable data delivery. When **SCTP** does not get a acknowledgment for a packet it sends within a specified time, it triggers a retransmission of the same packet. **SCTP** follows congestion control algorithms similar to those used by **TCP**. In addition to using cumulative acknowledgements like **TCP**, **SCTP** uses Selective Acknowledgment (SACK) mechanism which allows it to acknowledge packets selectively.
- **Chunk bundling**: A chunk may contain user data or **SCTP** control information. Multiple chunks can be bundled together under the same **SCTP** header. Chunk bundling requires assembly of chunks into **SCTP** packet at the sending end and subsequently disassembly of the packet into chunks at the receiver end.
- **Packet validation**: Each **SCTP** packet has a verification tag field that is set during the time of association startup by each endpoint. All packets are sent with the same verification tag through the lifetime of the association. If, during the lifetime of the association, a packet is received with an unexpected verification tag, the packet is discarded. Also the CRC-32 checksum should be set by sender of each **SCTP** packet to provide increased protection for the data corruption in the network. Any packet received with an invalid CRC-32 checksum is discarded.
- **Path management**: At the time of association setup, each endpoint may advertise the list of transport addresses it has. However only one primary path is defined for the **SCTP** association and is used for the normal data transfer. In case the primary path goes down, the other transport addresses are used. During the lifetime of the association, heartbeats are sent at regular interval through all the paths to monitor the status of the path.

SCTP socket APIs

The features of **SCTP** socket APIs include consistency, accessibility, and compatibility.

SCTP Socket APIs were designed to provide the following features:

- Maintain consistency with existing socket APIs
- Provide a base for access to new **SCTP** Features
- Provide compatibility so that most existing **TCP** and **UDP** applications can be migrated to **SCTP** with few changes

In order to facilitate easy migration of existing **TCP** and **UDP** applications, two different styles of **SCTP** APIs have been formulated:

- **UDP**-Style API – Semantics are similar to that defined for connectionless protocols like **UDP**
- **TCP**-Style API – Semantics are similar to that defined for connection-oriented protocols like **TCP**

Though **SCTP** allows for both **TCP** and **UDP** style of socket APIs to be defined and used, in AIX 5.3, only support for **UDP**-style socket syntax is provided because the **UDP**-style API provides more flexibility in accessing new features of **SCTP**. Using the **UDP**-style API a typical server uses the following sequence of calls during the lifetime of an association.

1. **socket**()
2. **bind**()
3. **listen**()
4. **recvmsg**()
5. **sendmsg**()
6. **close**()

A typical client uses the following sequence of socket API calls:

1. **socket**()
2. **sendmsg**()
3. **recvmsg**()
4. **close**()

The associations created using the above call sequence are called explicitly created associations. An association can be created implicitly after creating a socket, by simply calling **sendmsg**(), **recvmsg**() or **sendto**() and **recvto**(). In the case of implicit association, the **bind**() and **listen**() calls are not required. The syntax of all these system calls are similar to those used with **UDP** sockets. For the socket subroutine, the **Type** field should be set to `SOCK_SEQPACKET` and the **Protocol** field should be `IPPROTO_SCTP`. In addition to these standard socket APIs **SCTP** provides two new APIs: **sctp_peeloff**() and **sctp_opt_info**(). More information about the use of Socket API for **SCTP** can be found in the SCTP Socket API Draft. **SCTP** has been implemented as a kernel extension in AIX 5.3. A user can use the **sctpctrl** command to load and unload the SCTP kernel extension.

In addition, this command can also be used to view and change various other statistics and tunables of the **SCTP** kernel extension using different options like get and set. For more information on **sctpctrl** command see the **sctpctrl** command description in *Commands Reference, Volume 5*

sctp_bindx subroutine:

Adds or removes bind address on a socket.

Library

```
/usr/lib/libsctp.a
```

Syntax

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/sctp.h>

int sctp_bindx(int sd, struct sockaddr * addrs, int addrcnt, int flags);
```

Description

The **sctp_bindx** subroutine adds or removes a set of bind addresses passed in the **addrs** array to or from the socket **sd**. The **addrcnt** parameter is the number of addresses in the array, and the **flags** parameter specifies if the addresses need to be added or removed.

If the socket **sd** is an IPv4 socket, the addresses passed must be IPv4 addresses. If the socket **sd** is an IPv6 socket, the addresses passed can be either IPv4 or IPv6 addresses.

The **addrs** parameter is a pointer to an array of one or more socket addresses. Each address is contained in its appropriate structure, that is, **struct sockaddr_in** or **struct sockaddr_in6**. The family of the address type must be used to distinguish the address length. The caller specifies the number of addresses in the array along with **addrcnt**.

The **flags** parameter can be either **SCTP_BINDX_ADD_ADDR** or **SCTP_BINDX_REM_ADDR**. An application can use SCTP_BINDX_ADD_ADDR to associate additional addresses with an endpoint after calling the **bind** command. The **SCTP_BINDX_REM_ADDR** parameter directs SCTP to remove the given addresses from the association. A caller might not remove all addresses from an association. The command fails resulting in the **EINVAL** error code.

Return values

Upon successful completion, the **sctp_bindx()** command returns 0. On failure, the **sctp_bindx()** command returns -1 and sets the **errno** parameter to the appropriate error code.

Error codes

Error	Description
EINVAL	The **EINVAL** error code indicates that the port or address is invalid or the command is trying to remove all addresses from an association.
EOPNOTSUPP	The **EOPNOTSUPP** error code indicates that the command is trying to add or remove addresses from a connected association.

sctp_getladdrs and sctp_freeladdrs subroutines:

Returns all locally bound addresses on a socket.

Library

/usr/lib/libsctp.a

Syntax

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/sctp.h>

int sctp_getladdrs(int sd, sctp_assoc_t assoc_id, struct sockaddr **addrs);
void sctp_freeladdrs(struct sockaddr *addrs);
```

Description

The **sctp_getladdrs** subroutine returns all locally bound addresses on a socket. On return, the **addrs** parameter points to a dynamically allocated packed array of the **sockaddr** structures of the appropriate type for each local address. You must use the **sctp_freeladdrs** parameter to free the memory.

Note: The in or out parameter **addrs** must not be NULL.

If the **sd** parameter is an IPv4 socket, the addresses returned are all IPv4 addresses. If the **sd** parameter is an IPv6 socket, the addresses returned can be a mix of IPv4 or IPv6 addresses.

For one-to-many style sockets, the **id** field specifies the association to query. For one-to-one style sockets, the **id** field is ignored. If the **id** field is set to 0, the locally bound addresses are returned without regard to any particular association.

The **sctp_freeladdrs** subroutine frees all the resources allocated by the **sctp_getladdrs** subroutine.

Return value

On success, the **sctp_getladdrs** subroutine returns the number of local addresses bound to the socket. If the socket is unbound, 0 is returned and the value of the ***addrs** field is undefined. On error, the **sctp_getladdrs** subroutine returns -1, and the value of the ***addrs** field is undefined.

sctp_getpaddrs and sctp_freepaddrs subroutines:

Returns all peer addresses in an association.

Library

/usr/lib/libsctp.a

Syntax

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/sctp.h>

int sctp_getpaddrs(int sd, sctp_assoc_t assoc_id, struct sockaddr **addrs);
void sctp_freepaddrs(struct sockaddr *addrs);
```

Description

The **sctp_getpaddrs** subroutine returns all the peer addresses in an association. On return, the **addrs** parameter points to a dynamically allocated packed array of the **sockaddr** structures of the appropriate type for each address. You must use the **sctp_freepaddrs** subroutine to free the memory.

Note: The in or out parameter **addrs** must not be NULL.

If the **sd** parameter is an IPv4 socket, the addresses returned are all IPv4 addresses. If the **sd** parameter is an IPv6 socket, the addresses returned can be a mix of IPv4 or IPv6 addresses. For one-to-many style sockets, the **id** field specifies the association to query. For one-to-one style sockets, the **id** field is ignored.

The **sctp_freepaddrs** subroutine frees all the resources allocated by the **sctp_getpaddrs** subroutine.

Return value

On success, the **sctp_getpaddrs** subroutine returns the number of peer addresses in the association. If there is no association on this socket, 0 is returned and the value of the ***addrs** field is undefined. On error, the **sctp_getpaddrs** subroutine returns -1, and the value of the ***addrs** field is undefined.

Path MTU discovery

For two hosts communicating across a path of multiple networks, a transmitted packet becomes fragmented if its size is greater than the smallest MTU of any network in the path. Because packet fragmentation can result in reduced network performance, it is desirable to avoid fragmentation by transmitting packets with a size is no greater than the smallest MTU in the network path. This size is called the path MTU.

The operating system supports a path MTU discovery algorithm as described in RFC 1191. Path MTU discovery can be enabled for **TCP** and **UDP** applications by modifying the **tcp_pmtu_discover** and

udp_pmtu_discover options of the **no** command. When enabled for **TCP**, path MTU discovery will automatically force the size of all packets transmitted by **TCP** applications to not exceed the path MTU. Because **UDP** applications themselves determine the size of their transmitted packets, **UDP** applications must be specifically written to utilize path MTU information by using the **IP_FINDPMTU** socket option, even if the **udp_pmtu_discover no** option is enabled. By default, the **tcp_pmtu_discover** and **udp_pmtu_discover** are enabled.

When the Path MTU discovery is attempted for a destination, a pmtu entry gets created in a Path MTU (PMTU) table. This table can be displayed using the **pmtu** display command. Accumulation of pmtu entries can be avoided by allowing unused pmtu entries to expire and be deleted. PMTU entry expiration is controlled by the pmtu_expire option of the **no** command. pmtu_expire is set to 10 minutes by default.

Since routes can change dynamically, the path MTU value for a path might also change over time. Decreases in the path MTU value will result in packet fragmentation, so discovered path MTU values are periodically checked for decreases. By default, decreases are checked for every 10 minutes, and this value can be changed by modifying the value of the **pmtu_default_age** option of the **no** command.

UDP applications will always need to set the **IP_DONTFRAG** socket option to detect decreases in PMTU. This will enable immediate detection of decreases in Path MTU rather than checking for decreases every pmtu_default_age minutes.

Increases in the path MTU value can result in a potential increase in network performance, so discovered path MTU values are periodically checked for increases. By default, increases are checked for every 30 minutes, and this value can be changed by modifying the value of the **pmtu_rediscover_interval** option of the **no** command.

If not all of the routers in the network path support RFC 1191, then it might not be possible to determine an exact path MTU value. In these cases, the **mmtu** command can be used to add or delete path MTU values that are attempted.

Note:

1. Path MTU discovery cannot be used on duplicate routes, including those configured for group routing (see "Route use restrictions" on page 338). Path MTU discovery can be used on duplicate routes
2. Enabling path MTU discovery sets the value of the **arpqsize** option of the **no** command to a minimum value of 5. This value is not decreased if path MTU discovery is subsequently disabled.

TCP/IP Quality of Service

Quality of Service (QoS) is a family of evolving Internet standards that provides ways to give preferential treatment to certain types of IP traffic.

With the proper support for QoS along a route, this can ameliorate the effects of variable queueing delays and congestion that contribute to poor network performance. The operating system provides host support for QoS to classify outbound traffic into distinct classes of service and to announce and establish resource reservations as requested by client applications.

QoS can be used by an institution to deploy and enforce network policies governing the use of network bandwidth. With QoS, a host can:

- Regulate the amount of traffic of a certain type injected into the network;
- Mark selected packets according to some policy so that subsequent routers can deliver the indicated service;
- Support services such as the virtual leased line service with proper QoS support along the route; and
- Participate in the resource reservation requests from receivers and announce sender sessions available for resource reservation requests.

The QoS support provides the following functions:

- Differentiated services as defined in RFC 2474
- Traffic policing
- In-profile and out-of-profile packet marking
- Traffic shaping
- Metering
- Integrated services for client and server applications as defined in RFC 1633
- RSVP signaling (RFC 2205)
- Guaranteed service (RFC 2212)
- Controlled-Load service (RFC 2211)
- Policy-based networking
- RAPI shared library for application

The QoS subsystem consists of four components:

QoS kernel extension (/usr/lib/drivers/qos)
: The QoS kernel extension resides in `/usr/lib/drivers/qos` and is loaded and unloaded using the **cfgqos** and **ucfgqos** configuration methods. This kernel extension enables QoS support.

Policy agent (/usr/sbin/policyd)
: The policy agent is a user-level daemon that resides in `/usr/sbin/policyd`. It provides support for policy management and interfaces with the QoS kernel extension to install, modify, and delete policy rules. Policy rules can be defined in the local configuration file (`/etc/policyd.conf`), retrieved from a central network policy server using LDAP, or both.

RSVP agent (/usr/sbin/rsvpd)
: The RSVP agent is a user-level daemon that resides in `/usr/sbin/rsvpd`. It implements the RSVP signaling protocol semantics.

RAPI shared library (/usr/lib/librapi.a)
: Applications can use the RSVP API (RAPI) to request enhanced quality of service as defined by the Integrated Services Internet QoS model. This library interacts with the local RSVP agent to propagate the QoS request along the path of the data flow using the RSVP protocol. This API is an open standard.

Note: This implementation of QoS is based on a set of evolving Internet standards and draft standards currently under development by the Internet Engineering Task Force (IETF) and its various working groups. This technology will become more consistent and well defined as these standardization efforts progress within the IETF. It is also important to note that QoS is an emerging Internet technology that is just beginning to be deployed within the Internet. There are many benefits of QoS at all stages of deployment. However, true end-to-end services can only be realized when QoS support exists all along a particular route.

QoS models

The QoS models for the Internet are open standards defined by the IETF.

There are two Internet QoS models currently being standardized within the IETF: *integrated services* and *differentiated services*. These two Internet QoS models augment the traditional best-effort service model described in RFC 1812.

Integrated services:

Integrated Services (IS) is a dynamic resource reservation model for the Internet described in RFC 1633.

Hosts use a signaling protocol called Resource ReSerVation Protocol (RSVP) to dynamically request a specific quality of service from the network. QoS parameters are carried in these RSVP messages and each network node along the path installs the parameters to obtain the requested quality of service. These QoS parameters describe one of two currently defined services, guaranteed service and controlled-load service. An important characteristic of IS is that this signaling is done for each traffic flow and reservations are installed at each hop along the route. Although this model is well-suited for meeting the dynamically changing needs of applications, there exist some significant scaling issues that imply it cannot be deployed in a network in which single routers handle many simultaneous flows.

Differentiated services:

Differentiated Services (DS) removes the per-flow and per-hop scalability issues, replacing them with a simplified mechanism of classifying packets.

Rather than a dynamic signaling approach, DS uses bits in the IP type of service (TOS) byte to separate packets into classes. The particular bit pattern in the IP TOS byte is called the DS codepoint and is used by routers to define the quality of service delivered at that particular hop, in much the same way routers do IP forwarding using routing table lookups. The treatment given to a packet with a particular DS codepoint is called a per-hop behavior (PHB) and is administered independently at each network node. When the effects of these individual, independent PHBs are concatenated, this results in an end-to-end service.

Differentiated services is being standardized by an IETF working group, which has defined three PHBs: the Expedited Forwarding (EF) PHB, the Assured Forwarding (AF) PHB group, and the Default (DE) PHB. The EF PHB can be used to implement a low latency, low jitter, low loss, end-to-end service such as a virtual leased line (VLL). AF is a family of PHBs, called a PHB group, that is used to classify packets into various drop precedence levels. The drop precedence assigned to a packet determines the relative importance of a packet within the AF class. It can be used to implement the so-called *Olympic* service, which consists of three classes: bronze, silver, and gold. The DE PHB is the traditional best-effort service model as standardized in RFC 1812.

Supported standards and draft standards

These RFCs and Internet drafts describe the standards on which this QoS implementation is based.

Item	Description
RFC 2474	Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers
RFC 2475	An Architecture for Differentiated Services
RFC 1633	Integrated Services in the Internet Architecture: an Overview
RFC 2205	Resource ReSerVation Protocol (RSVP)
RFC 2210	The Use of RSVP with IETF Integrated Services
RFC 2211	Specification of the Controlled-Load Network Element Service
RFC 2212	Specification of Guaranteed Quality of Service
RFC 2215	General Characterization Parameters for Integrated Service Network Elements

Item	Description
draft-ietf-diffserv-framework-01.txt, October 1998	A Framework for Differentiated Services
draft-ietf-diffserv-rsvp-01.txt, November 1998	A Framework for Use of RSVP with DIFF-serv Networks
draft-ietf-diffserv-phb-ef-01.txt	An Expedited Forwarding PHB
draft-ietf-diffserv-af-04.txt	Assured Forwarding PHB Group
draft-rajan-policy-qosschema-00.txt, October 1998	Schema for Differentiated Services and Integrated Services in Networks
draft-ietf-rap-framework-01.txt, November 1998	A Framework for Policy-based Admission Control[25]
draft-ietf-rap-rsvp-ext-01.txt, November 1998	RSVP Extensions for Policy Control

Note: QoS is an emerging Internet technology. There are many benefits of QoS at all stages of deployment. However, true end-to-end services can only be realized when QoS support exists all along a particular route.

QoS installation

QoS is packaged with `bos.net.tcp.server`. This fileset must be installed in order to use QoS.

To use the `RAPI` shared library, `bos.adt.include` must also be installed.

Stopping and starting the QoS subsystem

QoS can be started or stopped through SMIT with the `smit qos` fast path or with the **mkqos** and **rmqos** commands.

1. To disable the QoS subsystem now and on the next system restart:
   ```
   /usr/sbin/rmqos -B
   ```
2. To enable the QoS subsystem now only:
   ```
   /usr/sbin/mkqos -N
   ```

See the command descriptions for **mkqos** and **rmqos** for the startup and removal command flags.

The **policyd** and **rsvpd** daemons are configured through the `/etc/policyd.conf` and `/etc/rsvpd.conf` configuration files, respectively. These configuration files *must* be edited to customize the QoS subsystem to the local environment. QoS does not work correctly with the supplied example configurations.

RSVP agent configuration

The RSVP agent is required if the host is to support the RSVP protocol.

The `/etc/rsvpd.conf` configuration file is used to configure the RSVP agent. The syntax of the configuration file is described in the sample configuration file installed in `/etc/rsvpd.conf`.

The following example illustrates a possible RSVP configuration in which the host has 4 interfaces (virtual or physical) given by the 4 IP addresses, `1.2.3.1`, `1.2.3.2`, `1.2.3.3`, and `1.2.3.4`.

```
interface 1.2.3.1
interface 1.2.3.2 disabled
interface 1.2.3.3 disabled
interface 1.2.3.4
{
  trafficControl
}

rsvp 1.2.3.1
{
  maxFlows 64
}

rsvp 1.2.3.4
{
  maxFlows 100
}
```

Interface `1.2.3.1` has been enabled for RSVP. However, traffic control has not been specified and incoming RSVP RESV messages do not cause resource reservation within the TCP subsystem. This interface can support a maximum of 64 simultaneous RSVP sessions.

Interfaces `1.2.3.2` and `1.2.3.3` have been disabled. The RSVP agent cannot use this interface to transmit or receive RSVP messages.

Interface `1.2.3.4` has been enabled for RSVP. In addition, it can install resource reservations into the **TCP** subsystem in response to an RSVP RESV message. This interface can support up to 100 RSVP sessions.

Any other interfaces present on the host but not mentioned explicitly in /etc/rsvpd.conf are disabled.

Policy agent configuration

The policy agent is a required component of the QoS subsystem.

The /etc/policyd.conf configuration file is used to configure the policy agent. The syntax of this configuration file is described in the sample configuration file installed in /etc/policyd.conf.

The policy agent can be configured by editing /etc/policyd.conf. Additionally, the following commands are provided to assist in configuring policies:

- **qosadd**
- **qosmod**
- **qoslist**
- **qosremove**

In the following example, a premium service category is created and used in the tcptraffic policy rule. This service category has a maximum rate of 110000 Kbps, a token bucket depth of 10000 bits, and an outgoing IP TOS value of 11100000 in binary. The tcptraffic policy rule gives this premium service to all traffic with source IP address given by 1.2.3.6, destination address 1.2.3.3, and destination port in the range 0 to 1024.

```
ServiceCategories  premium
{
   PolicyScope     DataTraffic
   MaxRate         110000
   MaxTokenBucket  10000
   OutgoingTOS     11100000
}

ServicePolicyRules   tcptraffic
{
   PolicyScope   DataTraffic
   ProtocolNumber 6  # tcp
   SourceAddressRange      1.2.3.6-1.2.3.6
   DestinationAddressRange 1.2.3.3-1.2.3.3
   DestinationPortRange    0-1024
   ServiceReference        premium
}
```

The following statements set up a default service category and use it to restrict the UDP traffic flowing from interfaces 1.2.3.1 through 1.2.3.4 to IP addresses 1.2.3.6 through 1.2.3.10, port 8000.

```
ServiceCategories  default
{
   MaxRate         110000
   MaxTokenBucket  10000
   OutgoingTOS     00000000
}

ServicePolicyRules   udptraffic
{
   ProtocolNumber 17  # udp
   SourceAddressRange      1.2.3.1-1.2.3.4
   DestinationAddressRange 1.2.3.6-1.2.3.10
   DestinationPortRange    8000-8000
   ServiceReference        default
}
```

The following example configuration can be used to download rules from an LDAP server using the distinguished subtree name, to lookup the policies on the LDAP server host.

```
    ReadFromDirectory
    {
      LDAP_Server      1.2.3.27
      Base             ou=NetworkPolicies,o=myhost.mydomain.com,c=us
    }
```

QoS troubleshooting

The **qosstat** command may be used to display status information about the installed and active policies in the QoS subsystem. This information may be useful to you in determining where a problem exists if you are troubleshooting your QoS configuration.

qosstat can be used to generate the following report.

```
Action:
  Token bucket rate (B/sec): 10240
  Token bucket depth (B): 1024
  Peak rate (B/sec): 10240
  Min policied unit (B): 20
  Max packet size (B): 1452
  Type: IS-CL
  Flags: 0x00001001 (POLICE,SHAPE)

  Statistics:
    Compliant packets: 1423 (440538 bytes)

  Conditions:
    Source address        Dest address          Protocol
    192.168.127.39:8000   192.168.256.29:35049  tcp       (1 connection)

Action:
  Token bucket rate (B/sec): 10240
  Token bucket depth (B): 1024
  Peak rate (B/sec): 10240
  Outgoing TOS (compliant): 0xc0
  Outgoing TOS (non-compliant): 0x00
  Flags: 0x00001011 (POLICE,MARK)
  Type: DS

  Statistics:
    Compliant packets: 335172 (20721355 bytes)
    Non-compliant packets: 5629 (187719 bytes)

  Conditions:
    Source address        Dest address          Protocol
    192.168.127.39:80     *:*                   tcp       (1 connection)
    192.168.127.40:80     *:*                   tcp       (5 connections)
```

QoS policy specification

The object classes and attributes used by the policy agent to specify policies for quality of service (QoS) on outgoing traffic are described here.

The object classes and attributes are defined, followed by guidelines to enable marking, policing, and shaping.

These conventions are used in the explanations that follow.

```
     p  : choose one in the allowed parameter set
     B  : integer value of a byte (i.e., 0 =< B =< 255)
     b  : bit string starting with left most bit (e.g., 101 is
          equivalent 10100000 in a byte field)
     i  : integer value
     s  : a character string
     a  : IP address format B.B.B.B
    (R) : Required parameter
    (O) : Optional parameter
```

ReadFromDirectory statement:

This statement specifies parameters for establishing an LDAP session.

The ReadFromDirectory statement is used in the /etc/policyd.conf file to establish the LDAP session.

```
ReadFromDirectory
{
  LDAP_Server    a   # IP address of directory server running LDAP
  LDAP_Port     i    # Port number LDAP server is listening to
  Base     s         # Distinguished Name for LDAP usage
  LDAP_SelectedTag s # Tag to match SelectorTag in object classes
}
```

where

```
LDAP_Server (R): IP address of LDAP server
LDAP_Port   (0): Unique port number, default port is 389
Base        (R): Example is o=ibm, c=us where o is your organization and c is country
LDAP_SelectedTag (R): Unique string matching SelectorTag attribute in the object class
```

ServiceCategories statement:

This statement specifies the type of service that a flow of IP packets (for example, from a **TCP** connection or **UDP** data) should receive end-to-end as they traverse the network.

ServiceCategories can be repeated with each having a different name so that they can be referred to later. A ServiceCategories object requires ServicePolicyRules to complete the policy definition.

```
ServiceCategories   s
{
   SelectorTag    s     # Required tag for LDAP Search
   MaxRate        i     # Target rate for traffic in this service class
   MaxTokenBucket i     # The bucket depth
   OutgoingTOS    b     # TOS value of outbound traffic for this service class
   FlowServiceType   p  # Type of traffic

}
```

where

```
s           (R)     : is the name of this service category
SelectorTag (R)     : Required only for LDAP to Search object classes
MaxRate     (O)     : in Kbps (K bits per second), default is 0
MaxTokenBucket(O)   : in Kb, default is system defined maximum
OutgoingTOS (O)     : default is 0
FlowServiceType  (O): ControlledLoad | Guaranteed, default is ControlledLoad
```

ServicePolicyRules statement:

This statement specifies characteristics of IP packets that are used to match to a corresponding service category.

In other words, it defines a set of IP datagrams that should receive a particular service. ServicePolicyRules are associated with ServiceCategories through the ServiceReference attribute. If two rules refer to the same ServiceCategory, each rule is associated with a unique instance of the ServiceCategory.

```
ServicePolicyRules   s
{
   SelectorTag       s   # Required tag for LDAP Search
   ProtocolNumber    i   # Transport protocol id for the policy rule
   SourceAddressRange      a1-a2
   DestinationAddressRange a1-a2
   SourcePortRange         i1-i2
```

```
   DestinationPortRange    i1-i2
   PolicyRulePriority  i       # Highest value is enforced first
   ServiceReference        s  # Service category name which for this policy rule
}
```

where

```
s           (R): is the name of this policy rule
SelectorTag (R): required only for LDAP to Search object class
ProtocolNumber     (R): default is 0 which causes no match, must explicity specify
SourceAddressRange (O): from a1 to a2 where a2 >= a1, default is 0, any source address
SourcePortRange    (O): from i1 to i2 where i2 >= i1, default is 0, any source port
DestinationAddressRange (O): same as SourceAddressRange
DestinationPortRange    (O): same as SourcePortRange
PolicyRulePriority (O): Important to specify when ovelapping policies exist
ServiceReference   (R): service category this rule uses
```

Guidelines for DiffServ environments

The following are guidelines to specify policies for marking, shaping, and/or policing in a DiffServ environment.

1. **Marking Only**

```
OutgoingTOS     : Desired Type Of Service
FlowServiceType : ControlledLoad
MaxRate         : Take default of 0
```

2. **Shaping Only**

```
OutgoingTOS     : Take default of 0
FlowServiceType : Guaranteed
MaxRate         : Target rate desired for traffic as a positive integer
```

3. **Marking and Policing (See Note)**

```
OutgoingTOS     : Desired Type of Service
FlowServiceType : ControlledLoad
MaxRate         : Target rate desired for traffic as a positive integer
```

4. **Marking and Shaping**

```
OutgoingTOS     : Desired Type of Service
FlowServiceType : Guaranteed
MaxRate         : Target rate desired for traffic as a positive integer
```

Note: The type of service set for the out of profile packets is set to zero in the case of policing.

Sample policyd configuration file

This is a complete example of the `/etc/policyd.conf` configuration file.

```
#loglevel  511    # Verbose logging

######################################################################
#
# Mark rsh traffic on TCP source ports 513 and 514.
ServiceCategories       tcp_513_514_svc
{
        MaxRate                 0               # Mark only
        OutgoingTOS             00011100        # binary
        FlowServiceType         ControlledLoad
}

ServicePolicyRules      tcp_513_514_flt
{
        ProtocolNumber          6  # TCP
        SourceAddressRange      0.0.0.0-0.0.0.0 # Any IP src addr
        DestinationAddressRange 0.0.0.0-0.0.0.0 # Any IP dst addr
        SourcePortRange         513-514
        DestinationPortRange    0-0             # Any dst port
        ServiceReference        tcp_513_514_svc
}
```

```
#
######################################################################
#
# Shape connected UDP traffic on source port 9000.
ServiceCategories       udp_9000_svc
{
        MaxRate                 8192     # kilobits
        MaxTokenBucket          64       # kilobits
        FlowServiceType         Guaranteed
}

ServicePolicyRules      udp_9000_flt
{
        ProtocolNumber          17  # UDP
        SourceAddressRange      0.0.0.0-0.0.0.0 # Any IP src addr
        DestinationAddressRange 0.0.0.0-0.0.0.0 # Any IP dst addr
        SourcePortRange         9000-9000
        DestinationPortRange    0-0             # Any dst port
        ServiceReference        udp_9000_svc
}
#
######################################################################
#
# Mark and police finger traffic on TCP source port 79.
ServiceCategories       tcp_79_svc
{
        MaxRate                 8         # kilobits
        MaxTokenBucket          32        # kilobits
        OutgoingTOS             00011100  # binary
        FlowServiceType         ControlledLoad
}

ServicePolicyRules      tcp_79_flt
{
        ProtocolNumber          6  # TCP
        SourceAddressRange      0.0.0.0-0.0.0.0 # Any IP src addr
        DestinationAddressRange 0.0.0.0-0.0.0.0 # Any IP dst addr
        SourcePortRange         79-79
        DestinationPortRange    0-0             # Any dst port
        ServiceReference        tcp_79_svc
}
#
######################################################################
#
# Mark and shape ftp-data traffic on TCP source port 20.
ServiceCategories       tcp_20_svc
{
        MaxRate                 81920        # kilobits
        MaxTokenBucket          128          # kilobits
        OutgoingTOS             00011101     # binary
        FlowServiceType         Guaranteed
}

ServicePolicyRules      tcp_20_flt
{
        ProtocolNumber          6  # TCP
        SourceAddressRange      0.0.0.0-0.0.0.0 # Any IP src addr
        DestinationAddressRange 0.0.0.0-0.0.0.0 # Any IP dst addr
        SourcePortRange         20-20
        DestinationPortRange    0-0             # Any dst port
        ServiceReference        tcp_20_svc
}
#
######################################################################
#
# LDAP server entry.
#ReadFromDirectory
```

```
#{
#   LDAP_Server             9.3.33.138  # IP address of LDAP server
#   Base                    o=ibm,c=us  # Base distinguished name
#   LDAP_SelectedTag        myhost      # Typically client hostname
#}
#
#################################################################
```

IBM SecureWay Directory Server policy loads

If the policy daemon is used with the IBM SecureWay Directory LDAP Server, use this schema as a guide to update /etc/ldapschema/V3.modifiedschema before starting the LDAP server.

Refer to "Planning and configuring for LDAP name resolution (IBM SecureWay Directory schema)" on page 197 for details.

```
objectClasses {
( ServiceCategories-OID NAME 'ServiceCategories' SUP top MUST
( objectClass $ SelectorTag $ serviceName ) MAY
( description $ FlowServiceType $ MaxRate $ MaxTokenBucket $ OutgoingTos ) )
( ServicePolicyRules-OID NAME 'ServicePolicyRules' SUP top MUST
( objectClass $ PolicyName $ SelectorTag ) MAY
( description $ DestinationAddressRange $ DestinationPortRange $
ProtocolNumber $ ServiceReference $ SourceAddressRange $ SourcePortRange ) )
}
attributeTypes {
( DestinationAddressRange-OID NAME 'DestinationAddressRange' SYNTAX
1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE )
( DestinationPortRange-OID NAME 'DestinationPortRange' SYNTAX
 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE )
( FlowServiceType-OID NAME 'FlowServiceType'
 SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE )
( MaxRate-OID NAME 'MaxRate' SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE )
( MaxTokenBucket-OID NAME 'MaxTokenBucket' SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE )
( OutgoingTos-OID NAME 'OutgoingTos' SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE )
( PolicyName-OID NAME 'PolicyName' SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE )
( ProtocolNumber-OID NAME 'ProtocolNumber' SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE )
( SelectorTag-OID NAME 'SelectorTag' SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE )
( ServiceReference-OID NAME 'ServiceReference' SYNTAX
 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE )
( SourceAddressRange-OID NAME 'SourceAddressRange' SYNTAX
 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE )
( SourcePortRange-OID NAME 'SourcePortRange' SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 SINGLE-VALUE )
}

IBMattributeTypes {
( DestinationAddressRange-OID DBNAME ( 'DestinationAddressRange' 'DestinationAddressRange' ) )
( DestinationPortRange-OID DBNAME ( 'DestinationPortRange' 'DestinationPortRange' ) )
( FlowServiceType-OID DBNAME ( 'FlowServiceType' 'FlowServiceType' ) )
( MaxRate-OID DBNAME ( 'MaxRate' 'MaxRate' ) )
( MaxTokenBucket-OID DBNAME ( 'MaxTokenBucket' 'MaxTokenBucket' ) )
( OutgoingTos-OID DBNAME ( 'OutgoingTos' 'OutgoingTos' ) )
( PolicyName-OID DBNAME ( 'PolicyName' 'PolicyName' ) )
( ProtocolNumber-OID DBNAME ( 'ProtocolNumber' 'ProtocolNumber' ) )
( SelectorTag-OID DBNAME ( 'SelectorTag' 'SelectorTag' ) )
( ServiceReference-OID DBNAME ( 'ServiceReference' 'ServiceReference' ) )
( SourceAddressRange-OID DBNAME ( 'SourceAddressRange' 'SourceAddressRange' ) )
( SourcePortRange-OID DBNAME ( 'SourcePortRange' 'SourcePortRange' ) )
}

ldapSyntaxes {
}

matchingRules {
}
```

QoS system configuration

Policies that overlap are installed in the QoS Manager in a nondeterministic ordering. In the case of overlapping policies the `PolicyRulePriority` attribute of the `ServicePolicyRules` should be specified to determine the ordering of enforcement of policies. The `PolicyRulePriority` attribute takes an integer as a parameter and, in the case of overlapping policies, the rule with the highest integer value is enforced.

Only connected **UDP** sockets are supported for QoS.

The policy and RSVP agents as mutually independent. Thus, care must be taken not to specify a policy that conflicts with, or is covered by, an existing RSVP reservation. In the presence of such conflicts, the system accepts the first policy or reservation while flagging a violation for the others.

For correct operation, the `MaxTokenBucket` attribute must be set to at least the maximum MTU of all interfaces configured in the system.

Policy modifications are handled by the policy agent by automatically deleting the existing policies and installing the new ones. This may result in a short, temporary window of time during which the corresponding traffic receives default (typically best effort) service.

IETF standards compliance for IntServ and DiffServ models

This release is compatible with evolving Internet Engineering Task Force (IETF) standards for Differentiated (DiffServ) and Integrated Services (IntServ) on the Internet.

The following RFCs describe various components of the IntServ model:

- The Use of RSVP with IETF Integrated Services (RFC 2210)
- Specification of the Controlled-Load Network Element Service (RFC 2211)
- Specification of Guaranteed Quality of Service (RFC 2212)

The following RFCs describe various components of the DiffServ model:

- Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers (RFC 2474)
- An Architecture for Differentiated Services (RFC 2475)

The following RFC outlines the current usage of the IP TOS octet:

- Type of Service in the Internet Protocol Suite (RFC 1349)

The following RFCs outline future practices governing usage of the IP TOS octet:

- Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers (RFC 2474)
- Assured Forwarding PHB Group (RFC 2597)
- An Expedited Forwarding PHB (RFC 2598)

IPv6 Support

QoS only supports IPv4. IPv6 is not supported.

Policy daemon control

You can control the policy daemon by using the system resource controller (SRC).

For example, the command:

```
startsrc -s policyd -a "-i 60"
```

starts the policy agent with a refresh interval of 60 seconds.

The command

```
stopsrc -s policyd
```

stops the policy daemon.

Note: Stopping the policy daemon does not remove the installed policies in the kernel. When you start the policy daemon again, the old policies (previously installed in the kernel) are deleted, and the policies defined in the `/etc/policyd.conf` file are reinstalled.

The **refresh** SRC command is not currently supported.

QoS commands and methods

The **TCP/IP** quality of service commands and methods are listed here.

For important updates to this documentation, consult the README file in `/usr/samples/tcpip/qos`.

The following QoS commands are supported:

- **qosadd**
- **qoslist**
- **qosmod**
- **qosremove**
- **qosstat**
- **mkqos**
- **rmqos**

The following QoS methods are supported:

- **cfgqos**
- **ucfgqos**

TCP/IP troubleshooting

The **netstat** command is a good tool for diagnosing common problems in a **Transmission Control Protocol/Internet Protocol (TCP/IP)** network environment.

The **netstat** command lets you determine which area of the network has a problem. After you have isolated the problem to an area, you can use more sophisticated tools to proceed. For example, you might use the **netstat -i** and **netstat -v** to determine if you have a problem with a particular hardware interface, and then run diagnostics to further isolate the problem. Or, if the **netstat -s** command shows that there are protocol errors, you could then use the **trpt** or **iptrace** commands.

Communication problems

Common **TCP/IP** communication problems include the inability to communicate with a host on your network and routing problems. These are some solutions.

If you cannot communicate with a host on your network:

- Try to contact the host, using the **ping** command. Run the **ping** command on the local host to verify that the local interface to the network is up and running.
- Try to resolve the name of the host, using the **host** command. If the name does not resolve, you have a name resolution problem. See “Name resolution problems” on page 394 for more information.

If the name resolves and you are trying to contact a host on another network, you may have a routing problem. See “TCP/IP routing problems” on page 395 for more information.

- If your network is a token-ring network, check to see if the target host is on another ring. If so, the **allcast** field is probably set incorrectly. Use the System Management Interface Tool (SMIT) fast path `smit chinet` to access the Network Interfaces menu. Then, set the Confine Broadcast to Local Ring field to **no** in the token-ring dialog.

- If there are a large number of **Address Resolution Protocol (ARP)** packets on your network, verify that your subnet mask is set correctly. This condition is known as a broadcast storm and can affect your system performance.

Name resolution problems

Resolver routines on hosts running **TCP/IP** attempt to resolve names, using these sources in the order listed.

1. DOMAIN name server (**named**)
2. Network Information Service (NIS)
3. Local `/etc/hosts` file

Resolving client host problems:

If you cannot get a host name resolved, and you are using flat name resolution (using the `/etc/hosts` file), verify that the host name and correct Internet Protocol (IP) address information is in the `/etc/hosts` file.

If you cannot get a host name resolved, and you are using a name server, follow these steps:

1. Verify that you have a `resolv.conf` file specifying the domain name and Internet address of a name server.
2. Verify that the local name server is up by issuing the **ping** command with the IP address of the name server (found in the local `resolv.conf` file).
3. If the local name server is up, verify that the **named** daemon on your local name server is active by issuing the **lssrc -s named** command on the name server.
4. If you are running the **syslogd**, check for logged messages. The output for these messages is defined in the `/etc/syslog.conf` file.

If these steps do not identify the problem, check the name server host.

Resolving name server host problems:

Use this procedure to resolve host name server problems.

If you cannot get a host name resolved:

1. Verify that the **named** daemon is active by issuing the following command:

   ```
   lssrc -s named
   ```

2. Verify that the address of the target host exists and is correct in the name server database. Send a **SIGINT** signal to the **named** daemon to dump the database and cache to the file `/var/tmp/named_dump.db`. Verify that the address you are trying to resolve is there and is correct.

   Add or correct name-to-address resolution information in the **named** hosts data file for the master name server of the domain. Then issue the following **SRC** command to reread the data files:

   ```
   refresh -s named
   ```

3. Verify that the name resolution requests are being processed. To do this, enter the **named** daemon from the command line and specify a debugging level. Valid debug levels are 1 through 9. The higher the level, the more information the debug mechanism logs.

   ```
   startsrc -s named -a "-d DebugLevel"
   ```

4. Check for configuration problems in the **named** data files. For more information, see “Name server resolution” on page 178. In addition, see the "DOMAIN Data File Format,", "DOMAIN Reverse Data File Format," "DOMAIN Cache File Format," and the "DOMAIN Local Data File Format" in the *Files Reference*.

   Note: A common error is the incorrect use of the `.` (period) and the `@` (at sign) in the DOMAIN data files.

If external users cannot reach your domains, make sure that all your non-master name servers (slave, hint) have equal time-to-live (TTL) information in the DOMAIN data files.

If external resolvers query your servers constantly, make sure your servers are distributing DOMAIN data files with reasonable TTL values. If the TTL is zero or another small value, the data you transfer times out very quickly. Set the minimum value in your start of authority (SOA) records to a week or more to solve this problem.

TCP/IP routing problems

If you cannot reach a destination host, consider the solutions to the following situations.

- If you receive a `Network Unreachable` error message, make sure that a route to the gateway host has been defined and is correct. Check this by using the **netstat -r** command to list kernel routing tables.
- If you receive a `No route to host` error message, verify that the local network interface is up by issuing the **ifconfig** interface_name command. The output indicates whether or not the interface is up. Use the **ping** command to try and reach another host on your network.
- If you receive a `Connection timed out` error message:
  - Verify that the local gateway is up using the **ping** command with the name or Internet address of the gateway.
  - Make sure that a route to the gateway host has been defined and is correct. Check this by using the **netstat -r** command to list kernel routing tables.
  - Make sure the host you want to communicate with has a routing table entry back to your machine.
- If you are using static routing, make sure that a route to the target host and gateway host has been defined. Check this by using the **netstat -r** command to list kernel routing tables.

  Note: Make sure the host you want to communicate with has a routing table entry to your machine.
- If you are using dynamic routing, verify that the gateway is listed and correct in the kernel routing tables by issuing the **netstat -r** command.
- If the gateway host is using the **Routing Information Protocol** (RIP) with the **routed** daemon, make sure that a static route to the target host is set up in the `/etc/gateways` file.

  Note: You need to do this only if the routing daemon cannot identify the route to a distant host through queries to other gateways.
- If the gateway host is using the **RIP** with the **gated** daemon, make sure that a static route to the target host is set up in the `gated.conf` file.
- If you are using dynamic routing with the **routed** daemon:
  - If **routed** cannot identify the route through queries (for example, if the target host is not running the **RIP**, check the `/etc/gateways` file to verify that a route to the target host is defined.
  - Make sure that gateways responsible for forwarding packets to the host are up and running the **RIP**. Otherwise, you'll need to define a static route.
  - Run the **routed** daemon using the debug option to log such information as bad packets received. Invoke the daemon from the command line using the following command:

    ```
    startsrc -s routed -a "-d"
    ```
  - Run the **routed** daemon using the **-t** flag, which causes all packets sent or received to be written to standard output. When **routed** is run in this mode, it remains under the control of the terminal that started it. Therefore, an interrupt from the controlling terminal kills the daemon.
- If you are using dynamic routing with the **gated** daemon:
  - Verify that the `/etc/gated.conf` file is configured correctly and that you are running the correct protocols.
  - Make sure the gateway on the source network is using the same protocol as the gateway on the destination network.

- Make sure that the machine with which you are trying to communicate has a route back to your host machine.
- Verify that the gateway names in the `gated.conf` file correspond to the gateway names listed in the `/etc/networks` file.

- If you are using the **RIP** or **HELLO** protocols, and routes to the destination cannot be identified through routing queries, check the `gated.conf` file to verify that a route to the target host is defined. Set static routes under the following conditions:
  - The destination host is not running the same protocol as the source host so cannot exchange routing information.
  - The host must be reached by a distant gateway (a gateway that is on a different autonomous system than the source host). The **RIP** can be used only among hosts on the same autonomous system.

If all else fails, you might want to turn on tracing for your routing daemon (either **routed** or **gated**). Use the SRC **traceson** command from the command line, or send a signal to the daemon to specify different levels of tracing. See the **gated** daemon or the **routed** daemon for specifics on sending signals to these daemons.

Resolving problems with SRC support

Use these suggestions to resolve common problems with the System Resource Controller.

- If changes to the `/etc/inetd.conf` file do not take effect:

  Update the **inetd** daemon by issuing the **refresh -s inetd** command or the **kill -1** *InetdPID* command.
- If the **startsrc -s** [*subsystem name*] returns the following error message:

  ```
  0513-00  The System Resource Controller is not active.
  ```

  The System Resource Controller subsystem has not been activated. Issue the **srcmstr &** command to start SRC, then reissue the **startsrc** command.

  You might also want to try starting the daemon from the command line without SRC support.
- If the **refresh -s** [*subsystem name*] or **lssrc -ls** [*subsystem name*] returns the following error message:

  ```
  [subsystem name] does not support this option.
  ```

  The subsystem does not support the SRC option issued. Check the subsystem documentation to verify options the subsystem supports.
- If the following message is displayed:

  ```
  SRC was not found, continuing without SRC support.
  ```

  A daemon was invoked directly from the command line instead of using the **startsrc** command. This is not a problem. However, SRC commands, such as **stopsrc** and **refresh,** will not manipulate a subsystem that is invoked directly.

If the **inetd** daemon is up and running correctly and the appropriate service seems to be correct but you still cannot connect, try running the **inetd** daemon processes through a debugger.

1. Stop the **inetd** daemon temporarily:

   ```
   stopsrc -s inetd
   ```

   The **stopsrc** command stops subsystems like the **inetd** daemon.
2. Edit the `syslog.conf` file to add a debugging line at the bottom. For example:

   ```
   vi /etc/syslog.conf
   ```

   a. Add the line `*.debug /tmp/myfile` at the bottom of the file and exit.

   b. The file that you specify must exist (`/tmp/myfile` in this example). You can use the **touch** command to make your file exists.
3. Refresh the file:
   - If you are using SRC, enter:

```
refresh -s syslogd
```

- If you are not using SRC, kill the **syslogd** daemon:

```
kill -1 `ps -e | grep /etc/syslogd | cut -c1-7`
```

4. Start the **inetd** daemon backup with debugging enabled:

```
startsrc -s inetd -a "-d"
```

   The **-d** flag enables debugging.

5. Try to make a connection to log errors in the `/tmp/myfile` debugging file. For example:

```
tn bastet
Trying...
connected to bastet
login:>
Connection closed
```

6. See if anything shows up as a problem in the debugging file. For example:

```
tail -f /tmp/myfile
```

Resolving telnet or rlogin problems

These explanations can be useful in solving problems with the **telnet** or **rlogin** command.

If you are having trouble with screen distortion in full-screen applications:

1. Check the **TERM** environment variable by issuing one of the following commands:

```
env
echo $TERM
```

2. Verify that the **TERM** variable is set to a value that matches the type of terminal display you are using.

telnet subcommands that can help in debugging problems include:

Item	Description
display	Displays set and toggle values.
toggle	Toggles the display of all network data in hex.
toggle options	Toggles the display of internal **telnet** process options.

If the **inetd** daemon could execute the **telnet** service but you still cannot connect using the **telnet** command, there may be something wrong with the **telnet** interface.

1. Verify that **telnet** is using the correct terminal type.
   a. Check the **$TERM** variable on your machine:

```
echo $TERM
```

   b. Log in to the machine to which you are trying to attach and check the **$TERM** variable:

```
echo $TERM
```

2. Use the **telnet** interface's debugging capabilities by entering the **telnet** command without flags.

```
telnet
tn>
```

   a. Type `open` *`host`* where *host* is the name of the machine.
   b. Press Ctrl-T to get to the `tn%gt;` prompt.
   c. At the `tn>` prompt, type `debug` for debugging mode.

3. Try to connect to another machine using the **telnet** interface:

```
telnet bastet
Trying...
Connected to bastet
Escape character is '^T'.
```

Watch the display as the various commands scroll up the screen. For example:

```
SENT do ECHO
SENT do SUPPRESS GO AHEAD
SENT will TERMINAL TYPE (reply)
SENT do SUPPORT SAK
SENT will SUPPORT SAK (reply)
RCVD do TERMINAL TYPE (don't reply)
RCVD will ECHO (don't reply)
RCVD will SUPPRESS GO AHEAD (don't reply)
RCVD wont SUPPORT SAK (reply)
SENT dont SUPPORT SAK (reply)
RCVD do SUPPORT SAK (don't reply)
SENT suboption TELOPT_NAWS Width 80, Height 25
RCVD suboption TELOPT_TTYPE SEND
RCVD suboption TELOPT_TTYPE aixterm
...
```

4. Check `/etc/termcap` or `/usr/lib/terminfo` for the `aixterm` definition. For example:

   ```
   ls -a /usr/lib/terminfo
   ```

5. If the `aixterm` definition is missing, add it by building the `ibm.ti` file. For example:

   ```
   tic ibm.ti
   ```

   The **tic** command is a terminal information compiler.

Problems with function and arrow keys can arise when using the **rlogin** and **telnet** commands with programs using extended curses. Function and arrow keys generate escape sequences, which are split if too little time is allotted for the entire key sequence. Curses waits a specific amount of time to decide whether an Esc indicates the escape key only or the start of a multibyte escape sequence generated by other keys, such as cursor keys, the action key, and function keys.

If no data, or data that is not valid, follows the Esc in the allotted amount of time, curses decides that the Esc is the escape key, and the key sequence is split. The delay resulting from the **rlogin** or **telnet** command is network dependent. Sometimes arrow and function keys work and sometimes they do not, depending on the speed of the network to which you are connecting. Setting the **ESCDELAY** environment variable to a large value (1000 to 1500) effectively solves this problem.

TCP/IP configuration problems

Network interfaces are automatically configured during the first system startup after the adapter card is installed. However, you still need to set some initial values for **TCP/IP** including the host name, the Internet address, and the subnet mask.

To do this, you can use the SMIT interface in the following ways:

- Use the `smit mktcpip` fast path to set the initial values for the host name, the Internet address, and the subnet mask.
- Use the `smit mktcpip` fast path to specify a name server to provide name resolution service. (Note that `smit mktcpip` configures one network interface only.)
- Use the `smit chinet` fast path to set other network attributes.

You may also want to set up any static routes the host needs for sending transmitting information, such as a route to the local gateway. Use the SMIT fast path, `smit mkroute`, to set these up permanently in the configuration database.

If you are having other problems with your configuration, see the “Configuring a TCP/IP network” on page 103 for more information.

Common TCP/IP problems with network interfaces

Network interfaces are configured automatically during the first system startup after the adapter card is installed. However, there are certain values that must be set in order for **TCP/IP** to start. These include the host name and Internet address and can be set using the SMIT fast path, `smit mktcpip`.

If you choose the SMIT method, use the `smit mktcpip` fast path to set these values permanently in the configuration database. Use the `smit chinet` and `smit hostname` fast paths to change them in a running system. The `smit mktcpip` fast path minimally configures **TCP/IP**. To add adapters, use the Further Configuration menu, which can be reached with the `smit tcpip` fast path.

If you have already checked these to verify accuracy and you are still having trouble sending and receiving information, check the following:

- Verify that your network adapter has a network interface by executing the **netstat -i** command. The output should list an interface, such as **tr0**, in the `Name` column. If it does not, create a network interface by entering the SMIT fast path `smit mkinet`.
- Verify that IP address for the interface is correct by executing the **netstat -i** command. The output should list the IP address in the Network column. If it is incorrect, set the IP address by entering the SMIT fast path `smit chinet`.
- Use the **arp** command to make sure you have the complete IP address for the target machine. For example:

  ```
  arp -a
  ```

  The **arp** command looks for the physical adapter address. This command might show an incomplete address. For example:

  ```
  ? (192.100.61.210) at (incomplete)
  ```

  This could be due to an unplugged machine, a stray address with no machine at that particular address, or a hardware problem (such as a machine that connects and receives packets but is not able to send packets back).
- Look for errors on the adapter card. For example:

  ```
  netstat -v
  ```

  The **netstat -v** command shows statistics for the Ethernet, Token Ring, X.25, and 802.3 adapter device drivers. The command also shows network and error logging data for all device drivers active on an interface including: `No Mbufs Errors`, `No Mbuf Extension Errors`, and `Packets Transmitted and Adapter Errors Detected`.
- Check the error log by running the **errpt** command to ensure that there are no adapter problems.
- Verify that the adapter card is good by running diagnostics. Use the `smit diag` fast path, or the **diag** command.

TCP/IP problems with a SLIP network interface:

In general, the most effective method for debugging problems with a **Serial Line Interface Protocol (SLIP)** interface is to retrace your configuration, verifying each step.

However, you can also:

- Verify that the **slattach** process is running and using the correct tty port by issuing the **ps -ef** command. If it is not, run the **slattach** command. (See "Configuring SLIP over a modem" on page 584 or "Configuring SLIP over a null modem cable" on page 586 for the exact syntax you should use.)
- Verify that the point-to-point addresses are specified correctly by entering the `smit chinet` fast path.

  Select the **SLIP** interface. Make sure that the **INTERNET ADDRESS** and **DESTINATION ADDRESS** fields are correct.

If the modem is not functioning correctly:

- Make sure that the modem was installed correctly. See the modem installation manual.

- Verify that any flow control the modem does is turned off.

If the tty is not functioning correctly, verify that the tty baud rate and modem characteristics are set correctly in the configuration database by entering the `smit tty` fast path.

TCP/IP problems with an Ethernet network interface:

Refer to this checklist if **TCP/IP** problems persist with an Ethernet network interface.

If the network interface has been initialized, the addresses correctly specified, and you have verified that the adapter card is good:

- Verify that you are using a T-connector plugged directly into the inboard/outboard transceiver.
- Make sure you are using an Ethernet cable. (Ethernet cable is 50 OHM.)
- Make sure you are using Ethernet terminators. (Ethernet terminators are 50 OHM.)
- Ethernet adapters can be used with either the transceiver that is on the card or with an external transceiver. There is a jumper on the adapter to specify which you are using. Verify that your jumper is set correctly (see your adapter manual for instructions).
- Verify that you are using the correct Ethernet connector type (thin is BNC; thick is DIX). If you change this connector type, use the **smit chgenet** SMIT fast path to set the Apply Change to Database Only field. (Set to Yes in SMIT.) Restart the machine to apply the configuration change. (See "Adapter management and configuration" on page 156.)

TCP/IP problems with a Token-Ring network interface:

Use these guidelines to resolve communication problems with your network interface.

If you cannot communicate with some of the machines on your network although the network interface has been initialized, the addresses correctly specified, and you have verified that the adapter card is good:

- Check to see if the hosts with whom you cannot communicate are on a different ring. If they are, use the SMIT fast path `smit chinet` to check the Confine BROADCAST to Local Token-Ring field. *Do not* set to No in SMIT.
- Check to see whether the token-ring adapter is configured to run at the correct ring speed. If it is configured incorrectly, use the SMIT to change the adapter ring speed attribute (see "Adapter management and configuration" on page 156). When **TCP/IP** is restarted, the token-ring adapter has the same ring speed as the rest of the network.

TCP/IP problems with a Token-Ring/Ethernet bridge:

If you cannot communicate between a token-ring and an Ethernet network, using a bridge, and you have verified that the bridge is functioning correctly, the Ethernet adapter might be dropping packets.

A machine drops packets if the incoming packet (including headers) is greater than the network adapter maximum transmission unit (MTU) value. For instance, a 1500-byte packet sent by a token-ring adapter over the bridge collects an 8-byte logical link control (LLC) header, making the total packet size 1508. If the receiving Ethernet adapter MTU is set to 1500, the packet is dropped.

Check the MTU values of both network adapters. To allow for the eight-byte LLL header, the token-ring adapter attaches to outgoing packets, set the MTU value for the token-ring adapter at least eight bytes lower than the MTU value for the Ethernet adapter. For example, set the MTU for a token-ring adapter to 1492 to communicate with an Ethernet adapter with an MTU of 1500.

TCP/IP problems with a Token-Ring/Token-Ring bridge:

When operating through a bridge, change the default value of 1500 for the maximum transmission unit (MTU) to a value that is eight less than the maximum information field (maximum I-frame) advertised by the bridge in the routing control field.

To find the routing control field value, use the **iptrace** daemon to look at incoming packets. Bits 1, 2, and 3 of Byte 1 are the Largest Frame Bits, which specify the maximum information field that can be transmitted between two communicating stations on a specific route. See the following for the format of the routing control field:

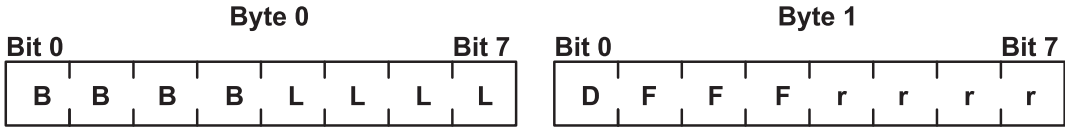


Figure 25. Routing control field

This illustration shows byte 0 and byte 1 of a routing control field. The eight bits of byte one are B, B, B, B, L, L, L, L. The eight bits of byte 1 are D, F, F, F, r, r, r, r.

Values for the Largest Frame Bits are as follows:

Item	Description
000	Specifies a maximum of 516 bytes in the information field.
001	Specifies a maximum of 1500 bytes in the information field.
010	Specifies a maximum of 2052 bytes in the information field.
011	Specifies a maximum of 4472 bytes in the information field.
100	Specifies a maximum of 8144 bytes in the information field.
101	Reserved.
110	Reserved.
111	Used in all-routes broadcast frames.

For example, if the maximum I-frame value is 2052 in the routing control field, the MTU size should be set to 2044. This is for token-ring network interfaces only.

Note: When using **iptrace**, the output file must *not* be on a Network File System (NFS).

TCP/IP problems communicating with a remote host

If you cannot communicate with a remote host, try the these suggestions.

- Run the **ping** command on the local host to verify that the local interface to the network is up and running.
- Use the **ping** command for hosts and gateways that are progressively more hops from the local host to determine the point at which communication fails.

If you are having trouble with packet loss or are experiencing delays in packet delivery, try the following:

- Use the **trpt** command to trace packets at the socket level.
- Use the **iptrace** command to trace all protocol layers.

If you cannot communicate between a token-ring and an Ethernet network using a bridge, and you have verified that the bridge is good:

- Check the MTU values of both adapters. The MTU values must be compatible to allow communication. A machine drops packets if the incoming packet (including headers) is greater than the adapter's MTU

values. For instance, a 1500-byte packet sent over the bridge collects an 8-byte LLC header, making the total packet size 1508. If the receiving machine MTU is set to 1500, a packet of 1508 bytes is dropped.

TCP/IP problems with snmpd response to queries

If **snmpd** is not responding to queries and there are no log messages received, the packet might be to large for the kernel **User Datagram Protocol (UDP)** packet handler.

If this is the case, increase the kernel variables, **udp_sendspace** and **udp_recvspace** by issuing the following commands:

```
no -o udp_sendspace=64000
no -o udp_recvspace=64000
```

The maximum size for a **UDP** packet is 64K. If your query is larger than 64K, it will be rejected. Split the packet into smaller packets to avoid this problem.

TCP/IP problems with Dynamic Host Configuration Protocol

In the event that you cannot obtain configuration data, try the following solutions.

If you cannot get an IP address or other configuration parameters:

- Check to see that you have specified an interface to be configured. This can be done by using the SMIT fast path `smit dhcp`.
- Check to see that there is a server on the local network or a relay agent configured to get your requests off the local network.
- Check to see that the **dhcpcd** program is running. If it is not, use the **startsrc -s dhcpcd** command.

TCP/IP commands

TCP/IP is part of the underlying structure of your system. It allows you to communicate with another terminal or system merely by executing a command or program.

TCP/IP is part of the underlying structure of your system. It allows you to communicate with another terminal or system merely by executing a command or program. Your system takes care of the rest.

Item	Description
chnamsv	Changes **Transmission Control Protocol/Internet Protocol (TCP/IP)** based name service configuration on a host.
chprtsv	Changes a print service configuration on a client or server machine.
hostent	Directly manipulates address-mapping entries in the system configuration database.
ifconfig	Configures or displays network interface parameters for a network, using **TCP/IP**.
mknamsv	Configures **TCP/IP**-based name service on a host for a client.
mkprtsv	Configures **TCP/IP**-based print service on a host.
mktcpip	Sets the required values for starting **TCP/IP** on a host.
no	Configures network options.
rmnamsv	Unconfigures **TCP/IP**-based name service on a host.
rmprtsv	Unconfigures a print service on a client or server machine.
slattach	Attaches serial lines as network interfaces.
arp	Displays or changes the Internet address to hardware address translation tables used by the **Address Resolution Protocol (ARP)**.
gettable	Gets Network Information Center (NIC) format host tables from a host.
hostid	Sets or displays the identifier of the current local host.
hostname	Sets or displays the name of the current host system.
htable	Converts host files to the format used by network library routines.
ipreport	Generates a packet trace report from the specified packet trace file.
iptrace	Provides interface-level packet tracing for Internet protocols.
lsnamsv	Shows name service information stored in the database.
lsprtsv	Shows print service information stored in the database.
mkhosts	Generates the host table file.

Item	Description
namerslv	Directly manipulates domain name server entries for local resolver routines in the system configuration database.
netstat	Shows network status.
route	Manually manipulates the routing tables.
ruser	Directly manipulates entries in three separate system databases that control foreign host access to programs.
ruptime	Displays the status of each host on a network.
securetcpip	Enables the network security feature.
setclock	Sets the time and date for a host on a network.
timedc	Returns information about the **timed** daemon.
trpt	Performs protocol tracing on **Transmission Control Protocol (TCP)** sockets.

SRC commands

SRC commands can affect one daemon, a group of daemons, or a daemon and those daemons it controls (subsystem with subservers).

In addition, some **TCP/IP** daemons do not respond to all SRC commands. The following is a list of SRC commands that can be used to control **TCP/IP** daemons and their exceptions.

Item	Description
startsrc	Starts all **TCP/IP** subsystems and **inetd** subservers. The **startsrc** command works for all **TCP/IP**P subsystems and **inetd** subservers.
stopsrc	Stops all **TCP/IP** subsystems and **inetd** subservers. This command is also called the **stop normal** command. The **stop normal** command allows subsystems to process all outstanding work and terminate gracefully. For **inetd** subservers, all pending connections are allowed to start and all existing connections are allowed to complete. The **stop normal** command works for all **TCP/IP** subsystems and **inetd** subservers.
stopsrc -f	Stops all **TCP/IP** subsystems and **inetd** subservers. This command is also called the **stop force**. The **stop force** command immediately terminates all subsystems. For **inetd** subservers, all pending connections and existing connections are terminated immediately.
refresh	Refreshes the following subsystems and subservers: the **inetd**, **syslogd**, **named**, **dhcpsd**, and **gated** subsystems.
lssrc	Provides short status for subsystems, which is the state of the specified subsystem (active or inoperative). Also provides short status for **inetd** subservers. The short status for **inetd** subservers includes: subserver name, state, subserver description, command name, and the arguments with which it was invoked.
lssrc -l	Provides the short status plus additional information (long status) for the following subsystems: **gated** State of debug or trace, routing protocols activated, routing tables, signals accepted and their function. **inetd** State of debug, list of active subservers and their short status; signals accepted and their function. **named** State of debug, `named.conf` file information. **dhcpsd** State of debug, all controlled IP addresses and their current state. **routed** State of debug and trace, state of supplying routing information, routing tables. **syslogd** **syslogd** configuration information. The **lssrc -l** command also provides long status for **inetd** subservers. The long status includes short status information and active connection information. Some subservers will provide additional information. The additional information by subserver includes: **ftpd** State of debug and logging **telnetd** Type of terminal emulating **rlogind** State of debug **fingerd** State of debug and logging The **rwhod** and **timed** subservers do not provide long status.

Item	Description
traceson	Turns on socket-level debugging. Use the **trpt** command to format the output. The **timed** and **iptraced** subsystems do not support the **traceson** command.
tracesoff	Turns off socket-level debugging. Use the **trpt** command to format the output. The **timed** and **iptraced** subsystems do not support the **tracesoff** command.

For examples of how to use these commands, see the topics about the individual commands. For more information on the System Resource Controller, see System Resource Controller in *Operating system and device management*.

File transfer commands

Brief descriptions of the file transfer commands are listed here.

Item	Description
ftp *hostname*	Transfers files between a local and a remote host.
rcp*file host:file*	Transfers files between local and remote host or between two remote hosts.
tftp	Transfers files between hosts.

Remote login commands

Brief descriptions of the **TCP/IP** remote login commands are listed here.

Item	Description
rexec *host command*	Executes commands one at a time on a remote host.
rlogin *remotehost*	Connects a local host with a remote host.
rsh and **remsh** *remotehost command*	Executes specified command at remote host or logs into the remote host.
telnet, **tn** and **tn3270** *hostname*	Connects the local host with a remote host, using the **TELNET** interface.

Status commands

Brief descriptions of the **TCP/IP** status commands are listed here.

Item	Description
finger or **f** *user@host*	Shows user information.
host *hostname*	Resolves a host name into an Internet address or an Internet address into a host name.
ping *hostname*	Sends an echo request to a network host.
rwho	Shows which users are logged in to hosts on the local network.
whois *name*	Identifies a user by user ID or alias.

Remote communication command

The **TCP/IP** remote communication command, **talk** *User@Host*, lets you converse with another user.

Item	Description
talk *User@Host*	Converse with another user.

Print commands

Brief descriptions of the **TCP/IP** print commands are listed here.

Item	Description
enq *file*	Enqueues a file.
refresh	Requests a refresh of a subsystem or group of subsystems.
smit	Performs system management.

TCP/IP daemons

A *subsystem* is a daemon, or server, that is controlled by the SRC. A *subserver* is a daemon that is controlled by a subsystem. (Daemon commands and daemon names are usually denoted by a **d** at the end of the name.)

The categories of subsystem and subserver are mutually exclusive. That is, daemons are not listed as both a subsystem and as a subserver. The only **TCP/IP** subsystem that controls other daemons is the **inetd** daemon. All **TCP/IP** subservers are also **inetd** subservers.

The following are **TCP/IP** daemons controlled by the SRC:

Subsystems

Item	Description
gated	Provides gateway routing functions and supports the **Routing Information Protocol (RIP)**, the **Routing Information Protocol Next Generation (RIPng)**, **Exterior Gateway Protocol (EGP)**, the **Border Gateway Protocol (BGP)** and **BGP4+**, the **Defense Communications Network Local-Network Protocol (HELLO)**, **Open Shortest Path First (OSPF)**, Intermediate System to Intermediate System (IS-IS), and **Internet Control Message Protocol (ICMP** and **ICMPv6)/Router Discovery routing** protocols. In addition, the **gated** daemon supports the **Simple Network Management Protocol (SNMP)**. The **gated** daemon is one of two routing daemons available for routing to network addresses and is the preferred routing daemon. The **gated** daemon is preferred over the **routed** daemon because the **gated** daemon supports more gateway protocols.
inetd	Invokes and schedules other daemons when requests for the daemon services are received. This daemon can also start other daemons. The **inetd** daemon is also known as the super daemon.
iptrace	Provides interface-level packet-tracing function for Internet protocols.
named	Provides the naming function for the **Domain Name Server** protocol (DOMAIN).
routed	Manages the network routing tables and supports the **Routing Information Protocol (RIP)**. The **gated** daemon is preferred over the **routed** daemon because the **gated** daemon supports more gateway protocols.
rwhod	Sends broadcasts to all other hosts every three minutes and stores information about logged-in users and network status. Use the **rwhod** daemon with extreme care, because it can use significant amounts of machine resources.
timed	Provides the time server function.

Note: Both the **routed** and **gated** daemons are listed as TCP/IP subsystems. Do not run the **startsrc -g tcpip** command, which initiates both of these routing daemons, along with all the other **TCP/IP** subsystems. Running both daemons simultaneously on one machine can produce unpredictable results.

TCP/IP daemons controlled by the **inetd** subsystem are the following:

inetd Subservers

Item	Description
comsat	Notifies users of incoming mail.
fingerd	Provides a status report on all logged-in users and network status at the specified remote host. This daemon uses the **Finger** protocol.
ftpd	Provides the file transfer function for a client process using the **File Transfer Protocol (FTP)**.
rexecd	Provides the foreign host server function for the **rexec** command.
rlogind	Provides the remote login facility function for the **rlogin** command.
rshd	Provides the remote command execution server function for the **rcp** and **rsh** commands.
talkd	Provides the conversation function for the **talk** command.

Item	Description
syslogd	Reads and logs system messages. This daemon is in the **Remote Access Service** (RAS) group of subsystems.
telnetd	Provides the server function for the **TELNET** protocol.
tftpd	Provides the server function for the **Trivial File Transfer Protocol (TFTP)**.
uucpd	Handles communications between the Basic Network Utilities (BNU) and **TCP/IP**.

Device methods

Device methods are programs associated with a device that perform basic device configuration operations.

See List of TCP/IP Programming References in *Communications Programming Concepts* for information about **TCP/IP** methods.

Request for comments

The following **TCP/IP** Request for Comments (RFCs) are supported in the AIX system.

For a list of the RFCs (Request for Comments) supported by this operating system, see the List of TCP/IP Programming References in *Communications Programming Concepts*.

- RFC 1359 *Connecting to the Internet: What connecting institutions should anticipate*
- RFC 1325 *FYI on questions and answers: Answers to commonly asked 'new Internet user' questions*
- RFC 1244 *Site Security Handbook*
- RFC 1178 *Choosing a Name for Your Computer*
- RFC 1173 *Responsibilities of host and network managers: A summary of the `oral tradition' of the Internet*

Basic Networking Utilities

The BNUs are a group of programs, directories, and files that establish communications between computer systems on local and remote networks. It can be used to communicate with any UNIX system on which a version of the UNIX-to-UNIX Copy Program (UUCP) is running. BNU is one of the extended services programs that can be installed with the base operating system.

A group of commands related to UUCP, a UNIX-to-UNIX communication program developed by AT&T and modified as part of the Berkeley Software Distribution (BSD) are contained in BNU. BNU provides commands, processes, and a supporting database for connections to local and remote systems. Communication networks such as Token-Ring and Ethernet are used to connect systems on local networks. A local network can be connected to a remote system by hardwire or telephone modem. Commands and files can then be exchanged between the local network and the remote system.

Before users on your system can run BNU programs, BNU must be installed and configured.

BNU is controlled by a set of configuration files that determine whether remote systems can log in to the local system and what they can do after they log in. These configuration files must be set up according to the requirements and resources of your system.

To maintain BNU, you must read and remove log files periodically and check the BNU queues to ensure jobs are correctly transferring to remote systems. You must also periodically update the configuration files to reflect changes in your system or remote systems.

How BNU works

BNU uses a set of hardware connections and software programs to communicate between systems.

A structure of directories and files tracks BNU activities. This structure includes a set of public directories, a group of administrative directories and files, configuration files, and lock files. Most of the directories for BNU are created during the installation process. Some of the administrative directories and files are created by various BNU programs.

With the exception of the remote login commands, BNU works as a batch system. When a user requests a job sent to a remote system, BNU stores the information needed to complete the job. This is known as *queuing* the job. At scheduled times, or when a user instructs it to do so, BNU contacts various remote systems, transfers queued work, and accepts jobs. These transfers are controlled by the configuration files on your system and those of the remote system.

National Language Support for BNU commands

All BNU commands, except **uucpadm**, are available for for National Language Support.

User names need not be in ASCII characters. However, all system names must be in ASCII characters. If a user attempts to schedule a transfer or a remote command execution involving non-ASCII system names, BNU returns an error message.

BNU file and directory structure

BNU uses a structure of directories and files to keep track of activities.

This structure includes the public directories, configuration files, administrative directories, and lock files.

Most of the directories for BNU are created during the installation process. Some of the administrative directories and files are created by various BNU programs as they run.

BNU public directories

When specified, the BNU public directory (`/var/spool/uucppublic`) stores files that have been transferred to the local system from other systems.

The files wait in the public directory until users claim them. The public directory is created when BNU is installed. Within the public directory, BNU creates a subdirectory for each remote system that sends files to the local system.

BNU configuration files

The BNU configuration files, also known as the BNU supporting database, reside in the `/etc/uucp` directory. The files must be configured specifically for your system.

They are owned by the `uucp` login ID and can be edited only with root authority. The configuration files contain information about:

- Accessible remote systems
- Devices for contacting the remote systems
- Times to contact the remote systems
- What the remote systems are allowed to do on your system.

Some configuration files also specify limits on BNU activities that prevent your system from becoming overloaded.

The BNU configuration files include:

Item	Description
Devices	Contains information about available devices, including both modems and direct connections.
Dialcodes	Contains dialing code abbreviations, which allow you to shorten phone numbers in the Systems file.
Dialers	Specifies calling command syntax for a specific modem type ("dialer").
Maxuuscheds	Limits simultaneous scheduled jobs.
Maxuuxqts	Limits simultaneous remote command executions.
Permissions	Contains access permission codes. This file is the primary file for determining the security for BNU.
Poll	Specifies when the BNU program should poll remote systems to initiate tasks.
Sysfiles	Lists files that serve as the Systems, Devices, and Dialers files for the BNU configuration. If this file is not used, the default files are /etc/uucp/Systems, /etc/uucp/Devices, and /etc/uucp/Dialers.
Systems	Lists accessible remote systems and information needed to contact them, including the device to use and the user name and password combinations you need to log in. Also specifies the times when the systems can be contacted.

The configuration files cross-reference each other when BNU is in use. For example:

- The Devices file contains a *Token* field that refers to entries in the Dialers file.
- The Systems file contains an entry for a *Class* of device. A device of each *Class* referred to in the Systems file must be defined in the Devices file.
- The Poll file contains entries for systems your system calls. Each of these systems must be defined in the Systems file.

Entries in the BNU configuration files depend on the types of connections between your system and each remote system. For example, special entries must be made if Transmission Control Protocol/Internet Protocol (TCP/IP) or direct connections are used to contact other systems. If modems are used to contact other systems, the modems must be defined in the Dialers file.

The Systems, Devices, and Permissions files must be configured on your system before you can contact remote systems using BNU. Other configuration files enable you to use BNU capabilities, such as automatic polling. Many of the configuration files must be modified periodically to reflect changes to your system or the systems you contact. The Sysfiles file can be used to specify files other than the default Systems, Devices, and Dialers files to fulfill the same role.

BNU administrative directories and files

The BNU administrative directories and files are in subdirectories of the /var/spool/uucp directory.

These directories and files contain two types of information:

- Data waiting to be transferred to other systems
- Log and error information about BNU activities.

Under the /var/spool/uucp directory, BNU creates the following directories:

Item	Description
.Admin	Contains four administrative files: • audit • Foreign • errors • xferstats These files contain error and log information about BNU activities.
.Corrupt	Contains copies of files that cannot be processed by the BNU program.
.Log and .Old	Contain log files from BNU transactions.
.Status	Stores the last time the **uucico** daemon tried to contact remote systems.
.Workspace	Holds temporary files that the file transport programs use internally.
.Xqtdir	Contains execute files with lists of commands that remote systems can run.

Item	Description
SystemName	Contains files used by file transport programs. These files are: • Command (**C.***) • Data (**D.***) • Execute (**X.***) • Temporary (**TM.***) BNU creates a *SystemName* directory for each remote system it contacts.

The directories whose names begin with a dot are *hidden*. They cannot be found with an **ls** or **li** command unless the **-a** flag is used. When the **uucico** daemon is started, it searches the `/var/spool/uucp` directory for work files and transfers the files from any directory that is not hidden. The **uucico** daemon sees only the *SystemName* directories, not the other administrative directories.

The files in the hidden directories are owned by the uucp login ID. These files can be accessed only with root authority or with a login ID with a UID of 5.

For further information about maintaining the BNU administrative directories, see "BNU maintenance" on page 421.

BNU lock files

The BNU lock files are stored in the `/var/locks` directory. When BNU uses a device to connect to a remote computer, it places a lock file for that device in the `/var/locks` directory.

When another BNU program or any other program needs the device, that program checks the `/var/locks` directory for a lock file. If a lock file exists, the program waits until the device is available or uses another device for the communication.

In addition, the **uucico** daemon places lock files for remote systems in the `/var/locks` directory. Before contacting a remote system, the **uucico** daemon checks the `/var/locks` directory for a lock file for that system. These files prevent other instances of the **uucico** daemon from establishing duplicate connections to the same remote system.

Note: Other software besides BNU, such as Asynchronous Terminal Emulation (ATE) and TCP/IP, uses the `/var/locks` directory.

Configuring BNU

This procedure explains how to configure Basic Network Utilities (BNU) for various types of connections, such as direct, modem, and Transmission Control Protocol/Internet Protocol (TCP/IP) connections.

Prerequisites

- BNU must be installed on your system.
- You must have root user authority to edit the BNU configuration files.
- If you use direct connections for BNU communications, the appropriate connections between your system and the remote systems must be set up.
- If you use modems for BNU communications, you must install and configure each modem.
- If one or more of your connections uses TCP/IP, then TCP/IP must be running between your system and the appropriate remote systems.
- Collect the information that you require to configure BNU (see the following list). This information includes a list of remote systems and lists of devices and modems to use for connecting to the systems.

Gathering required system information

Before you configure BNU, gather the following information:

- For each *remote system* that your system calls, collect the following information:
  - The system name
  - The login name that your system uses on the remote system
  - The password for the login name.
  - Login and password prompts on the remote system.
  - The type of connection you use to reach the remote system (direct, modem, or TCP/IP)

  If the connection is direct, collect the following information:
  - The bit rate of the connection
  - The port on the local system to which the connection is attached.

  If the connection is through a modem (telephone connection), collect the following information:
  - The telephone number of the remote system.
  - The speed of your modem that is compatible with the speed of the remote system.

  Note: If any of the remote systems calls your system, ensure that the BNU administrator on each of the remote systems has all the preceding information about your system.
- For each *local modem* that you use for BNU connections, collect the following information:
  - The chat script for the modem (consult the modem documentation).

    Note: For some modems, the chat script is available in the `/etc/uucp/Dialers` file.
  - The local port of the modem.

Creating a list of system devices

Use the information that you collected to make a list of each system device that you need to connect to a remote system. The following is a sample list for the local system `morgan`:

```
direct:
hera 9600 tty5
zeus& 2400 tty2
ariadne 2400 tty1
hayes modem (tty3):  apollo, athena
TCP/IP:  merlin, arthur, percy
```

In the previous example, to connect to system `hera`, a `direct` connection at a speed of `9600` from port `tty5` is used. To connect to system `apollo`, the `hayes` modem, which is connected to port `tty3`, is used. TCP/IP is used to connect to systems `merlin`, `arthur`, and `percy`.

Configuring remote communication facilities

For BNU to function correctly at your site, you must configure the remote communication facilities as follows:

- List the devices that are used to establish a direct, telephone, or modem communications link.
- List the modems that are used to contact remote systems over the telephone network.
- List the accessible remote systems.
- List the abbreviations by representing the prefixes of telephone numbers that are used to contact the specified remote systems (optional).
- Set access permissions by specifying the ways in which local and remote systems communicate.
- Schedule monitoring for the networked remote systems (optional).

To create these lists, permissions, and schedules, complete the following steps:

- Change the BNU configuration files.

- Edit the /var/spool/cron/crontabs/uucp file to remove the comment characters (#) from the beginning of the lines that schedule the automatic maintenance routines.

Note: You must configure the Systems, Devices, and Permissions files to ensure that BNU runs correctly at your site. However, it is not necessary to change the BNU configuration files in any particular order.

After you complete the preceding procedures, you can configure BNU on your system.

Configuring BNU on your system

To configure BNU, complete the following steps:

1. Ensure that BNU is installed on your system by running the following command:
   ```
   lslpp -h bos.net.uucp
   ```
   If BNU is installed, bos.net.uucp is displayed in the output. If you do not see it, install BNU from the installation tape.
2. Set appropriate login IDs and passwords for the remote systems that call your system, and provide the person responsible for administering BNU or the UNIX-to-UNIX Copy Program (UUCP) on each remote system with the login and password. This step is completed by editing the /etc/passwd, /etc/group, /etc/security/login.cfg, and /etc/security/passwd files.

   Attention: If you allow the remote systems to log in to the local system with the UUCP login ID, the security of your system is at risk. Remote systems that are logged in with the UUCP ID can display and possibly change the local Systems and Permissions files. These actions of the remote system depend on the permissions that are specified in the LOGNAME entry of the Permissions file. It is recommended that you create other BNU login IDs for remote systems and reserve the UUCP login ID for the person administering BNU on the local system. For the best security, each remote system that contacts the local system must have a unique login ID with a unique user ID (UID) number. These login IDs must have group IDs (GIDs) of 5. By default, the operating system includes the NUUCP login ID for transferring files.

   a. If you need to maintain complete control over the access of each individual system, you must create separate login IDs and combine the MACHINE and LOGNAME entries in the Permissions file. You have the option of maintaining separate logins, or having one login for all BNU connections. A few example /etc/passwd entries follow:
      ```
      Umicrtk:!:105:5:micrtk uucp:/usr/spool/uucppublic:/usr/sbin/uucp/uucico
      Ufloydl:!:106:5:floydl uucp:/usr/spool/uucppublic:/usr/sbin/uucp/uucico
      Uicus:!:107:5:icus uucp:/usr/spool/uucppublic:/usr/sbin/uucp/uucico
      Urisctkr:!:108:5::/usr/spool/uucppublic:/usr/sbin/uucp/uucico
      ```
   b. If you want to have one set of permissions and do not want to maintain separate control of any of the UUCP connections, you can have a single login for all the systems. An example entry for such a scenario follows:
      ```
      nuucp:!:6:5::/usr/spool/uucppublic:/usr/sbin/uucp/uucico
      ```
      Note:
      - The UID, which is the third colon-separated field, must be unique to avoid any security risk.
      - The GID, which is the fourth colon-separated field, must be 5 to ensure that it is in the same group as UUCP.
      - The home directory, which is the sixth colon-separated field, can be changed to any valid directory.
      - The login shell, which is the seventh colon-separated field, must always be /usr/sbin/uucp/uucico.
   c. Ensure that the /etc/group file contains the new users. An example of such an entry follows:
      ```
      uucp:!:5:uucp,uucpadm,nuucp,Umicrtk,Uicus,Urisctakr
      ```

d. Add any users to the UUCP group, who use modems to connect with programs other than the **cu** command.
e. After you edit these files as root, set a password for the new users with the command **passwd** *UserName*.

   Note: If you change a password from the root login, the flags entry in the stanza for the user in the `/etc/security/passwd` file contains the following line:

   ```
   flags = ADMCHG
   ```

   You must change the preceding line, as shown in the following example:

   ```
   flags =
   ```

   Otherwise, when the remote **uucico** process logs in to your system, the system prompts it to enter a new password. This action is not possible and hence the login fails.
f. To avoid interruptions in the login process that are caused by the **uucico** process, which might be started by the default herald with all of its Ctrl-J's, comment the default stanza (with asterisks) and define a stanza for your tty, as shown in the following example:

   ```
   /dev/tty0:
              herald = "\nrisc001 login:"
   ```
g. Use an ASCII text editor or the **uucpadm** command to edit the `Poll` file. Add an entry for each system that your system polls.

   Note: The systems that are listed in the `Poll` file must also be listed in the `/etc/uucp/Systems` file.
h. Use an ASCII text editor to edit the `/var/spool/cron/crontabs/uucp` file. Remove the comment characters (#) from the lines that run the **uudemon.hour** and **uudemon.poll** commands. You can change the number of times these commands are run. However, be sure to schedule the **uudemon.poll** command approximately 5 minutes before you schedule the **uudemon.hour** command.
i. Ensure that your changes took effect by running the following command:

   ```
   crontab -l uucp
   ```
j. Set up the following BNU data files: `Systems`, `Permissions`, `Devices`, `Dialers`, and `Sysfiles`. You might use the `/usr/sbin/uucp/uucpadm` command to initially set up the files and then edit them to suit your needs. Use the `Sysfiles` file to specify files other than `/etc/uucp/Systems`, `/etc/uucp/Devices`, and `/etc/uucp/Dialers` for BNU configuration. For more information, see `Sysfiles`.

3. If you decide to use dial-code abbreviations for telephone numbers in the `Systems` files, set up the `Dialcodes` entry for each abbreviation. For details, see Dialcodes File Format for BNU.

   If you use TCP/IP for your BNU connections, use the **netstat** command to see whether the **uucpd** daemon is working, by entering the following command:

   ```
   netstat -a
   ```

   The **uucpd** daemon is started by the **inetd** daemon. If the **uucpd** daemon does not run, reconfigure the **inetd** daemon to start the **uucpd** daemon. For more information, see "Configuring the inetd daemon" on page 333).
4. Use the list of devices that you collected, before you begin this procedure, to change the `Devices` file on your system. Make an entry for each modem and each direct connection. If you use TCP/IP, uncomment the TCP/IP entry in the `Devices` file. You can configure the `/etc/uucp/Sysfiles` file to specify other files to be used for configuration of devices. For details about the Devices file, see the Devices File Format for BNU.

   Also, if you use TCP/IP, verify that the `/etc/services` file includes the following line:

   ```
   uucp        540/tcp          uucpd
   ```

   If not, add this line to the file.

5. Use the information about each remote system that you collected, before you begin this procedure, to change the `Systems` file on your system. Use the commented examples in the `Systems` file as a guide when you specify your configuration. If you use TCP/IP, ensure that the host name table in the `/etc/hosts` file includes the name of the remote computer with which you want to connect. You can configure the `/etc/uucp/Sysfiles` file to specify other files to be used for configuration of systems.
6. Use the information about devices and modems that you collected, before you begin this procedure, to ensure that the `Dialers` file on your system contains an entry for each modem. If you use TCP/IP and direct connections, ensure that the TCP/IP entry and direct entries are present in the file. You can configure the `/etc/uucp/Sysfiles` file to specify other files to be used for configuration of dialers.
7. Decide how much access to your system you want to provide to each remote system that you call and to each remote system that calls you. Set up appropriate entries for each system and each login name in the `Permissions` file.
8. Use the **uucheck** command to verify that the directories, programs, and support files are set up properly:

   ```
   /usr/sbin/uucp/uucheck -v
   ```

   The **uucheck** command verifies that the directories, programs, and support files are set up properly and that the `Permissions` file entries are consistent. If the **uucheck** command reports any errors, fix the errors.
9. Optional: Set up automatic monitoring of BNU operations and automatic polling of remote systems. For more information, see "Setting up automatic monitoring of BNU" and "Setting up BNU polling of remote systems").

Setting up automatic monitoring of BNU

BNU uses the **cron** daemon to start BNU daemons and to monitor BNU activity.

Prerequisites

- Complete the steps listed in "Configuring BNU" on page 409.
- You must have root user authority to edit the `/var/spool/cron/crontabs/uucp` file.

The **cron** daemon reads the `/var/spool/cron/crontabs/uucp` file for instructions about when to start BNU procedures.

To set up automatic monitoring of BNU, complete the following steps:

1. Log in as a user with root user authority.
2. Use an ASCII text editor to edit the `/var/spool/cron/crontabs/uucp` file.
3. Uncomment the lines for the BNU maintenance procedures, `uudemon.admin` and `uudemon.cleanup`. You can change the times that these procedures are run if your system needs maintenance at more or less frequent intervals. It is best, however, to run the `uudemon.admin` command at least once a day and the `uudemon.cleanup` command at least once a week.
4. Use the `crontabs/uucp` file to schedule other BNU maintenance commands, such as the **uulog**, **uuclean**, or **uucleanup** commands. In addition, you can use the `crontabs/uucp` file to instruct the **cron** daemon to start the **uucico**, **uuxqt**, or **uusched** daemons at specific times.

Setting up BNU polling of remote systems

To enable BNU to poll remote systems for jobs, list the systems in the `/etc/uucp/Poll` file.

Prerequisites

- Complete the steps that are listed in "Configuring BNU" on page 409.
- You must have root authority to edit the `/var/spool/cron/crontabs/uucp` file and the `/etc/uucp/Poll` file.

In addition to listing the systems in the /etc/uucp/Poll file, run the **uudemon.hour** and **uudemon.poll** commands periodically.

To set up BNU polling of remote systems, complete the following steps:

1. Decide which remote systems to automatically poll. Decide how often you want to poll each system. Specify the times for each system with the Poll file, which can be as seldom as once a day or as often as you want.
2. Log in as a user with root authority.
3. Using an ASCII text editor or the **uucpadm** command, edit the Poll file. Add an entry for each system that your system is set up to poll.

   Note: The systems that are listed in the Poll file must also be listed in the /etc/uucp/Systems file.
4. Using an ASCII text editor, edit the /var/spool/cron/crontabs/uucp file. Remove the comment characters (#) from the lines that run the **uudemon.hour** and **uudemon.poll** commands. You can change the times that these commands are run. However, be sure to schedule the **uudemon.poll** command approximately 5 minutes before you schedule the **uudemon.hour** command.

BNU is now set up to automatically poll the systems that are listed in the Poll file at the times you specified.

/etc/uucp/Systems file

The remote systems are listed in the /etc/uucp/Systems files.

The /etc/uucp/Systems file is the default Systems file. The system administrator can specify additional files in the **/etc/uucp/Sysfiles** file.

Each entry in a Systems file contains the following items:

- The name of the remote system
- The times when users can connect to the remote system
- The type of link (direct line or modem)
- The speed of transmission over the link
- The information that is needed to log in to the remote system

Each entry in a Systems file represents one remote system. To establish communications, the remote system must be listed in the local Systems file. A Systems file must be present on every system that uses the BNU facility. Normally, only the root user can read the Systems files. Any user, however, can list the names of remote BNU systems, by using the **uuname** command.

Editing the Devices files for a direct connection

To edit the Devices file for a direct connection, you must have root authority to edit the /etc/uucp/Devices file or another file that is specified in the /etc/uucp/Sysfiles file as a Devices file.

To set up a direct connection that specifies a port and a remote system, make an entry as follows:

1. Enter the name of the remote system to which you want to connect the local computer over the direct line in the**Type** field in the second line of the entry.
2. Enter the device name appropriate for the direct connection that is used at your site in the **Line** field in both lines of the entry.
3. Enter a hyphen (-) for a placeholder in the **Line2** field in both lines of the entry.
4. Enter the transmission rate appropriate for the direct connection that is used at your site in the **Speed** field in both lines of the entry.
5. Enter direct (all lowercase) in the **Dialer-Token Pairs** field in both lines of the entry.

   For example:

```
type device - speed direct
```

Continue adding entries to the Devices file until you have listed each device that connects the local system directly to a remote system.

To set up a direct connection between two systems that use a permanent asynchronous serial connection, make a one-line entry as follows:

1. Enter the name of the remote system in the **Type** field.
2. Enter the name of the tty device in the **Line** field.
3. Enter a hyphen (-) for a placeholder in the **Line2** field.
4. Enter the transmission rate appropriate for the direct connection that is used at your site in the **Class** field.
5. Enter direct (all lowercase) in the **Dialer-Token Pairs** field. For example:

   ```
   type device - speed direct
   ```

Continue adding entries to the Devices file until you have listed each direct device that connects the local system to a remote system.

Editing the Devices file for an autodialer connection

Follow these steps when you edit the /etc/uucp/Devices file.

You must have root authority to edit the /etc/uucp/Devices file or another file that is specified in the /etc/uucp/Sysfiles file as a Devices file.

In telephone-connection entries, the **Type** field is specified as an automatic calling unit (ACU). Enter ACU as the **Type** field entry in all remote connections that are established over a phone line. To set up Devices file entries for autodialer connections, make a one-line entry for each modem:

1. In the **Type** field, enter ACU.
2. In the **Line** field, enter the device name that is attached to the modem.
3. In the **Line2** field, enter a hyphen (-) as a placeholder, unless the autodialer is a standard 801 dialer. If the autodialer is a standard 801 dialer, enter 801.
4. In the **Speed** field, enter the baud rate appropriate for your modem and line, or the class of your modem (for example, D2400). The value of baud rate can be 300, 1200, 2400, or higher, depending on the modem.

   Note: If the modem can be used at more than one specified baud rate, make a separate entry in the Devices file for each rate. If the modem can be used at any baud rate, enter the word Any in the **Speed** field.
5. Enter the name of the modem as the **Dialer** field entry in the **Dialer-Token Pair** field. If you are planning to include complete phone numbers in the /etc/uucp/Systems file or another Systems file, which is specified in the /etc/uucp/Sysfiles file, leave the **Token** field blank. A blank instructs the BNU program to use the default \D token. If you are planning to use the dialing-code abbreviations that are specified in the /etc/uucp/Dialcodes file, enter the \T token.

   For example:

   ```
   type line - speed dialer - token pair
   ```

Continue adding entries to the Devices file until you have listed each connection between the local system and a remote system that uses a telephone line and a modem.

Editing the Devices file for TCP/IP

Follow these steps when you edit the /etc/uucp/Devices file.

You must have root authority to edit the /etc/uucp/Devices file, or another file that is specified in the /etc/uucp/Sysfiles file as a Devices file.

If your site is using the TCP/IP to connect systems, include the relevant TCP/IP entry in the Devices file. To set up the file for use with the TCP/IP, enter the following line in the Devices file:

```
TCP - - - TCP
```

Examples: BNU configuration for a TCP/IP connection

This group of examples configures BNU for a TCP/IP connection.

The following files are set up for a TCP/IP connection between systems zeus and hera, where zeus is the local system and hera is the remote system.

BNU files for TCP/IP connection entries in the local system files:

These BNU files are entries on the local system zeus.

- **Systems file:** The Systems file on system zeus contains the following entry so that zeus can contact system hera:

  ```
  hera Any TCP,t - - in:--in: uzeus word: birthday
  ```

  This example specifies that system zeus can call system hera at any time, by using the **t** protocol for communications with system hera. System zeus logs in to system hera as uzeus with the password birthday.

  Note: The **t** protocol supports **TCP**. Therefore, always use the **t** protocol for BNU communications over TCP/IP connections. However, the **t** protocol cannot be used when the **Type** field is ACU (automatic calling unit) or when a modem connection is used.

  BNU uses the **Type** and **Class** fields in the Systems file to find the appropriate device for the connection. It checks the Devices file for an entry of type TCP.

- **Devices file:** The Devices file that is used by the **uucico** daemon on system zeus contains the following entry for TCP/IP connections:

  ```
  TCP  -  -  -  TCP
  ```

  Because the device type is TCP, there are no *Class*, *Line*, or *Line2* entries. The *Dialer* is also specified as TCP. BNU looks in the Dialers files for a TCP entry.

- **Dialers file:** The Dialers file that is used by the **uucico** daemon on system zeus contains a TCP/IP entry as follows:

  ```
  TCP
  ```

  This entry specifies that no dialer configuration is required.

  Note: Dialer configuration is never required over a TCP/IP connection.

- **Permissions file:** The Permissions file on system zeus contains the following entry, which grants system hera access to system zeus:

  ```
  LOGNAME=uhera SENDFILES=yes REQUEST=yes \
  MACHINE=zeus:hera VALIDATE=uhera \
  READ=/var/spool/uucppublic:/home/hera \
  WRITE=/var/spool/uucppublic:/home/hera COMMANDS=ALL
  ```

  The combined LOGNAME and MACHINE entries provide the following permissions to system hera, when system zeus and system hera are connected:

  - System hera can request and send files regardless of who initiated the call.
  - System hera can read and write to the public directory and to the /home/hera directory on system zeus.
  - System hera can run all commands on system zeus.
  - System hera must log in to system zeus as user uhera, and system hera cannot use any other login ID for BNU transactions.

Note: Because the permissions are the same regardless of which system initiates the call, the preceding LOGNAME and MACHINE entries are combined. If the permissions are not the same for system hera and system zeus, the LOGNAME and MACHINE entries are as follows:

```
LOGNAME=uhera VALIDATE=hera SENDFILES=yes REQUEST=yes \
READ=/var/spool/uucppublic:/home/hera \
WRITE=/var/spool/uucppublic:/home/hera

MACHINE=zeus:hera REQUEST=yes COMMANDS=ALL\
READ=/var/spool/uucppublic:/home/hera \
WRITE=/var/spool/uucppublic:/home/hera
```

BNU files for TCP/IP connection entries in the remote system's files:

These files are on the remote system hera.

- **Systems file:** A Systems file on system hera contains the following entry to allow hera to contact system zeus:

  ```
  zeus Any TCP,t - - ogin:--ogin: uhera ord: lightning
  ```

  This example specifies that system hera can call system zeus at any time, by using the **t** protocol for communications with system zeus. System hera logs in to system zeus as user uhera with the password lightning. Again, BNU next checks the Devices files for an entry of type TCP.

  Note: The **t** protocol supports the **TCP**. Therefore, always use the **t** protocol for BNU communications over TCP/IP connections. However, the **t** protocol cannot be used when the *Type* field is ACU or when a modem connection is being used.

- **Devices file:** The Devices file that is used by the **uucico** daemon on system hera contains the following entry for TCP/IP connections:

  ```
  TCP  -  -  -  TCP
  ```

  Because the device type is TCP, there are no *Type*, *Line*, or *Line2* entries. The *Dialer* is also specified as TCP. BNU looks in the Dialers files for a TCP entry.

- **Dialers file**: The Dialers file that is used by the **uucico** daemon on system hera contains a TCP/IP entry as follows:

  ```
  TCP
  ```

  This entry specifies that no dialer configuration is required.

  Note: Dialer configuration is never required over a TCP/IP connection.

- **Permissions file**: The Permissions file on system hera contains the following entry, which grants system zeus access to system hera:

  ```
  LOGNAME=uzeus SENDFILES=yes REQUEST=yes \
  MACHINE=hera:zeus VALIDATE=zeus COMMANDS=rmail:who:uucp
  ```

  The combined LOGNAME and MACHINE entries provide the following permissions to system zeus, when system zeus and system hera are connected:

  – System zeus can request and send files regardless of who initiated the call.
  – System zeus can read and write only to the public directory (the default).
  – System zeus can run only the **rmail**, **who**, and **uucp** commands.
  – System zeus must log in to system hera as user uzeus and system zeus cannot use any other login ID for BNU transactions.

  Note: If the permissions are not the same for system hera and system zeus, the LOGNAME and MACHINE entries are as follows:

  ```
  LOGNAME=uzeus VALIDATE=zeus SENDFILES=yes REQUEST=yes
  MACHINE=hera:zeus COMMANDS=rmail:who:uucp REQUEST=yes
  ```

Examples: BNU configuration for a telephone connection

Example files are set up to connect systems venus and merlin over a telephone line by using modems.

The system venus is the local system, and the system merlin is the remote system.

On both systems, the device tty1 is connected to a Hayes modem at 1200 baud. The login ID used for system venus to log in to system merlin is uvenus, and the associated password is mirror. The login ID for system merlin to log in to system venus is umerlin, and the associated password is oaktree. The phone number for the modem that is attached to venus is 9=3251436; the number of the merlin modem is 9=4458784. Both computers include partial phone numbers in their Systems files and include dial-codes in their Dialcodes files.

The following example files are set up to connect systems venus and merlin:

- **Systems File:** The Systems file on system venus contains the following entry for system merlin, including a phone number and a dialing prefix:

  ```
  merlin Any ACU 1200 local8784 "" in:--in: uvenus word: mirror
  ```

  System venus can call system merlin at any time, by using an ACU device at 1200 baud and logging in as uvenus with the password mirror. The telephone number is expanded based on the code local in the Dialcodes file, and the device to be used is determined based on the *Type* and *Class* entries. BNU checks the Devices files for a device of type ACU and class 1200.
- **Dialcodes File:** The Dialcodes file on system venus contains the following dial-code prefix for use with the number in the Systems file:

  ```
  local 9=445
  ```

  Given this code, the telephone number for system merlin in the Systems file is expanded to 9=4458784.
- **Devices file:** The Devices file on system venus contains the following entry for the connection to system merlin:

  ```
  ACU tty1  -  1200  hayes \T
  ```

  The port to be used is tty1, and the *Dialer* entry in the *Dialer-Token Pairs* field is hayes. The *Token* entry, \T, indicates that the telephone number is to be expanded by using a code from the Dialcodes file. BNU checks the Dialers files for a hayes dialer type.
- **Dialers file:** The Dialers file that is used by the **uucico** daemon on system venus contains the following entry for the hayes modem:

  ```
  hayes =,-, "" \dAT\r\c OK \pATDT\T\r\c CONNECT
  ```

  Note: The expect-send characters are defined in the Dialers file format.
- **Permissions file:** The Permissions file on system venus contains the following entries, which specify the ways in which system merlin can conduct **uucico** and **uuxqt** transactions with system venus:

  ```
  LOGNAME=umerlin REQUEST=yes SENDFILES=yes \
  READ=/var/spool/uucppublic:/home/merlin \
  WRITE=/var/spool/uucppublic:/home/merlin
  MACHINE=venus:merlin VALIDATE=umerlin REQUEST=yes SENDFILES=yes    \
  COMMANDS=ALL \
  READ=/var/spool/uucppublic:/home/merlin \
  WRITE=/var/spool/uucppublic:/home/merlin
  ```

  System merlin logs in to system venus as umerlin, which is a unique login for system merlin. System merlin can request and send files regardless of who initiated the call. Also, system merlin can read and write to the /var/spool/uucppublic directory and to the /home/merlin directory on system venus. System merlin can issue all commands in the default command-set on system venus.

BNU files with telephone connection entries on the local system:

These files contain telephone connection entries on the local system venus.

- **Systems file:** A Systems file on system venus contains the following entry for system merlin, including a phone number and a dialing prefix:

```
merlin Any ACU 1200 local8784 "" in:--in: uvenus word: mirror
```

System venus can call system merlin at any time, by using an ACU device at 1200 baud and logging in as user uvenus with the password mirror. The telephone number is expanded based on the code local in the Dialcodes file, and the device to be used is determined based on the *Type* and *Class* entries. BNU checks the Devices files for a device of type ACU and class 1200.

- **Dialcodes file:** The Dialcodes file on system venus contains the following dial-code prefix for use with the number in the Systems file:

  ```
  local 9=445
  ```

  Given this code, the telephone number for system merlin in the Systems file is expanded to 9=4458784.

- **Devices file:** The Devices file on system venus contains the following entry for the connection to system merlin:

  ```
  ACU  tty1  -  1200  hayes \T
  ```

  The port to be used is tty1, and the *Dialer* entry in the **Dialer-Token Pairs** field is hayes. The *Token* entry, \T, indicates that the telephone number is to be expanded by using a code from the Dialcodes file. BNU checks the Dialers files for an entry of a hayes dialer type.

- **Dialers file:** The Dialers file that is used by the **uucico** daemon on system venus contains the following entry for the hayes modem:

  ```
  hayes  =,-,  ""  \dAT\r\c OK \pATDT\T\r\c CONNECT
  ```

  Note: The expect-send characters are defined in the Dialers file format.

- **Permissions file:** The Permissions file on system venus contains the following entries, which specify the ways in which system merlin can conduct **uucico** and **uuxqt** transactions with system venus:

  ```
  LOGNAME=umerlin REQUEST=yes SENDFILES=yes \
  READ=/var/spool/uucppublic:/home/merlin \
  WRITE=/var/spool/uucppublic:/home/merlin
  MACHINE=venus:merlin VALIDATE=umerlin REQUEST=yes SENDFILES=yes    \
  COMMANDS=ALL \
  READ=/var/spool/uucppublic:/home/merlin \
  WRITE=/var/spool/uucppublic:/home/merlin
  ```

  System merlin logs in to system venus as umerlin, which is a unique login for system merlin. System merlin can request and send files regardless of who initiated the call. Also, system merlin can read and write to the /var/spool/uucppublic directory and to the /home/merlin directory on system venus. System merlin can issue all commands in the default command-set on system venus.

BNU files with telephone connection entries on the remote system:

These files contain telephone connection entries on the remote system merlin.

- **Systems file:** The Systems file on system merlin contains the following entry for system venus, including a phone number and a dialing prefix:

  ```
  venus Any ACU 1200 intown4362 "" in:--in: umerlin word: oaktree
  ```

  System merlin can call system venus at any time, by using an ACU device at 1200 baud and logging in as user umerlin with the password oaktree. The telephone number is expanded based on the code intown in the Dialcodes file, and the device to be used is determined based on the *Type* and *Class* entries. BNU checks the Devices files for a device of type ACU and class 1200.

- **Dialcodes file:** The Dialcodes file on system merlin contains the following dial-code prefix for use with the number in the Systems file:

  ```
  intown 9=325
  ```

  Therefore, the expanded telephone number to reach system venus is 9=3254362.

- **Devices file:** The Devices file on system merlin contains the following entry for the connection to system venus:

  ```
  ACU  tty1  -  1200  hayes \T
  ```

The ACU is attached to port tty1, and the dialer is hayes. The telephone number is expanded with information from the Dialcodes file. BNU checks the Dialers files for an entry of the hayes modem.

- **Dialers file:** The Dialers file that is used by the **uucico** daemon on system merlin contains the following entry for its modem:

  ```
  hayes  =,-,  ""  \dAT\r\c OK \pATDT\T\r\c CONNECT
  ```

- **Permissions file:** The Permissions file on system merlin contains the following entries, which grant system venus access to system merlin:

  ```
  LOGNAME=uvenus SENDFILES=call REQUEST=no \
  WRITE=/var/spool/uucppublic:/home/venus \
  READ=/var/spool/uucppublic:/home/venus
  MACHINE=merlin:venus VALIDATE=uvenus  \
  READ=/ WRITE=/ COMMANDS=ALL REQUEST=yes \
  NOREAD=/etc/uucp:/usr/etc/secure \
  NOWRITE=/etc/uucp:/usr/etc/secure
  ```

Examples: BNU configuration for a direct connection

The following example files are set up for a direct connection between systems zeus and hera, where zeus is the local system and hera is the remote system.

The direct device on system zeus is tty5. On system hera, the direct device is tty1. The speed of the connection is 1200 bps. The login ID for system zeus on system hera is uzeus, and the associated password is thunder. The login ID for system hera on system zeus is uhera, and the associated password is portent.

BNU files with direct connection in the local system's files:

These files contain telephone connection entries on the local system zeus.

- **Systems file:** The Systems file on system zeus contains the following entry for the remote system hera:

  ```
  hera Any hera 1200 - "" \r\d\r\d\r in:--in: uzeus word:  thunder
  ```

  This entry specifies that system hera can log in to system zeus at any time, by using a direct connection, which is specified in the Devices files. To find the entry in the Devices files, BNU uses the third and fourth fields of the Systems entry. Thus, BNU looks for an entry in the Devices files with a *Type* of hera and a *Class* of 1200. System zeus logs in to system hera as user uzeus with the password thunder.

- **Devices file:** The Devices file on system zeus contains the following entry to connect to the remote system hera:

  ```
  hera    tty5  -  1200  direct
  ```

  This entry specifies that system zeus uses device tty5 at 1200 bps to communicate with system hera. Note that the *Dialer* in both **Dialer-Token Pairs** fields is direct. When you connect to system hera, BNU checks the Dialers file for a direct entry.

- **Dialers file:** The Dialers file on system zeus contains the following entry for direct connections:

  ```
  direct
  ```

  This entry specifies that no handshaking is required for direct connection.

- **Permissions file:** The Permissions file on the local system zeus contains the following entry, which specifies the ways in which the remote system hera can conduct **uucico** and **uuxqt** transactions with system zeus:

  ```
  LOGNAME=uhera MACHINE=hera VALIDATE=uhera REQUEST=yes \
  SENDFILES=yes MACHINE=zeus READ=/ WRITE=/ COMMANDS=ALL
  ```

  This entry specifies that system hera logs in as uhera. Because the VALIDATE=uhera option is included, system hera cannot log in to system zeus with any other login ID, nor can any other remote system use the uhera ID. System hera can read and write to any directory on system zeus, and can send and request files regardless of who initiated the call. System hera can also initiate any commands on system zeus.

Note: Because the permissions that are granted are the same regardless of which system initiated the connection, the LOGNAME and MACHINE entries have been combined. If the permissions are not the same for system hera and system zeus, the LOGNAME and MACHINE entries are as follows:

```
LOGNAME=uhera REQUEST=yes SENDFILES=yes READ=/ WRITE=/
MACHINE=zeus:hera VALIDATE=uhera READ=/ WRITE=/ REQUEST=yes \
COMMANDS=ALL
```

Attention: Providing the permissions in the preceding example is equivalent to giving any user on the remote system a login ID on the local system. Such liberal permissions can jeopardize your security and are given only to well-trusted remote systems at the same site.

BNU files with direct connection in the remote system files:

These files contain telephone connection entries on the remote system hera.

- **Systems file:** The Systems file on system hera contains the following entry for system zeus:

  ```
  zeus Any zeus 1200 - "" \r\d\r\d\r in:--in: uhera word: portent
  ```

  This entry specifies that system hera can log in to system zeus at any time, by using a direct connection, which is specified in the Devices files. To find the entry in the Devices files, BNU uses the third and fourth fields of the Systems entry. Thus, BNU looks for an entry in the Devices files with the **Type** field of value zeus and a **Class** field of value 1200. System hera logs in to system zeus as user uhera with the password portent.
- **Devices file:** The Devices file on system hera contains the following entry for communications with the system zeus:

  ```
  zeus    tty1  -  1200  direct
  ```

  This entry specifies that system hera uses device tty1 at 1200 bps to communicate with system zeus. Because the *Dialer* is specified as direct, BNU checks the Dialers files for a direct entry.
- **Dialers file:** The Dialers file on system hera contains the following entry for direct connections:

  ```
  direct
  ```

  This entry specifies that no dialer configuration is required on the direct connection.
- **Permissions file:** The Permissions file on system hera contains the following entries, which specify the ways in which zeus can conduct **uucico** and **uuxqt** transactions with system hera:

  ```
  LOGNAME=uzeus REQUEST=yes SENDFILES=yes READ=/ WRITE=/
  MACHINE=hera:zeus VALIDATE=uzeus REQUEST=yes COMMANDS=ALL READ=/\
  WRITE=/
  ```

  These entries specify that system zeus logs in to system hera as uzeus. Because the VALIDATE=uzeus parameter is included, system zeus cannot log in to system hera with any other login ID, nor can any other remote system use the uzeus ID. System zeus can read and write to any directory on system hera, and can send and request files regardless of who initiated the call. System zeus can also initiate any commands on system hera.

  Attention: If you provide all the permissions in the preceding example, it is equivalent to giving any user on the remote system a login ID on the local system. Such liberal permissions can jeopardize your security and are given only to remote systems at the same site.

BNU maintenance

BNU must be maintained to work correctly on your system.

To maintain BNU:

- Read and remove log files periodically.
- Use the **uuq** and **uustat** commands to check the BNU queues to ensure jobs are transferring to remote systems correctly.
- Schedule automatic commands that poll remote systems for jobs, return unsent files to users, and send you periodic messages about BNU status.

- Periodically update the configuration files to reflect changes in your system.

In addition, occasionally check with administrators of remote systems to keep up with changes on their systems that might affect your configuration. For example, if the supervisor of system `venus` changes your system password, you must put the new password in the `/etc/uucp/Systems` file (or the appropriate Systems file specified by `/etc/uucp/Sysfiles`) before your system can log in to system `venus`.

BNU log files

BNU creates log files and error files to track its own activities.

These files must be checked and removed periodically to keep them from filling the storage space on your system. BNU provides several commands for use in cleaning log files:

- `uulog`
- `uuclean`
- `uucleanup`
- `uudemon.cleanu.`

Run these commands manually or use entries in the `/var/spool/cron/crontabs/uucp` file to run the commands by the **cron** daemon.

Log files in the .Log and .Old directories:

BNU creates individual log files in the `/var/spool/uucp/.Log` directory.

BNU creates these log files for each accessible remote system, using the **uucp**, **uucico**, **uux**, and **uuxqt** commands. BNU places status information about each transaction in the appropriate log file each time someone on the system uses BNU. When more than one BNU process is running the system cannot access the log file. Instead, it places the status information in a separate file with a `.LOG` prefix.

The **uulog** command displays a summary of **uucp** or **uux** requests, by user or by system. The **uulog** command displays the files. However, you can also have BNU automatically combine the log files into a primary log file. This is called *compacting* the log files and can be done with the **uudemon.cleanu** command, usually run by the **cron** daemon.

The **cron** daemon runs the **uudemon.cleanu** command. The **uudemon.cleanu** command combines the **uucico** and **uuxqt** log files on the local system and stores them in the `/var/spool/uucp/.Old` directory. At the same time, the command removes old log files previously stored in the `.Old` directory. By default, the **uudemon.cleanu** command saves log files that are two days old.

If storage space is a problem, consider reducing the number of days that files are kept. To track BNU transactions over a longer period of time, consider increasing the number of days that files are kept. To change the default time for saving log files, modify the shell procedure for the **uudemon.cleanu** command. This script is stored in the `/usr/sbin/uucp` directory and can be modified with root authority.

BNU /.Admin log files:

BNU also collects information and stores it in the `/var/spool/uucp/.Admin` directory. This directory contains the `errors`, `xferstats`, `Foreign`, and `audit` files.

These files must be checked and removed occasionally to save storage space. BNU creates each file when it is needed.

When another system contacts your system with the **uucico** daemon debugging mode on, it invokes the **uucico** daemon on your system with debugging turned on. The debugging messages generated by the daemon on the local system are stored in the `audit` file. This file can get quite large. Check and remove the `audit` file often.

The `errors` file records errors encountered by the **uucico** daemon. Checking this file can help you correct problems such as incorrect permissions on BNU work files.

The `xferstats` file contains information about the status of every file transfer. Check and remove this file occasionally.

The `Foreign` file is important to the security of your system. Whenever an unknown system attempts to log in to the local system, BNU calls the `remote.unknown` shell procedure. This shell procedure logs the attempt in the `Foreign` file. The `Foreign` file contains the names of the systems that have attempted to call the local system and been refused. If a system has been attempting frequent calls, use this information when considering whether to allow that system access.

Systemwide log files used by BNU:

Because many BNU processes need root authority to complete their tasks, BNU creates frequent entries in the `/var/spool/sulog log` file.

Similarly, using the **cron** daemon to schedule BNU tasks creates multiple entries in the `/var/spool/cron/log` file. When using BNU, check and clean these files.

BNU maintenance commands

The Basic Networking Utilities contain several commands for monitoring BNU activities and cleaning BNU directories and files.

BNU cleanup commands:

BNU contains three commands that clean directories and remove files that have not been sent.

Item	Description
uuclean	Deletes all files older than a specified number of hours, from the BNU administrative directories. Use the **uuclean** command to specify a directory to be cleaned or a type of file to be deleted. You can also instruct the command to notify the owners of the deleted files. The **uuclean** command is the Berkeley equivalent of the **uucleanup** command.
uucleanup	Performs functions similar to the **uuclean** command. However, the **uucleanup** command checks the age of files based on *days* rather than hours. Use the **uucleanup** command to send a warning message to users whose files have not been transferred, notifying them that the files are still in the queue. The **uucleanup** command also removes files relating to a specified remote system.
uudemon.cleanu	A shell procedure that issues the **uulog** and **uucleanup** commands to compress the BNU log files and remove log and work files over three days old. The **uudemon.cleanu** command is run by the **cron** daemon.

BNU status-checking commands:

BNU also provides commands for checking the status of transfers and log files.

Item	Description
uuq	Displays jobs currently in the BNU job queue. Use the **uuq** command to display the status of a specified job or of all jobs. With root authority, you can use the **uuq** command to delete a job from the queue.
uustat	Provides information similar to that provided by the **uuq** command, in a different format. Use the **uustat** command to check the status of jobs and delete jobs you own. With root authority, you can also delete jobs belonging to other users.
uulog	Displays a summary of **uucp** or **uux** requests, by user or by system. The **uulog** command displays the file names. See "BNU log files" on page 422.
uupoll	Forces a poll of a remote system. This is helpful when work for that system is waiting in the queue and needs to be transferred, before the system is scheduled to be called automatically.
uusnap	Displays a very brief summary of BNU status. For each remote system, this command shows the number of files awaiting transfer. However, it does not show how long they have been waiting. The **uusnap** command is the Berkeley equivalent of the **uustat** command.

BNU shell procedures:

BNU is delivered with two shell procedures used for maintenance:

Item	Description
uudemon.cleanu	Discussed in "BNU cleanup commands" on page 423.
uudemon.admin	Issues the **uustat** command The **uustat** command reports the status of BNU jobs. It sends the results to the uucp login ID as mail. You can modify the **uudemon.admin** shell procedure to send the mail elsewhere, or use a mail program to reroute all mail for the uucp login ID to the user responsible for BNU administration.

These shell procedures are stored in the `/usr/sbin/uucp` directory. Copy the procedures and modify the copy, if you want to change what they do. Run the procedures from the command line or schedule them to be run by the **cron** daemon.

To automatically run the **uudemon.cleanu** and **uudemon.admin** commands, remove the comment characters (#) from the beginning of the relevant lines in the `/var/spool/cron/crontabs/uucp` file.

BNU path names

Path names used with Basic Networking Utilities (BNU) commands can be entered in a number of different ways.

Path names consist of either the root directory or a shortcut path to the target, which is the name of a remote system or systems. Each path variation follows specific guidelines.

Full path name

A full path name starts at the root and traces all the directories down to the target directory and file.

For example, `/etc/uucp/Devices` refers to the `Devices` file in the `uucp` directory in the root directory `etc`.

Always type a preceding forward slash (`/`) to signify a root directory. Always separate elements in paths with a forward slash (`/`).

Relative path name

The relative path name lists only those directories that are relative to the current directory.

For example, if your current directory is `/usr/bin` and your target directory is `/usr/bin/reports`, type the relative path name `reports` (without the leading slash).

Relative path names can be used with the **cu**, **uucp**, and **uux** commands, and with the name of the source file in the **uuto** command.

Note: Relative path names might not work with all BNU commands. If you have trouble using a relative path name, type the command again with the full path name.

~ [option] path name

The ~ [option] path name represents the home directory of the specified user.

The tilde (~) can be used as a shortcut to certain directories.

For example, ~jane refers to the home directory of the user jane. The entry, ~uucp or ~ (tilde alone) refers to the BNU public directory on the remote system. The full path name for the BNU public directory is /var/spool/uucppublic.

Note: This use of the tilde should not be confused with the other use of the tilde in BNU. The tilde is also used to preface commands for execution on a local system when logged in to a remote system when using the **cu** command.

system_name! path name

The *system_name*! path name identifies the path to a file on another system.

For example, distant!/account/march refers to the march file in the account directory on the remote system distant.

system_name!system_name! path name

The *system_name*!*system_name*! path name identifies a path through multiple systems.

For example, if the system named distant can only be reached through another system called near, then the path name is near!distant!/account/march.

Separate system names with an exclamation mark (!). In the case of multiple-system path names, the rule of separating elements with a forward slash (/) does not apply to the system names. However, the rule does hold for the termination system, where directories and files are stipulated.

Note: When you use a bourne shell, separate system names with an exclamation mark (!). When you use BNU in either a C or korn shell, precede the exclamation mark with a backward slash (\). The backward slash is an escape character necessary to interpret the next character literally rather than as the special character.

BNU daemons

The BNU software includes four daemons stored in the /usr/sbin/uucp directory.

Item	Description
uucico	Facilitates file transfers (see "uucico daemon")
uusched	Facilitates work request scheduling of files queued in the local spooling directory (see "uusched daemon" on page 426)
uuxqt	Facilitates remote command executions (see "uuxqt daemon" on page 426)
uucpd	Facilitates communications using TCP/IP (see "uucpd daemon" on page 427)

The **uucico**, **uusched**, and **uuxqt** daemons are started by the **cron** daemon according to a schedule set by the BNU administrator. With root authority, you can also start these daemons manually. The **uucpd** daemon should be started by the TCP/IP **inetd** daemon.

uucico daemon

The **uucico** daemon transports the files required to send data from one system to another.

The **uucp** and **uux** commands start the **uucico** daemon to transfer command, data, and execute files to the designated system. The **uucico** daemon is also started periodically by the BNU scheduler, the

uusched daemon. When started by the **uusched** daemon, the **uucico** daemon attempts to contact other systems and execute the instructions in the command files.

To run the instructions in the command files, the **uucico** daemon first checks the `/etc/uucp/Systems` file (or one or more other files specified by `/etc/uucp/Sysfiles`) for the system to be called. The daemon then checks the `Systems` file entry for a valid time to call. If the time is valid, the **uucico** daemon checks the *Type* and *Class* fields and accesses the `/etc/uucp/Devices` file (or one or more other files specified by `/etc/uucp/Sysfiles`) for a device that matches.

After finding a device, the **uucico** daemon checks the `/var/locks` directory for a lock file for the device. If one exists, the daemon checks for another device of the requested type and speed.

When no device is available, the daemon returns to the `Systems` files for another entry for the remote system. If one exists, the daemon repeats the process of searching for a device. If another entry is not found, the daemon makes an entry in the `/var/spool/uucp/.Status/`*SystemName* file for that remote system and goes on to the next request. The command file remains in the queue. The **uucico** daemon attempts the transfer again at a later time. The later attempt is called a *retry*.

When the **uucico** daemon reaches the remote system, it uses the instructions in the `Systems` files to log in. This causes an instance of the **uucico** daemon to be invoked on the remote system as well.

The two **uucico** daemons, one on each system, work together to make the transfer. The **uucico** daemon on the calling system controls the link, specifying the requests to be performed. The **uucico** daemon on the remote system checks the local permissions for whether they allow the request to be performed. If so, the file transfer starts.

After the **uucico** daemon on the calling system has finished transferring all requests it has for the remote system, it sends a hangup request. When the remote **uucico** daemon has transactions to send to the calling system, it denies the hangup request, and the two daemons reverse roles.

Note: Either the `/etc/uucp/Permissions` file on the local system or the `/etc/uucp/Permissions` file on the remote system can forbid the daemons to reverse roles. In this case, the remote system must wait to transfer files until it calls the local system.

When nothing is left to be transferred in either direction, the two **uucico** daemons hang up. At this point, the **uuxqt** daemon (“uuxqt daemon”) is called to execute remote command requests.

Throughout the transfer process, the **uucico** daemons on both systems log messages in the BNU log and error files.

uusched daemon

The **uusched** daemon schedules the transfer of files that are queued in the spooling directory on the local system.

The spooling directory is `/var/spool/uucppublic`. When the **uusched** daemon is invoked, it scans the spooling directory for command files, then randomizes the files and starts the **uucico** daemon. The **uucico** daemon transfers the files.

uuxqt daemon

When a user issues the **uux** command to run a specified command on a designated system, the **uuxqt** daemon runs the command.

After creating the necessary files, the **uux** command starts the **uucico** daemon, which transfers those files to the public spooling directory on the specified system.

The **uuxqt** daemon periodically searches the spooling directory for command-execution requests on every connected system. When it locates such a request, the **uuxqt** daemon checks for necessary files and permissions. Then, if permitted, the daemon runs the specified command.

uucpd daemon

The **uucpd** daemon must be able to run on the remote system before BNU can establish communications with a remote computer with **Transmission Control Protocol/Internet Protocol (TCP/IP)**.

The **uucpd** daemon is a subserver of the TCP/IP **inetd** daemon and is started by the **inetd** daemon.

By default, the **uucpd** daemon is commented in the `inetd.conf` file. To use it, you must remove the comment character and restart **inetd**. However, if this has been changed on your system, you might need to reconfigure the **inetd** daemon to start the **uucpd** daemon.

BNU security

Because other systems contact your system to log in, transfer files, and enter commands, BNU provides a means to establish security.

BNU security enables you to restrict what users of remote systems can do on the local system (users of remote systems can also restrict what you can do on their systems). BNU runs several daemons to complete its activities and uses administrative directories to store the files it needs. BNU also keeps a log of its own activities.

BNU security works on several levels. When you configure BNU, you can determine:

- Who on your system has access to BNU files.
- Which remote systems your system can contact.
- How users on remote systems log in to your system.
- What users on remote systems can do on your system once they log in.

uucp login ID

When BNU is installed, all of the configuration files, daemons, and many of the commands and shell procedures are owned by the uucp login ID.

The uucp login ID has a user ID (UID) of 5 and a group ID (GID) of 5. The **cron** daemon reads the `/var/spool/cron/crontabs/uucp` file to schedule automatic jobs for BNU.

Usually, logging in as user uucp is not allowed. To change files that are owned by the uucp login ID, log in with root authority.

Attention: Allowing remote systems to log in to the local system with the uucp login ID seriously jeopardizes the security of the local system. Remote systems logged in with the uucp ID can display and possibly modify the local `Systems` and `Permissions` files depending on the other permissions specified in the LOGNAME entry. It is strongly recommended that you create other BNU login IDs for remote systems and reserve the uucp login ID for the person responsible for administering BNU on the local system. For the best security, each remote system that contacts the local system must have a unique login ID with a unique UID number.

The operating system provides a default nuucp login ID for transferring files.

BNU login IDs

The startup shell for BNU login IDs is the **uucico** daemon (`/usr/sbin/uucp/uucico`).

When remote systems call your system, they automatically start the **uucico** daemon on your system. Login IDs for BNU have a uucp group ID of 5.

Login IDs used by remote systems need passwords. In order to prevent security from prompting a new BNU login ID for a new password when the remote system logs in, you must set the password as soon as you create the account. To do this, use the **passwd** command followed by the **pwdadm** command. For example, to set a password for the login ID `nuucp`, log in as the root user and enter the following commands:

```
passwd nuucp
```

```
pwadm -f NOCHECK
nuucp
```

The system prompts you for a password for the `nuucp` login ID. Completing these steps allows the remote system to log in without being immediately prompted for a new password (which the batch-oriented `nuucp` login ID cannot provide).

After creating the login ID for a remote system, notify that system BNU administrator of the login ID and password to access your system.

A user with root authority can set up a BNU administrative login ID. This is useful if you want to delegate BNU administration duties to a user without root authority. The BNU administrative login ID should have password security, a UID of 5, and be in a uucp group ID 5. The login shell for the administrative login should be the `/usr/bin/sh` program (instead of the **uucico** daemon). Giving the BNU administrative login a UID of 5 causes it to have the same privileges as the **uucp** login ID. Thus, for security, remote systems should not be allowed to log in as the BNU administrator.

Security and the Systems and remote.unknown files

On most BNU systems, only remote systems listed in the `/etc/uucp/Systems` file or one of its substitutes (specified in the `Sysfiles` file) can log in to the local system.

The `/usr/sbin/uucp/remote.unknown` script is executed whenever an unknown system attempts to call the local system. This script refuses to let the unknown system log in and makes an entry in the `/var/spool/uucp/.Admin/Foreign` file recording the time of the login attempt.

With root authority, or as a BNU administrator, you can modify the `remote.unknown` shell procedure to log more information about the remote system or to store the information in a different file. For example, you can modify the shell procedure to send mail to the BNU administrator whenever an unknown system tries to log in.

By taking away execute permissions on the `remote.unknown` shell procedure, you enable unknown machines to log in. In this case, you should add a MACHINE=OTHER entry to the `/etc/uucp/Permissions` file to establish permissions for the unknown machines.

Your system can contact only remote systems listed in the `Systems` file. This prevents users on your system from contacting unknown systems.

Security and the Permissions file

Consider the following security issues when using the `Permissions` file.

The `/etc/uucp/Permissions` file determines:

- Remote login user names for logging in to the local system
- Approved commands and privileges for remote systems logging in to the local system.

The `/etc/uucp/Permissions` file contains two types of entries:

Item	Description
LOGNAME	Defines login names and the privileges associated with them. LOGNAME entries take effect when a remote system calls the local system and attempts to log in.
MACHINE	Defines machine names and the privileges associated with them. MACHINE entries take effect when the remote system attempts to carry out commands on the local system.

Options in the `Permissions` file enable you to establish various levels of security for each remote system. For example, if many remote systems share one login ID on the local system, use the VALIDATE option to require each remote system to use a unique login ID. The SENDFILES, REQUEST, and CALLBACK options specify which system has control, keeping the local system in control of transactions if necessary.

The READ, WRITE, NOREAD, and NOWRITE options define access to specific directories on the local system. These options also control where on your system remote users can place data. The COMMANDS option limits the number of commands users on remote systems can execute on the local system. The COMMANDS=ALL option allows total privileges to systems closely associated with your system.

Attention: The COMMANDS=ALL option can seriously jeopardize the security of your system.

Communication between local and remote systems

In order to communicate between a remote system and a local system, the remote system must have a hardwire or modem link to the local system, a UNIX-based operating system installed, and either BNU or another version of UNIX-to-UNIX Copy Program (UUCP) running.

> **Note:** You can use BNU to communicate to a non-UNIX system, but such connections might require additional hardware or software.

BNU has two commands that enable you to communicate with remote systems. The **cu** command connects systems over either hardwire or telephone lines. The **ct** command connects systems over telephone lines only, using a modem.

Use the **cu** command to establish communication between networks when you know either the phone number or the name of the target system. To use the **ct** command, you *must* have the phone number of the target system.

> **Note:** A third command, **tip**, functions very much like the **cu** command. However, the **tip** command is a component of the Berkeley Software Distribution (BSD) version of the UUCP program. Its installation with BNU requires special configuration.

Communication with another system by hardwire or modem

Use the **cu** command from your local system to accomplish these communication tasks:

- Establish a connection to a specified remote system
- Log in to the remote system
- Perform tasks on the remote system
- Switch back and forth, working concurrently on both systems

If the remote system is running under the same operating system, you can issue regular commands from your local system. For example, you can issue commands to change directories, list directory contents, view files, or send files to the print queue on the remote system. To issue commands for use on your local system, or to initiate remote command and file exchanges, use special **cu** local commands, prefaced with a tilde (~).

Communication with another system by modem

Issue the **ct** command to communicate by modem with another system.

Enter the **ct** command, followed by a phone number, to call the remote modem. When the connection is made, the remote login prompt is displayed on your screen.

The **ct** command can be useful under certain conditions. For details on using the BNU **ct** command, see:

- "Dialing a number until a connection is made"
- "Dialing multiple numbers until a connection is made"

Dialing a number until a connection is made

This procedure describes how to use the **ct** command to continue dialing a remote modem number until a connection is made or until a specified time has passed.

The system to be called must be running Basic Networking Utilities (BNU) or some version of the UNIX-to-UNIX Copy Program (UUCP).

At the command line on the local system, type:

```
ct -w3 5550990
```

This dials the remote modem at phone numbers `555-0990`. The **-w3** flag and number instructs the **ct** command to dial the remote modem at one minute intervals until a connection is made or until three minutes have passed.

> **Note:** Type the phone number of the remote modem on the **ct** command line, either before or after the flag.

Dialing multiple numbers until a connection is made

This procedure describes how to use the **ct** command to continue dialing multiple remote modem numbers until a connection is made or until a specified time has passed.

The system to be called must be running Basic Networking Utilities (BNU) or some version of the UNIX-to-UNIX Copy Program (UUCP).

At the command line on the local system, type:

```
ct -w6 5550990 5550991 5550992 5550993
```

This dials the remote modems at phone numbers `555-0990`, `555-0991`, `555-0992`, and `555-0993`. The **-w6** flag and number instructs the **ct** command to dial the remote modems at one minute intervals until a connection is made or until six minutes have passed.

> **Note:** Type the phone numbers of the remote modems on the **ct** command line, either before or after the flag.

File exchanges between local and remote systems

The transfer of files among systems is the most common application of the Basic Networking Utilities (BNU). BNU uses four commands, **uucp**, **uusend**, **uuto**, and **uupick**, to exchange files between local and remote systems.

The **uucp** command is the primary BNU-data transfer utility. The **uusend** command is the Berkeley Software Distribution (BSD) transfer command incorporated into BNU. The **uuto** and **uupick** commands are specialized send and receive commands working with the **uucp** command.

The BNU commands, **uuencode** and **uudecode**, assist file transferring. These commands encode and decode binary files transmitted through the BNU mail facility.

Sending and receiving files

The commands used for sending and receiving files through a BNU connection include the **uucp** and the **uusend** commands.

Use the **uucp** command and options to exchange files within your local system, between your local and a remote system, and between remote systems. The **uucp** options can, for example, create directories to hold files on the receiving end or mail messages on the success or failure of file transfers.

Use the **uusend** command to send files to a remote system that is not directly linked to the sending system but is accessible through a chain of BNU connections. Though equipped with fewer options than the **uucp** command, **uusend** is included among the BNU utilities to satisfy the preferences of BSD UNIX-to-UNIX Copy Program (UUCP) users.

Sending files to a specific user

In order to send files to a specific user, the sending and receiving systems must be running either Basic Networking Utilities (BNU) or some version of the UNIX-to-UNIX Copy Program (UUCP).

Use the **uuto** command to send file from one system to another. It is part of the **uucp** command and simplifies the file exchange process for senders and receivers. The **uuto** command sends files to a specific user and deposits them directly into the user's personal directory under that system's BNU public directory. It notifies the recipient that a file has arrived. The recipient uses the **uupick** command to handle the new file.

Sending a file with the uuto command:

When you use the **uuto** command to send a file, include the file to be sent, the remote system destination, and the user at the destination.

For example:

```
uuto /home/bin/file1 distant!joe
```

This sends `file1` from the local `/home/bin` directory to user `joe` on the remote system `distant`.

The **uuto** command runs under the **uucp** command. The file is transferred to the remote system, in `/var/spool/uucppublic`. The file is deposited in the `/var/spool/uucppublic/receive/`*user*`/System` directory on the remote system. If the target directory does not exist, it is created during the file exchange.

The BNU **rmail** command notifies the receiver that a file has arrived.

> **Note:** To send a file to a user on a *local* system, enter the **uuto** command, and include the file to be sent, the local system destination, and the user at the local destination. For example:
>
> ```
> uuto /home/bin/file2 near!nick
> ```
>
> This sends `file2` from the local `/home/bin` directory to user `nick` on the local system `near`.

Receiving files

In order to receive and handle files, the sending and receiving systems must be running either Basic Networking Utilities (BNU) or some version of the UNIX-to-UNIX Copy Program (UUCP).

Use the **uupick** command to receive and manipulate files sent with the **uuto** command. It has file-handling options that allow the recipient to find sent files, move files to a specified directory, execute commands, or delete files.

Receiving a file with the uupick command:

Use the **uupick** command to receive a file.

For example:

```
uupick
```

The **uupick** command searches the public directory for files that include the remote user ID in the path names. The **uupick** command then displays on the remote screen a message similar to:

```
from system base: file file1?
```

The **?** (question mark) on the second line of the notification display prompts the receiver to use any **uupick** options for handling files in BNU's public directory.

For a list of all available options, type an asterisk (*****) on the line below the question mark (**?**) prompt. The display, save, and quit options are:

Item	Description
p	Displays the contents of the file.
m *[Directory]*	Saves the file to the directory specified by the *[Directory]* variable. If no destination is given with the **m** option, the file is moved to the current working directory.
q	Quits (exits) from the **uupick** file-handling process.

Encoding and decoding files for transfer

Use the **uuencode** and **uudecode** commands to prepare files for transmission by modem.

The commands work in tandem. The **uuencode** command transforms binary files into ASCII files. These files can be sent by the mail facility to a remote system.

With the **uudecode** command, the receiving user converts ASCII-encoded files back to binary format.

Command and file exchange status reports

You can view file exchange status reports using the **uusnap**, **uuq**, and **uustat** commands.

Displaying the status of systems connected by BNU

The **uusnap** command displays a table of information about all the systems connected by BNU.

The table shows a line for each system, reporting the names and the numbers of command files, data files, and remote command executions holding in the systems' queues. The last item on each line is a status message. This message indicates either a successful BNU connection or an explanation of why BNU did not establish a link.

Refer to the **uusnap** command.

BNU job queue display

The **uuq** command lists any entries in the BNU job queue.

The format of the list is similar to the format displayed by the **ls** command. The display for each entry includes the job number, followed on the same line by a summary, including the system name, the number of jobs for the system, and the total number of bytes to send. Users with root authority can use a **uuq** command to identify specific queued jobs by their job numbers.

Refer to the **uuq** command in *Commands Reference, Volume 5*.

Status of BNU operations

The **uustat** command provides the status of a particular command or file exchange in the BNU system.

Entered without flag options, the **uustat** command displays a single line for each job requested by the current user, including:

- Job ID number
- Date and time
- Status (send or receive)
- System name
- User ID of the person who issued the command
- Size and name of the job file

Equipped with several flags, the **uustat** command can report on all jobs, by all users, in the queue, or on jobs requested by other systems on the network.

The **uustat** command gives users limited control of jobs queued to run on a remote computer. You can examine the status of BNU connections to other systems and track file and command exchanges. Then, for example, you can cancel copy requests started by the **uucp** command.

Refer to the **uustat** command.

Command exchanges between local and remote systems

The Basic Networking Utilities (BNU) enable users to exchange commands between local and remote systems.

The **uux** command runs commands on a remote system. The **uupoll** command controls the timing for command execution.

Command execution requests on remote system

Use the **uux** command to request execution of a command on a remote system.

The **uux** command does not execute the commands on the remote system. Instead, it prepares the necessary control and data files in `/var/spool/uucp`. The **uucico** daemon is invoked to do the transfer. After the transfer is complete, the remote system's **uucico** creates an execute file in its spool directory.

When the two **uucico** daemons agree to hang up, the **uuxt** daemon scans the spool directory for outstanding execution requests, checks permissions, and checks to see if additional information is needed. It then forks a command to do what was requested.

> **Note:** You can use the **uux** command on any system configured to run a specified command. However, policies at some sites might restrict the use of certain commands for security reasons. Some sites, for example, might only permit execution of the **mail** command.

After the files are received on the remote system, the **uuxqt** daemon runs the specified command on that system. The **uuxqt** daemon periodically scans the remote system's public spool directory for files received in **uux** transmissions. The **uuxqt** daemon checks that data to be accessed by the sent files is present on the remote system. It also verifies that the sending system has permission to access the data. The **uuxqt** daemon then either executes the command or notifies the sending system that the command did not run.

Monitoring a BNU remote connection

Use the following procedure for monitoring a BNU remote connection.

- The BNU program must be installed on your system.
- A link (hardwired, modem, or TCP/IP) must be set up between your system and the remote system.

- The BNU configuration files, including the `Systems` file, `Permissions` file, `Devices` file, and `Dialers` file (and `Sysfiles` file, if applicable), must be set up for communications between your system and the remote system.

  Note: You must have root user authority to modify the BNU configuration files.

The **Uutry** command can help you monitor the **uucico** daemon process if users at your site report file-transfer problems.

1. Issue the **uustat** command to determine the status of all the transfer jobs in the current queue as follows:

   ```
   uustat -q
   ```

   The system displays a status report like the following:

   ```
   venus 3C (2) 05/09-11:02 CAN'T ACCESS DEVICE
   hera 1C 05/09-11:12 SUCCESSFUL
   merlin 2C 5/09-10:54 NO DEVICES AVAILABLE
   ```

   This report indicates that three command (`C.*`) files intended for remote system `venus` have been in the queue for two days. There could be several reasons for this delay. For example, perhaps system `venus` has been shut down for maintenance or the modem has been turned off.
2. Before you begin more extensive troubleshooting activities, issue the **Uutry** command as follows to determine whether your local system can contact system `venus` now:

   ```
   /usr/sbin/uucp/Uutry -r venus
   ```

   This command starts the **uucico** daemon with a moderate amount of debugging and the instruction to override the default retry time. The **Uutry** command directs the debugging output to a temporary file, `/tmp/venus`.
3. If your local system succeeds in establishing a connection to system `venus`, the debugging output contains a good deal of information. However, the final line in this script, which follows, is the most important:

   ```
   Conversation Complete: Status SUCCEEDED
   ```

   If the connection is successful, assume that the temporary file-transfer problems are now resolved. Issue the **uustat** command again to make certain that the files in the spooling directory have been transferred successfully to the remote system. If they have not, use the steps in "Monitoring a BNU file transfer" on page 435 to check for file-transfer problems between your system and the remote system.
4. If your local system cannot contact the remote system, the debugging output generated by the **Uutry** command contains the following type of information (the exact form of the output might vary):

   ```
   mchFind called (venus)
   conn (venus)
   getto ret -1
   Call Failed: CAN'T ACCESS DEVICE
   exit code 101
   Conversation Complete: Status FAILED
   ```

   First, check the physical connections between the local and remote systems. Make sure that the remote computer is turned on and all cables are properly connected, that the ports are enabled or disabled (as appropriate) on both systems, and that the modems (if applicable) are working.

   If the physical connections are correct and secure, then verify all the relevant configuration files on both the local and remote systems, including the following:

   - Make certain that the entries in the `Devices`, `Systems`, and `Permissions` files (and `Sysfiles` file, if applicable) in the **/etc/uucp** directory are correct on both systems.

- If you are using a modem, make sure that the `/etc/uucp/Dialers` file (or an alternate file specified in **/etc/uucp/Sysfiles**) contains the correct entry. If you are using dial-code abbreviations, be sure the abbreviations are defined in the `/etc/uucp/Dialcodes` file.
- If you are using a TCP/IP connection, make sure that the **uucpd** daemon can be run on the remote system and that the configuration files contain the correct TCP entries.

5. After you have checked the physical connections and configuration files, issue the **Uutry** command again. If the debugging output still reports that the connection failed, you might need to confer with a member of your systems support team. Save the debugging output produced by the **Uutry** command. This might prove helpful in diagnosing the problem.

Transferring a file to a remote system for printing

Use the **uux** command to transfer a file to a remote system for printing.

To transfer a file to a remote system for printing, the following prerequisites must be met:

- A Basic Networking Utilities (BNU) connection must be established to the remote system
- You must have permission to execute operations on the remote system

On the command line of the local system, type:

```
uux remote!/usr/bin/lpr local!filename
```

This prints the local file *filename* on the remote system.

Monitoring a BNU file transfer:

Use this procedure to monitor a file transfer to a remote system.

- The BNU program must be installed on and configured for your system.
- Establish a connection to a remote system using the steps given in "Monitoring a BNU remote connection" on page 433.

Monitoring a file transfer is useful when file transfers to the remote system in question are failing for unknown reasons. The debugging information produced by the **uucico** daemon (called by the **Uutry** command) can help you find out what is working incorrectly.

The **Uutry** command enables you to monitor file transfers, as follows:

1. Prepare a file for transfer using the **uucp** command with the **-r** flag by entering:

   ```
   uucp -r test1 venus!~/test2
   ```

   The **-r** flag instructs the UUCP program to create and queue all necessary transfer files but *not* to start the **uucico** daemon.

2. Issue the **Uutry** command with the **-r** flag to start the **uucico** daemon with debugging turned on by entering:

   ```
   /usr/sbin/uucp/Uutry -r venus
   ```

   This instructs the **uucico** daemon to contact remote system `venus` overriding the default retry time. The daemon contacts system `venus`, logs in, and transfers the file, while the **Uutry** command produces debugging output that enables you to monitor the **uucico** process. Press the Interrupt key sequence to stop the debugging output and return to the command prompt.

   The **Uutry** command also stores the debugging output in the `/tmp/SystemName` file. If you break out of the debugging output before the connection is complete, you can page through the output file to see the outcome of the connection.

Spooled job transmissions:

Use the **uupoll** command to start the transmission of jobs stored in the local system's public spooling directory.

The **uupoll** command creates a null job in the public directory for the remote system and starts the **uucico** daemon. This forces the **uucico** daemon to immediately contact the remote system and transfer any queued jobs.

Identifying compatible systems

Use the **uuname** command to display a list of all systems accessible to the local system.

For example, when you type:

```
uuname
```

at the command line, the system displays a list such as:

```
arthur
hera
merlin
zeus
```

This information is used to determine the name of an accessible system before copying a file to it. The **uuname** command is also used to establish the identity of the local system. The **uuname** command acquires its information by reading the `/etc/uucp/systems` file.

Communication with connected UNIX systems using the tip command

Use the **tip** command to contact any connected system running the UNIX operating system.

The **tip** command is installed with the Basic Networking Utilities (BNU) and can use the same asynchronous connections used by BNU.

The **tip** command uses variables and escape signals, as well as flags, to control its operations. The flags can be entered at the command line. The escape signals can be used over a connection with a remote system to start and stop file transfers, change the direction of a file transfer, and exit to a subshell.

tip command variables:

The **tip** command variables define settings such as the end-of-line character, the break signal, and the mode of file transfers.

Variable settings can be initialized at run time using a `.tiprc` file. Variable settings can also be changed during execution using the `~s` escape signal. Some variables, such as the end-of-line character, can be set for an individual system in the system entry of the `remote` file.

The **tip** command reads three files, the `phones` file, `remote` file, and `.tiprc` file, to determine initial settings for its variables. The `.tiprc` file must always be in the user home directory. The names and locations of the `remote` and `phones` files can vary. The names of the `remote` file and the `phones` file can be determined by environment variables:

Item	Description
PHONES	Specifies the name of the user phone file. The file can have any valid file name and must be set up in the format of the file /usr/lib/phones-file. The default file is etc/phones. If a file is specified with the PHONES variable, it is used in place of (not in addition to) the /etc/phones file.
REMOTE	Specifies the name of the user remote system definition file. The file can have any valid file name and must be set up in the format of the /usr/lib/remote-file file. The default file is /etc/remote. If a file is specified with the **REMOTE** variable, it is used in place of (not in addition to) the /etc/remote file.

To use an environment variable, set it before starting the **tip** command. As an alternative, the names of the phones and remote files can be determined using the **tip** command phones variable and remote variable, respectively, in the .tiprc file.

Note: The **tip** command reads only the *last* remote or phones file specified. Thus, if you specify a remote or phones file with a variable, the new file is used in place of (not in addition to) any previous files you specified.

The **tip** command uses variable settings in the following order:

1. The command checks the settings of the **PHONES** and **REMOTE** environment variables for the files to use as the phones and remote files.
2. The command reads the .tiprc file and sets all variables accordingly. If the phones or remote variable is set in the .tiprc file, this setting overrides the environment variable setting.
3. When a connection to a remote system is initiated, the command reads the remote file entry for that system. The settings in the remote file entry override settings made in the .tiprc file.
4. If the - BaudRate flag is used with the **tip** command, the specified rate overrides all previous baud rate settings.
5. A setting made with the ~s escape signal overrides all previous settings of a variable.

   Note: Any **tip** user can create a .tiprc file and use this file to specify initial settings for **tip** variables. The .tiprc file must be placed in the user $HOME directory.

tip command configuration files:

Before the **tip** command can connect to a remote system, the /etc/remote and /etc/phones files must be established.

Item	Description
/etc/remote	Defines attributes of remote systems such as the port and type of device to use to reach the system, as well as the signals to use to indicate the beginnings and endings of transmissions.
/etc/phones	Lists telephone numbers used to contact remote systems over a modem line.

Sample remote and phones files are delivered with the bos.net.uucp package. The sample remote file is named /usr/lib/remote-file. The sample phones file is named /usr/lib/phones-file. Copy /usr/lib/remote-file to /etc/remote and modify /etc/remote. To establish one of these files, copy a sample file to the correct name and modify it to suit the needs of your site.

A **tip** user can also create customized remote and phones files. An individual remote file must be in the format of the /usr/lib/remote-file file and specified with the remote variable or the REMOTE environment variable. An individual phones file must be in the format of the /usr/lib/phones-file file and specified with the phones variable or the PHONES environment variable. If an individual phones or remote file is specified with one of the variables, that file is read in place of (not in addition to) the /etc/phones or /etc/remote file.

Users of **tip** can use combinations of individual phones and remote files. For example, a user could use the default remote file, /etc/remote, but use an individual phones file named with the phones variable.

Canceling remote jobs

Use the **uustat** command to cancel a BNU process issued to a remote system.

To cancel a remote job, the following prerequisites must be met:

- A Basic Networking Utilities (BNU) connection must be established with the target remote system
- A remote job must have been issued from the local system

1. Determine the job ID number of the process, listed in the remote queue. On the command line of the local system, type:

   ```
   uustat -a
   ```

   The **-a** option displays all jobs in the remote system's holding queue and the job requests of any other BNU users on the system.

   BNU responds with a message similar to:

   ```
   heraC3113   11/06-17:47 S hera   you  289 D.venus471afd8
   merlinC3119 11/06-17:49 S merlin jane 338 D.venus471bc0a
   ```

2. Then type:

   ```
   uustat -k heraC3113
   ```

   The **-k** option cancels the `heraC3113` job request.

BNU troubleshooting

BNU error messages can be linked to a specific phase in the conversation flow. Use the "BNU conversation flow diagram" and the following error descriptions to help diagnose your BNU problems.

Some of the following messages might not be sent from BNU, but are included in case another UUCP version is in use.

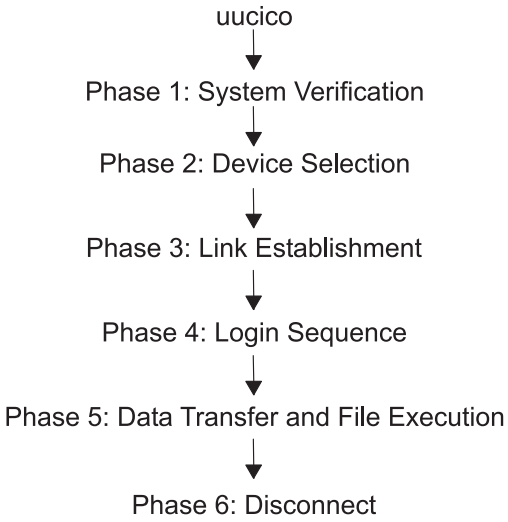


Figure 26. BNU conversion flow diagram

This illustration shows the flow and different phases of BNU conversion. From uucico at the top, data is passed to Phase 1-System Verification, then Phase 2-Device Selection, and Phase 3-Link Establishment, then Phase 4-Login Sequence, next Phase 5-Data Transfer and File Execution and last, Phase 6-Disconnect.

BNU PHASE 1 status messages

There are five BNU PHASE 1 status messages. The following table decribes them.

Item	Description
Assert Error	The local system unit is having problems. Check the error report for possible causes by issuing the command `errpt -a | pg`.
System not in Systems	If you supply a remote system name that is not found in the Systems files, this status message is created, BNU will terminate. Use the **uuname** command to check the system name again.
Wrong time to call	The Systems file has restrictions on times to allow outgoing calls. BNU will keep trying until the time is right. Check the Systems file.
Callback required	The network has restricted usage either for security or economic reasons, and access is denied at this time.
Cannot call No Call	These errors mean BNU recently tried to call the remote system and failed. It will not immediately try again. They can also be caused by an old system status file being retained thus keeping the **uucico** daemon from trying again.

BNU PHASE 2 status messages

There are four BNU PHASE 2 status messages. The following table decribes them.

Item	Description
Dialer Script Failed	Your Dialers file script did not complete successfully.
No Device Available Can't Access Device	The modem or the outgoing phone line from your system is busy. Check for an error in the device entry of the Systems file. Also, check the Devices and Dialers files to be sure logical devices have physical devices associated with them. The file /etc/uucp/Sysfiles might be specifying an alternate Systems, Devices, or Dialers file that is not correctly configured. Is the device in use by some other program? Check the /var/locks directory for lock on port. If a lock file exists (for example, LCK..TTY0), check to see if the process identified by the number in the lock file is still active. If not, you can remove it (for example, rm /var/locks/LCK..TTY0). Also check the permissions on the port.
Dial Failed Failed (call to system)	These errors appear when your system dials another successfully but the other system does not answer. It might also indicate a problem in the Devices files. Enter the command uucico -r1 -x6 -s *SystemName*. It could be that BNU is expecting some string that it is not receiving. Make the connection by hand to find out what needs to be incorporated into the Systems files entry to satisfy the request. Please keep "timing" in mind; perhaps some delays in the modem dial string are needed. This could also mean that the port is busy, you dialed an incorrect number, or BNU lost ownership of the port.
OK Auto Dial	These are informative messages only and do not indicate an error.

BNU PHASE 3 status messages

There are five BNU PHASE 3 status messages. The following table decribes them.

Item	Description
Handshake Failed (LCK)	The device is being used by someone else; the process could not create the LCK file. Sometimes LCK files must be manually removed by the administrator. After a number of retries, see your system administrator. See if another process has control of the port (for example, another instance of the **uucico** daemon).
Login Failed	The login failed due to a bad connection or possibly a slow machine.
Timeout	The remote system did not respond within a set period of time. This could also indicate a problem with the chat script.
Succeeded (Call to System)	The call was completed.
BNU (continued)	These are informative messages only and do not indicate an error.

BNU PHASE 4 status messages

There are six BNU PHASE 4 status messages. The following table decribes them.

Item	Description
`Startup Failed Remote reject after login`	After login, the **uucico** daemon is started on the remote system. If there is a problem initiating a conversation between the two systems, these messages are created. You might have also logged into the incorrect BNU account or the initial handshake failed.
`Wrong machine name`	A machine was called incorrectly or the machine name was changed.
`Bad login/machine combination`	The login to the remote system failed. The problem could be an incorrect phone number, an incorrect login or password, or an error in the chat script.
`Remote has a LCK file for me`	Both systems were simultaneously trying to call each other. The local request will fail temporarily.
`OK Talking`	These are informative messages only and do not indicate an error.
`LOGIN: PASSWORD:`	If the login or password prompt is in all capital letters, the modem might be in echo mode (E1 on Hayes compatibles). This causes the modem to echo back, or send, a RING to your system when an incoming call is received. The **getty** command receives the string and accordingly changes the `login:` or `password:` into all caps. Change the echo mode on the modem to off (use `ATE0` for Hayes compatibles). **Note:** Keep in mind that once this change is made, you should use `ATE1` in the chat script of your `Dialers` files, or you will not get the expected `OK` back from the modem. If the remote port is set for `delay` or `getty -r` and the chat script expects key input, then the ports set for `delay` are expecting one or more carriage returns before proceeding with the login. Try beginning the chat script on the dialing system with the following: `"" \r\d\r\d\r\d\r in:--in: ...` Interpreted, this chat script reads as follows: expect nothing, send return, delay, return, delay, return, delay, return.

BNU PHASE 5 status messages

There are five BNU PHASE 5 status messages. The following table decribes them.

Item	Description
`Alarm`	The **uucico** daemon is having trouble with the connection. Either the connection is bad or "xon/xoff" is set to yes on the modem.
`Remote access to path/file denied copy (failed)`	These messages indicate a permission problem; check file and path permissions.
`Bad read`	The remote system ran out of space, most likely in the spool area, or the **uucico** daemon could not read or write to device.
`Conversation failed`	The modem carrier detect was lost. Possibly the modem was turned off, the cable is loose or disconnected, or the remote system crashed or is shut down. Telephone disconnection can also cause this error.
`Requested Copy (succeeded)`	These are informative messages only and do not indicate an error.

BNU PHASE 6 status messages

There are two BNU PHASE 6 status messages. The following table decribes them.

Item	Description
`OK (Conversation Complete)`	The remote system can deny the hangup request and reverse the roles (meaning the remote system has work for the local system to do). After the two **uucico** daemons agree that no more work exists, they hang up.
`Conversation succeeded`	This is an informative message only and does not indicate an error.

Debugging BNU login failures using the uucico daemon

Use the **uucico** daemon to debut BNU login failures.

- BNU must be installed on your system.
- A link (hardwired, modem, or TCP/IP) must be set up between your system and the remote system.
- The BNU configuration files, including the `Sysfiles` file (if applicable), the `Systems` file, `Permissions` file, `Devices` file, and `Dialers` file, must be set up for communications between your system and the remote system.

Note: You must have root user authority to modify the BNU configuration files.

- You must have root user authority to invoke the **uucico** daemon in debugging mode.

1. To produce debugging information about a local-to-remote system connection that is not working, start the **uucico** daemon with the **-x** flag as follows:

   ```
   /usr/sbin/uucp/uucico -r 1 -s venus -x 9
   ```

   where `-r 1` specifies the master, or caller mode; `-s venus`, the name of the remote system to which you are trying to connect; and `-x 9`, the debug level that produces the most detailed debugging information.

2. If the expect-send sequence entry in a `Systems` file in the format of `/etc/uucp/Systems` is:

   ```
   venus Any venus 1200 - "" \n in:--in: uucp1 word:
    mirror
   ```

   the**uucico** daemon connects the local system to the remote system `venus`. The debugging output is similar to:

   ```
   expect: ""
   got it
   sendthem (^J^M)
   expect (in:)^
   M^Jlogin:got it
   sendthem (uucp1^M)
   expect (word:)^
   M^JPassword:got it
   sendthem (mirror^M)
   imsg >^M^J^PShere^@Login Successful: System=venus
   ```

   where:

Item	Description
`expect: ""`	Specifies that the local system will not wait for any information from the remote system.
`got it`	Acknowledges that the message has been received.
`sendthem (^J^M)`	Specifies that the local system will send the remote system a carriage return and a new line.
`expect (in:)`	Specifies that the local system expects to receive the remote system login prompt, which ends in the `in:` character string.
`^M^Jlogin:got it`	Confirms that the local system received the remote login prompt.
`sendthem (uucp1^M)`	Specifies that the local system will send the `uucp1` login ID to the remote system.
`expect (word:)`	Specifies that the local system expects to receive the remote system password prompt, which ends in the `word:` character string.
`^M^JPassword:got it`	Confirms the local system received the remote password prompt.
`sendthem (mirror^M)`	Specifies that the local system will send the password for the `uucp1` login ID to the remote system.
`imsg >^M^J^PShere^@Login Successful: System=venus`	Confirms the local system is successfully logged in to remote system `venus`.

Note:

1. The expect-send debugging output produced by the **uucico** command can come either from information in the `/etc/uucp/Dialers` file or from information in the `/etc/uucp/Systems` file. Information about communication with the modem comes from the `Dialers` file, while information about communication with the remote system comes from the `Systems` file. (Note that `/etc/uucp/Systems` and `/etc/uucp/Dialers` are default BNU configuration files. Other files can be specified in `/etc/uucp/Sysfiles` to serve the same role.)
2. To set up a connection with a remote system, you must be familiar with the login sequence of that system.

SNMP for network management

The Network Management facility provides comprehensive management of system networks through the use of **Simple Network Management Protocol (SNMP)**, enabling network hosts to exchange management information.

SNMP is an internetworking protocol designed for use with **TCP/IP**-based internets.

When the AIX operating system is installed, the **SNMPv3** non-encrypted version is installed by default and is started at system boot time. If you have your own communities, traps, and SMUX entries configured in your `/etc/snmpd.conf` file, you will need to migrate those manually to the `/etc/snmpdv3.conf` file. For information on migrating the communities, see "Migrating from SNMPv1 to SNMPv3" on page 451.

You might also want to consult the information in SNMP Overview for Programmers in *Communications Programming Concepts*.

SNMP network management is based on the familiar client/server model that is widely used in **TCP/IP**-based network applications. Each host that is to be managed runs a process called an *agent*. The agent is a server process that maintains the Management Information Base (MIB) database for the host. Hosts that are involved in network management decision-making can run a process called a manager. A *manager* is a client application that generates requests for MIB information and processes responses. In addition, a manager can send requests to agent servers to modify MIB information.

SNMP in AIX provides support for the following RFCs:

Item	Description
RFC 1155	Structure and Identification of Management Information for **TCP/IP**-based Internets
RFC 1157	A **Simple Network Management Protocol (SNMP)**
RFC 1213	Management Information Base for Network Management of **TCP/IP**-based internets: MIB-II
RFC 1227	**Simple Network Management Protocol (SNMP)** single multiplexer (SMUX) protocol and Management Information Base (MIB)
RFC 1229	Extensions to the generic interface Management Information Base (MIB)
RFC 1231	IEEE 802.5 token-ring Management Information Base (MIB)
RFC 1398	Definitions of Managed Objects for the Ethernet-like Interface Types
RFC 1512	FDDI Management Information Base
RFC 1514	Host Resources MIB
RFC 1592	**Simple Network Management Protocol**-Distributed Program Interface Version 2
RFC 1905	Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)
RFC 1907	Management Information Base for Version 2 of the Simple Network Management Protocol (SNMPv2)
RFC 2572	Message Processing and Dispatching for the **Simple Network Management Protocol (SNMP)**
RFC 2573	**SNMP** Applications
RFC 2574	User-based Security Model (USM) for version 3 of the **Simple Network Management Protocol (SNMPv3)**
RFC 2575	View-based Access Control Model (VACM) for the **Simple Network Management Protocol (SNMP)**

SNMPv3

In earlier versions of the AIX operating system, **SNMPv1** was the only available version of **SNMP** for AIX. **SNMPv3**, provided in the AIX operating system, delivers a powerful and flexible framework for message security and access control.

The information in this section applies to **SNMPv3** only.

Message security involves providing the following:

- Data integrity checking to ensure that the data was not altered in transit.
- Data origin verification to ensure that the request or response originates from the source that it claims to have come from.

- Message timeliness checking and, optionally, data confidentiality to protect against eavesdropping.

The **SNMPv3** architecture introduces the User-based Security Model (USM) for message security and the View-based Access Control Model (VACM) for access control. The architecture supports the concurrent use of different security, access control, and message processing models. For example, community-based security can be used concurrently with USM, if desired.

USM uses the concept of a user for which security parameters (levels of security, authentication and privacy protocols, and keys) are configured at both the agent and the manager. Messages sent using USM are better protected than messages sent with community-based security, where passwords are sent in the clear and displayed in traces. With USM, messages exchanged between the manager and the agent have data integrity checking and data origin authentication. Message delays and message replays (beyond what happens normally due to a connectionless transport protocol) are prevented by the use of time indicators and request IDs. Data confidentiality, or encryption, is also available, where permitted, as a separately installable product. The **SNMP** encrypted version can be found on the AIX Expansion Pack.

The use of VACM involves defining collections of data (called views), groups of users of the data, and access statements that define which views a particular group of users can use for reading, writing, or receipt in a trap.

SNMPv3 also introduces the ability to dynamically configure the **SNMP** agent using **SNMP** SET commands against the MIB objects that represent the agent's configuration. This dynamic configuration support enables addition, deletion, and modification of configuration entries either locally or remotely.

SNMPv3 access policies and security parameters are specified in the `/etc/snmpdv3.conf` file on the **SNMP** agent and `/etc/clsnmp.conf` file on the **SNMP** manager. For a scenario on how to configure these files, see "Creating users in SNMPv3" on page 454. You can also refer to the `/etc/snmpdv3.conf` and `/etc/clsnmp.conf` file formats in *Files Reference*.

SNMPv3 architecture

There are four main parts to the **SNMPv3** architecture.

The interaction between these systems to provide the necessary data requested is depicted in the following figure:

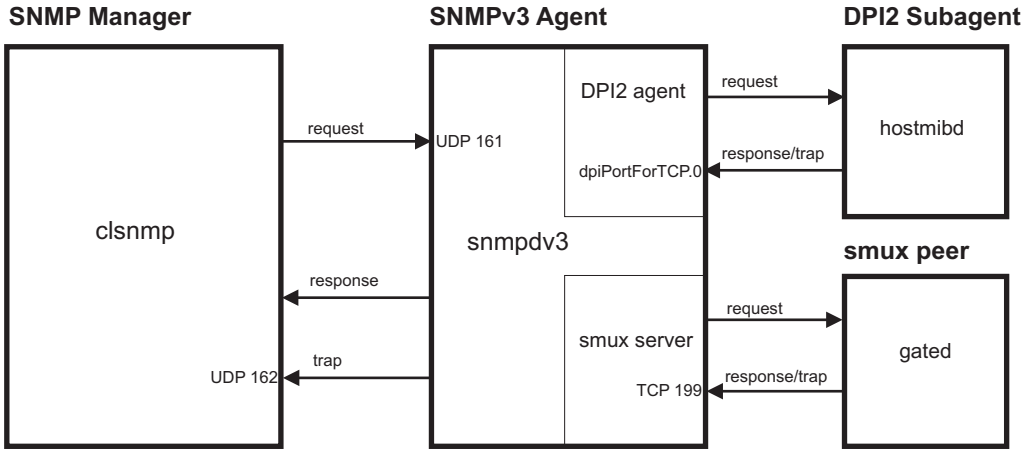


Figure 27. The primary parts of the SNMPv3 architecture

This illustration shows and example of the **SNMPv3** architecture. The DPI2 subagent, smux peer, **SNMP** manager, and **SNMP** agent are shown. In addition, how they communicate with each other is shown.

SNMP agent:

The **SNMP** agent receives requests from, and makes responses to, the **SNMP** manager.

In addition, the **SNMP** agent communicates with all DPI2 subagents and SMUX peers on the system. The **SNMP** agent manages some MIB variables, and all DPI2 subagents and SMUX peers register their MIB variables with the **SNMP** agent.

When **clsnmp** (the **SNMP** manager) issues a request, it is sent to UDP 161 on the **SNMP** agent. If the request is an **SNMPv1** or **SNMPv2c** request, the **SNMP** agent will verify the community name and process the request. If the request is an **SNMPv3** request, the **SNMP** agent will attempt to authenticate the user requesting the data and ensure that the user has the access permissions required to fulfill the request by using the authentication keys, and, if the encrypted version is running, privacy keys. If the **SNMP** agent cannot authenticate the user, or if the user does not have the correct access permissions to fulfill the request, the **SNMP** agent will not honor the request. For information on creating users in **SNMPv3**, see "Creating users in SNMPv3" on page 454.

If the user is authenticated and has the correct access permissions, the **SNMP** agent will fulfill the request. The **SNMP** agent will locate the MIB variables being requested. If the **SNMP** agent itself is managing the requested MIB variables, it will process the request and send a response back to the **SNMP** manager. If a DPI2 subagent or SMUX peer is managing the requested MIB variables, the **SNMP** agent will forward the request to the DPI2 subagent or SMUX peer on which the MIB variables are managed, allow it to process the request, and will then respond to the **SNMP** manager.

DPI2 subagents:

A DPI2 subagent, such as **hostmibd**, communicates with the DPI2 agent, which, in **SNMPv3**, is part of the SNMP agent.

The DPI2 subagent sends responses and traps to the DPI2 agent through dpiPortForTCP.0. Because this is not a well-known port, the DPI2 subagent must first issue a request for the port number for dpiPortForTCP.0. This request is issued to UDP 161 on the **SNMP** agent, after which the **SNMP** agent responds to the DPI2 subagent with the port number for dpiPortForTCP.0. After the port number is received, the DPI2 subagent establishes a connection with the DPI2 agent using the port number given. The DPI2 subagent then registers its MIB subtrees with the DPI2 agent.

Note: To enable the **SNMP** Agent to listen on a port other than UDP 161, you must set the **SNMP_PORT** environment. There are two ways to set this variable:

- **Method 1**: Stop the DPI2 subagent and type the following commands:
  - `SNMP_PORT=<port_number> /usr/sbin/aixmibd -d 128`
  - `SNMP_PORT=<port_number> /usr/sbin/hostmibd -d 128`
  - `SNMP_PORT=<port_number> /usr/sbin/snmpmibd -d 128`

  where *port_number* is the number of the port that you want to use.

  After the commands have completed executing, start the DPI2 subagent.
- **Method 2**: Include the **SNMP_PORT** variable in the `/etc/environment` file and assign the new port value to it. Allow the **aixmibd**, **hostmibd**, **snmpmibd**, and **snmpd** daemons to run from `/etc/rc.tcpip` as is. In this method, you do not have to run the **aixmibd**, **hostmibd**, and **snmpmibd** commands from the command line.

After the connection is established and the MIB subtrees have been registered, the DPI2 subagent is ready to respond to requests received from the DPI2 agent. When a request is received, the DPI2 subagent processes the request and responds with the necessary information.

The DPI2 subagent is also ready to send traps, if necessary. When a trap is sent, the **SNMP** agent will check its `/etc/snmpdv3.conf` file to determine the IP address or addresses to which the trap must be forwarded to, and it will send the trap to those addresses.

SMUX peers:

A SNMP Multiplexing (SMUX) peer, such as **gated**, when started, will establish the connection to **TCP** 199 and will initialize the SMUX association.

Following the initialization, the SMUX peer will register the MIB subtrees it is going to manage.

After the registration, the SMUX peer is ready to accept any incoming request from the SMUX server and send responses back. When the SMUX peer receives a request, it will process the request and send a response back the SMUX server.

The SMUX peer can also send a trap to the SMUX server. If a trap is sent, the **SNMP** agent will check the `/etc/snmpdv3.conf` file to determine the IP address or addresses to which the trap must be forwarded, and it will send the trap to those addresses.

SNMP manager:

The **SNMP** manager runs **clsnmp**, which is compatible with **SNMPv1**, **SNMPv2c**, and **SNMPv3**.

Use the **clsnmp** command to issue a request, such as a `get`, `get-next`, `get-bulk`, or `set` request. The request is sent to UDP 161 on the **SNMP** agent, after which it waits for the response from the **SNMP** agent.

Note: To allow the **SNMP** Manager to use a port other than UDP 161, you need to declare the port number you want to use and the IP address in the **targetAgent** field of the `/etc/clsnmp.conf` file. For information on the `/etc/clsnmp.conf` file, see clsnmp.conf File in *Files Reference*.

It also can listen to **SNMP** traps on UDP 162. The **SNMP** manager will receive traps if its IP address is so specified in the `/etc/snmpdv3.conf` file on the **SNMP** agent.

MIB variables:

Information on MIB variables can be found in the following locations.

For information on MIB variables, see Management Information Base, Terminology Related to Management Information Base Variables, Working with Management Information Base Variables, and Management Information Base Database in *Communications Programming Concepts*.

If you want to configure your own DPI2 subagent or smux peer, see the `/usr/samples/snmpd/smux` and `/usr/samples/snmpd/dpi2` directories.

SNMPv3 authentication keys

Authentication is generally required for **SNMPv3** requests to be processed (unless the security level requested is `noAuth`).

When authenticating a request, the **SNMP** agent verifies that the authentication key sent in an **SNMPv3** request can be used to create a message digest that matches the message digest created from the authentication key defined by the user.

When a request is issued from the **SNMP** manager, the **clsnmp** command uses the authentication key found on an entry in the `/etc/clsnmp.conf` file on the **SNMP** manager. It needs to correlate with the

authentication key specified on a USM_USER entry for that user in the **SNMP** agent's /etc/snmpdv3.conf file. Authentication keys are generated using the **pwtokey** command.

The authentication key is generated from two pieces of information:

- The specified password
- The identification of the **SNMP** agent at which the key will be used. If the agent is an IBM agent, and its engineID was generated using the vendor-specific engineID formula, the agent may be identified by IP address or host name. Otherwise, the engineID must be provided as the agent identification.

A key that incorporates the identification of the agent at which it will be used is called a localized key. It can be used only at that agent. A key that does not incorporate the engineID of the agent at which it will be used is called non-localized.

Keys stored in the **clsnmp** command's configuration file, /etc/clsnmp.conf, are expected to be non-localized keys. Keys stored in the **SNMP** agent's configuration file, /etc/snmpdv3.conf, can be either localized or non-localized, though using localized keys is considered more secure.

As an alternative to storing authentication keys in the client configuration file, the **clsnmp** command allows user passwords to be stored. If the **clsnmp** command is configured with a password, the code generates an authentication key (and a privacy key if requested, and if the encrypted version is installed) for the user. These keys must produce the same authentication values as the keys configured for the USM_USER in the agent's /etc/snmpdv3.conf file or configured dynamically with the **SNMP** SET commands. However, the use of passwords in the client configuration file is considered less secure that the use of keys in the configuration file.

SNMPv3 privacy keys

Encryption is available as a separate product on the AIX Expansion Pack where export laws allow. Keys used for encryption are generated using the same algorithms as those used for authentication.

However, key lengths may differ. For example, an HMAC-SHA authentication key is 20 bytes long, but a localized encryption key used with HMAC-SHA is only 16 bytes long.

The encrypted version is automatically activated after installation. To switch back to the non-encrypted version, use the **snmpv3_ssw** command.

SNMPv3 generating keys

AIX uses the **pwtokey** command to generate authentication and, when applicable, privacy keys.

The **pwtokey** command enables the conversion of passwords into localized and non-localized authentication and privacy keys. The **pwtokey** procedure takes a password and an identifier as the agent and generates authentication and privacy keys. Because the procedure used by the **pwtokey** command is the same algorithm used by the **clsnmp** command, the person configuring the **SNMP** agent can generate appropriate authentication (and privacy) keys to put into the /etc/clsnmp.conf file on the **SNMP** manager for a user, given a particular password and the IP address at which the target will run on.

After you have generated the authentication keys (and privacy keys if you are running the encrypted version), you will need to enter those keys in the /etc/snmpdv3.conf file on the **SNMP** agent and in the /etc/clsnmp.conf file on the **SNMP** manager.

In **SNMPv3**, there are nine possible user configurations. Each possible configuration, along with an example of each, is given below. These particular keys were generated using defaultpassword for the password and 9.3.149.49 as the IP address. The following command was used:

```
pwtokey -u all -p all defaultpassword 9.3.149.49
```

The following authentication and privacy keys were generated:

```
Display of 16 byte HMAC-MD5 authKey:
  18a2c7b78f3df552367383eef9db2e9f

Display of 16 byte HMAC-MD5 localized authKey:
  a59fa9783c04bcbe00359fb1e181a4b4

Display of 16 byte HMAC-MD5 privKey:
  18a2c7b78f3df552367383eef9db2e9f

Display of 16 byte HMAC-MD5 localized privKey:
  a59fa9783c04bcbe00359fb1e181a4b4

Display of 20 byte HMAC-SHA authKey:
  754ebf6ab740556be9f0930b2a2256ca40e76ef9

Display of 20 byte HMAC-SHA localized authKey:
  cd988a098b4b627a0e8adc24b8f8cd02550463e3

Display of 20 byte HMAC-SHA privKey:
  754ebf6ab740556be9f0930b2a2256ca40e76ef9

Display of 16 byte HMAC-SHA localized privKey:
  cd988a098b4b627a0e8adc24b8f8cd02
```

These entries would appear in the `/etc/snmpdv3.conf` file. The following nine configurations are possible:

- Localized authentication and privacy keys using the HMAC-MD5 protocol:

  ```
  USM_USER user1 - HMAC-MD5 a59fa9783c04bcbe00359fb1e181a4b4 DES a59fa9783c04bcbe00359fb1e181a4b4 L - -
  ```

- Non-localized authentication and privacy keys using the HMAC-MD5 protocol:

  ```
  USM_USER user2 - HMAC-MD5 18a2c7b78f3df552367383eef9db2e9f DES 18a2c7b78f3df552367383eef9db2e9f N  - -
  ```

- Localized authentication key using the HMAC-MD5 protocol:

  ```
  USM_USER user3 - HMAC-MD5 a59fa9783c04bcbe00359fb1e181a4b4 - - L -
  ```

- Non-localized authentication key using the HMAC-MD5 protocol:

  ```
  USM_USER user4 - HMAC-MD5 18a2c7b78f3df552367383eef9db2e9f - - N -
  ```

- Localized authentication and privacy keys using the HMAC-SHA protocol:

  ```
  USM_USER user5 - HMAC-SHA cd988a098b4b627a0e8adc24b8f8cd02550463e3 DES
  cd988a098b4b627a0e8adc24b8f8cd02 L -
  ```

- Non-localized authentication and privacy keys using the HMAC-SHA protocol:

  ```
  USM_USER user6 - HMAC-SHA 754ebf6ab740556be9f0930b2a2256ca40e76ef9 DES
  754ebf6ab740556be9f0930b2a2256ca40e76ef9 N -
  ```

- Localized authentication key using the HMAC-SHA protocol:

  ```
  USM_USER user7 - HMAC-SHA cd988a098b4b627a0e8adc24b8f8cd02550463e3 - - L -
  ```

- Non-localized authentication key using the HMAC-SHA protocol:

  ```
  USM_USER user8 - HMAC-SHA 754ebf6ab740556be9f0930b2a2256ca40e76ef9 - - N -
  ```

- Neither authentication nor privacy keys used (**SNMPv1**)

  ```
  USM_USER user9 - none - none - - -
  ```

Configuring users in **SNMPv3** requires configuration of both the `/etc/snmpdv3.conf` file and the `/etc/clsnmp.conf` file. For a scenario on generating user keys and editing the necessary configuration files, see "Creating users in SNMPv3" on page 454. In addition, see the **pwtokey** command in *Commands Reference, Volume 4* and the **clsnmp** command in *Commands Reference, Volume 1*, and the file formats for the `/etc/clsnmp.conf` file and `/etc/snmpdv3.conf` file in *Files Reference*. You can also refer to the sample `snmpdv3.conf` configuration file and `clsnmp.conf` configuration file located in the `/usr/samples/snmpdv3` directory.

SNMPv3 updating keys

SNMPv3 offers the capability of updating user keys based on new passwords dynamically.

This is done by using the **pwchange** command to generate new user keys based on an updated password, using the **clsnmp** command to dynamically update the user key in the /etc/snmpdv3.conf file, and editing the /etc/clsnmp.conf file with the new keys. During this process, the new password is never communicated between machines.

For step-by-step instructions on updating user keys, see "Dynamically updating authentication and privacy keys in SNMPv3." In addition, refer to the **pwchange** coomand in *Commands Reference, Volume 4* and the **clsnmp** command in *Commands Reference, Volume 1*, and the /etc/clsnmp.conf file format and /etc/snmpdv3.conf file format in *Files Reference*.

Dynamically updating authentication and privacy keys in SNMPv3

This scenario shows how to dynamically update the authentication keys for a user in **SNMPv3**.

In this scenario, user u4 will update the authentication keys for user u8. Both users u4 and u8 have already had authentication keys created based on password defaultpassword and IP address 9.3.149.49, and everything is working.

During this scenario, new keys will be created for user u8 and the /etc/snmpdv3.conf file will be dynamically updated. The authentication key for user u8 in the manager side's /etc/clsnmp.conf file will then need to be manually edited to reflect the new keys.

Make a backup of the /etc/snmpdv3.conf file on the **SNMP** agent and a backup of the /etc/clsnmp.conf file on the **SNMP** manager before you start this procedure.

Below is the /etc/snmpdv3.conf file that will be dynamically updated:

```
USM_USER u4 - HMAC-MD5 18a2c7b78f3df552367383eef9db2e9f - - N -
USM_USER u8 - HMAC-SHA 754ebf6ab740556be9f0930b2a2256ca40e76ef9 - - N -

VACM_GROUP group1 SNMPv1  public  -
VACM_GROUP group2 USM u4 -
VACM_GROUP group2 USM u8 -

VACM_VIEW defaultView        internet                   - included -

VACM_ACCESS  group1 - - noAuthNoPriv SNMPv1  defaultView - defaultView -
VACM_ACCESS  group2 - - noAuthNoPriv USM defaultView defaultView defaultView -
VACM_ACCESS  group2 - - AuthNoPriv USM defaultView defaultView defaultView -
VACM_ACCESS  group2 - - AuthPriv USM defaultView defaultView defaultView -

NOTIFY notify1 traptag trap -

TARGET_ADDRESS Target1 UDP 127.0.0.1       traptag trapparms1 - - -
TARGET_ADDRESS Target2 UDP 9.3.149.49       traptag trapparms2 - - -
TARGET_ADDRESS Target3 UDP 9.3.149.49       traptag trapparms3 - - -
TARGET_ADDRESS Target4 UDP 9.3.149.49       traptag trapparms4 - - -

TARGET_PARAMETERS trapparms1 SNMPv1  SNMPv1  public  noAuthNoPriv -
TARGET_PARAMETERS trapparms3 SNMPv2c  SNMPv2c  publicv2c  noAuthNoPriv -
TARGET_PARAMETERS trapparms4 SNMPv3  USM     u4 AuthNoPriv -
```

Below is the /etc/clsnmp.conf file that will be updated for user u8:

```
testu4 9.3.149.49 snmpv3 u4 - - AuthNoPriv HMAC-MD5 18a2c7b78f3df552367383eef9db2e9f - -
testu8 9.3.149.49 snmpv3 u8 - - AuthNoPriv HMAC-SHA 754ebf6ab740556be9f0930b2a2256ca40e76ef9 - -
```

Things to consider

- The information in this how-to scenario was tested using specific versions of AIX. The results you obtain might vary significantly depending on your version and level of AIX.

Update your password and authentication keys

The community names in the /etc/snmpd.conf file become part of the VACM_GROUP entries in the /etc/snmpdv3.conf file. Each community must be placed in a group. You will then give the groups the view and access permissions needed.

1. On the **SNMP** manager side, run the **pwchange** command. In this scenario, we ran the following command:

   ```
   pwchange -u auth -p HMAC-SHA defaultpassword newpassword 9.3.149.49
   ```

   This command will generate a new authentication key.

   - -u auth specifies that only an authentication key will be created. If you are updating privacy keys as well, use -u all.
   - -p HMAC-SHA specifies the protocol that will be used to create the authentication key. If you are updating privacy keys as well, us -p all.
   - *defaultpassword* is the password used to create the latest authentication key (for example, if bluepen would have been used to create the latest authentication key, bluepen would be used here as well)
   - *newpassword* is the new password that will be used to generate the authentication key. Keep this password for future reference
   - 9.3.149.49 is the IP address where the **SNMP** agent is running.

   This command produced the following output:

   ```
   Dump of 40 byte HMAC-SHA authKey keyChange value:
     8173701d7c00913af002a3379d4b150a
     f9566f56a4dbde21dd778bb166a86249
     4aa3a477e3b96e7d
   ```

   You will use this authentication key in the next step.

   Note: Keep the new passwords you use in a safe place. You will need to use them again when making changes in the future.

2. On the **SNMP** manager, user u4 will change the authentication key for user u8 by entering the following command:

   ```
   clsnmp -h testu4 set usmUserAuthKeyChange.12.0.0.0.2.0.0.0.0.9.3.149.49.2.117.56
    \'8173701d7c00913af002a3379d4b150af9566f56a4dbde21dd778bb166a862494aa3a477e3b96e7d\'h
   ```

   - testu4 is used because it is mapped to user u4 in the /etc/clsnmp.conf file.
   - The instance ID of *usmUserAuthKeyChange* includes, in decimal values, the engine ID of the SNMP agent where the update is taking place and username whose authentication key is being updated. The engine ID can be found in the /etc/snmpd.boots file (the /etc/snmpd.boots file contains two strings of numbers. The engine ID is the first string. Ignore the second string of numbers).

     The engine ID will need to be converted from hexadecimal values to decimal values in order to be used here. Each two numbers in the hexadecimal engine ID convert to one decimal value. For example, engine ID 0000000020000000009039531 would be read as 00 00 00 02 00 00 00 00 09 03 95 31. Each of those numbers must be converted to decimal values, resulting in, 0.0.0.2.0.0.0.0.9.3.149.49 (For a conversion table, see ASCII, decimal, hexadecimal, octal, and binary conversion table.). The first number in the string is the number of bytes in the decimal string. In this case, it is 12, resulting in 12.0.0.0.2.0.0.0.0.9.3.149.49.

     The following number is the number of bytes in the username, followed by the decimal values for the username itself. In this case, the username is u8. When converted to decimal values, u8 becomes 117.56. Because the username is 2 bytes long, the value representing the username becomes 2.117.56. Add that to the end of the decimal engine ID (For a conversion table, see ASCII, decimal, hexadecimal, octal, and binary conversion table.).

     In this case, the result is 12.0.0.0.2.0.0.0.0.9.3.149.49.2.117.56.

- The next value in the command is the new authentication key generated using the **pwchange** command in the previous step.

  Note: If the user also has privacy keys configured, this procedure must be repeated to update the privacy keys. When updating the privacy keys, use the `usmUserPrivKeyChange` value instead of the `usmUserAuthKeyChange` value.

  Using `usmUserOwnAuthKeyChange` instead of `usmUserAuthKeyChange` will allow a user to change his or her own authentication key. For example, user `u4` could change its own authentication key using `usmUserOwnAuthKeyChange`.

The output of the command follows:

```
1.3.6.1.6.3.15.1.2.2.1.6.12.0.0.0.2.0.0.0.0.9.3.149.49.2.117.56 = '8173701d7c00913af002a3379
d4b150af9566f56a4dbde21dd778bb166a862494aa3a477e3b96e7d'h
```

After this command is completed, the `/etc/snmpdv3.conf` file will be automatically updated after five minutes on the **SNMP** agent side. You can also stop and start the **SNMP** daemon to update the file. The following entry for user `u8` will be dynamically updated in the `/etc/snmpdv3.conf` file:

```
USM_USER u8 000000020000000009039531 HMAC-SHA 4be657b3ae92beee322ee5eaeef665b338caf2d9
None - L nonVolatile
```

3. On the **SNMP** manager side, run the **pwtokey** command to generate the new authentication key based on the new password to place in the `/etc/clsnmp.conf` file. In this scenario, we ran the following command:

   ```
   pwtokey -u auth -p HMAC-SHA  newpassword 9.3.149.49
   ```

   - `-u auth` specifies that only an authentication key will be created. If you are updating privacy keys as well, use `-u all`.
   - `-p HMAC-SHA` specifies the protocol that will be used in creating the authentication key. If you are updating privacy keys as well, use `-p all`.
   - The password used (in this case *newpassword*) must be the same as the password used when generating new authentication keys with the **pwchange** command.
   - The IP address used (in this case `9.3.149.49`) must be the IP address where the agent is running.

   The result gives the localized and non-localized authentication keys:

   ```
   Display of 20 byte HMAC-SHA authKey:
     79ce23370c820332a7f2c7840c3439d12826c10d

   Display of 20 byte HMAC-SHA localized authKey:
     b07086b278163a4b873aace53a1a9ca250913f91
   ```

4. Open the `/etc/clsnmp.conf` file with your favorite text editor and place the non-localized authentication key in the line for the user whose keys are being updated. In this scenario, the entry is as follows:

   ```
   testu8 9.3.149.49 snmpv3 u8 - - AuthNoPriv HMAC-SHA 79ce23370c820332a7f2c7840c3439d12826c10d - -
   ```

   Save and close the file.

5. Test the updated configuration by running the following command:

   ```
   clsnmp -v -h testu8 walk mib
   ```

   where *mib* is a MIB variable to which user `u8` has read access. In this case, user `u8` has access to `internet`.

SNMPv3 requests

The **clsnmp** command is used to send **SNMP** requests to **SNMP** agents on local or remote hosts.

The requests can be **SNMPv1**, **SNMPv2c**, or **SNMPv3** requests. In order to process requests, the `/etc/clsnmp.conf` file must be configured.

The **clsnmp** command can issue get, getnext, getbulk, set, walk, and findname requests. Each of these requests is briefly described below:

get allows the user to gather data from one MIB variable

getnext
gives the next MIB variable in the MIB subtree

getbulk
gives all MIB variables from multiple MIB subtrees

set allows the user to set a MIB variable

walk gives all MIB variables of one subtree

findname
maps the OID to the variable name

trap allows **clsnmp** to listen to traps on port 162

For detailed information on issuing **clsnmp** requests, see the **clsnmp** command in *Commands Reference, Volume 1*.

Migrating from SNMPv1 to SNMPv3

This scenario shows a typical migration from **SNMPv1** to **SNMPv3**.

In the AIX operating system, the default **SNMP** agent running at system boot time is the non-encrypted version of **SNMPv3**. **SNMPv3** uses the `/etc/snmpdv3.conf` file as its configuration file. Any parameters that you had configured in the `/etc/snmpd.conf` file, which is used by **SNMPv1** in earlier versions of the AIX operating system, will need to be manually migrated to the `/etc/snmpdv3.conf` file.

In this scenario, the communities and traps that were configured in the `/etc/snmpd.conf` file will be migrated to the `/etc/snmpdv3.conf` file. By the end of the scenario, **SNMPv3** will provide identical functionality that **SNMPv1** offered. If you did not configure any of your own **SNMPv1** communities or traps, there is no need for you to complete this procedure.

This file does not contain any information about features available in **SNMPv3**. For information about creating users using **SNMPv3** features not available in **SNMPv1**, see "Creating users in SNMPv3" on page 454.

The following file is the example `/etc/snmpd.conf` file that is going to be migrated. The following communities are configured: `daniel`, `vasu`, and `david`. These communities must be migrated manually.

```
logging         file=/usr/tmp/snmpd.log         enabled
logging         size=0                          level=0

community       daniel       0.0.0.0     0.0.0.0         readWrite  1.17.35
community       vasu         9.3.149.49  255.255.255.255 readOnly   10.3.5
community       david        9.53.150.67 255.255.255.255 readWrite  1.17.35

view 1.17.35    udp  icmp  snmp 1.3.6.1.2.1.25
view 10.3.5     system interfaces tcp icmp

trap            daniel          9.3.149.49      1.17.35   fe
trap            vasu            9.3.149.49      10.3.5   fe
trap            david           9.53.150.67     1.17.35   fe

smux            1.3.6.1.4.1.2.3.1.2.3.1.1         sampled_password  # sampled
```

To complete the steps in this scenario, refer to your `/etc/snmpd.conf` file. Have a copy of that file ready when you start this procedure.

Things to consider

- The information in this how-to scenario was tested using specific versions of AIX. The results you obtain might vary significantly depending on your version and level of AIX.

Step 1. Migrate the community information

The community names in the /etc/snmpd.conf file become part of the VACM_GROUP entries in the /etc/snmpdv3.conf file. Each community must be placed in a group. You will then give the groups the view and access permissions needed.

1. With root authority, open the /etc/snmpdv3.conf file with your favorite text editor. Locate the VACM_GROUP entries in the file.
2. Create a VACM_GROUP entry for each community that you want to migrate. If multiple communities are going to share the same view and access permissions, you need to create only one group for them. The community names in the /etc/snmpd.conf file become the *securityName* values for the VACM_GROUP entries. In this scenario, the following entries were added for vasu, daniel, and david:

   ```
   #--------------------------------------------------------------------
   # VACM_GROUP entries
   #    Defines a security group (made up of users or communities)
   #    for the View-based Access Control Model (VACM).
   # Format is:
   #  groupName securityModel securityName storageType
   VACM_GROUP group2 SNMPv1  vasu -
   VACM_GROUP group3 SNMPv1  daniel -
   VACM_GROUP group3 SNMPv1  david -
   #--------------------------------------------------------------------
   ```

   - *groupName* can be any value you choose, except group1.
   - *securityModel* remains SNMPv1 because we are migrating the SNMPv1 communities.
   - In this scenario, daniel and david share the same view and access permissions in the /etc/snmpd.conf file. Therefore, they are both members of group3 in the /etc/snmpdv3.conf file. Community vasu is placed in a different group because its view and access permissions are different than those for david and daniel.

   The communities are now placed in groups.

Step 2. Migrate the view information

The view information in the /etc/snmpd.conf file will become COMMUNITY, VACM_VIEW, and VACM_ACCESS entries in the /etc/snmpdv3.conf file. These entries will determine the view and access permissions for each group.

1. Create COMMUNITY entries for daniel, vasu, and david, maintaining the same IP addresses for netAddr and netMask as were specified in the /etc/snmpd.conf file.

   ```
   #--------------------------------------------------------------------------
   # COMMUNITY
   #    Defines a community for community-based security.
   # Format is:
   #  communityName securityName securityLevel netAddr netMask storageType
   COMMUNITY public    public     noAuthNoPriv 0.0.0.0     0.0.0.0         -
   COMMUNITY daniel    daniel     noAuthNoPriv 0.0.0.0     0.0.0.0         -
   COMMUNITY vasu      vasu       noAuthNoPriv 9.3.149.49  255.255.255.255 -
   COMMUNITY david     david      noAuthNoPriv 9.53.150.67 255.255.255.255 -
   #--------------------------------------------------------------------------
   ```

2. Create a VACM_VIEW entry for every MIB object or variable that each group has access to. According to the /etc/snmpd.conf file, daniel and david have access to udp, icmp, snmp, and 1.3.6.1.2.1.25 (host subtree as defined in RFC 1514), and vasu has access to system, interfaces, tcp, and icmp. These view entries are migrated to the /etc/snmpdv3.conf file as follows:

   ```
   #--------------------------------------------------------------------
   # VACM_VIEW entries
   #    Defines a particular set of MIB data, called a view, for the
   ```

```
#    View-based Access Control Model.
# Format is:
#  viewName viewSubtree viewMask viewType storageType

VACM_VIEW group2View        system                    - included -
VACM_VIEW group2View        interfaces                - included -
VACM_VIEW group2View        tcp                       - included -
VACM_VIEW group2View        icmp                      - included -

VACM_VIEW group3View        udp                       - included -
VACM_VIEW group3View        icmp                      - included -
VACM_VIEW group3View        snmp                      - included -
VACM_VIEW group3View        1.3.6.1.2.1.25            - included -
#---------------------------------------------------------------------
```

3. Define access permissions to the MIB variables defined in the VACM_VIEW entries by adding VACM_ACCESS entries. In the /etc/snmpd.conf file, daniel and david both have readWrite permission to their MIB variables, whereas vasu has readOnly.

   Define these permissions by adding VACM_ACCESS entries. In this scenario, we gave group2 (vasu) the group2View for readView, but gave it - for writeView because vasu had readOnly in the /etc/snmpd.conf file. We gave group3 (daniel and david) the group3View for both readView and writeView because those groups had readWrite access in /etc/snmpd.conf. See the following example.

```
#-------------------------------------------------------------------------------------------------------------
# VACM_ACCESS entries
#    Identifies the access permitted to different security groups
#    for the View-based Access Control Model.
# Format is:
# groupName contextPrefix contextMatch securityLevel securityModel readView writeView notifyView storageType
VACM_ACCESS  group1 - - noAuthNoPriv SNMPv1  defaultView - defaultView -
VACM_ACCESS  group2 - - noAuthNoPriv SNMPv1  group2View - group2View -
VACM_ACCESS  group3 - - noAuthNoPriv SNMPv1  group3View group3View group3View -
#-------------------------------------------------------------------------------------------------------------
```

Step 3. Migrate the trap information

The trap entries in the /etc/snmpd.conf file will become the NOTIFY, TARGET_ADDRESS, and TARGET_PARAMETERS entries in the /etc/snmpdv3.conf file. However, only the TARGET_ADDRESS and TARGET_PARAMETERS will need to be migrated.

1. The IP addresses listed in the trap entries in the /etc/snmpd.conf file become part of the TARGET_ADDRESS entries in the /etc/snmpdv3.conf file. This line specifies the host where the trap will be sent. You can define the targetParams entries. In this scenario, we use trapparms1, trapparms2, trapparms3, and trapparms4, which will be defined in the TARGET_PARAMETERS entries.

```
#--------------------------------------------------------------------------------
# TARGET_ADDRESS
#    Defines a management application's address and parameters
#    to be used in sending  notifications.
# Format is:
#  targetAddrName tDomain tAddress tagList targetParams timeout retryCount storageType
TARGET_ADDRESS Target1 UDP 127.0.0.1    traptag trapparms1 - - -
TARGET_ADDRESS Target2 UDP 9.3.149.49   traptag trapparms2 - - -
TARGET_ADDRESS Target3 UDP 9.3.149.49   traptag trapparms3 - - -
TARGET_ADDRESS Target4 UDP 9.53.150.67  traptag trapparms4 - - -
#--------------------------------------------------------------------------------
```

2. The community names specified in the trap entries in the /etc/snmpd.conf file become part of the TARGET_PARAMETERS entries in the /etc/snmpdv3.conf file. The community names must be mapped to a specific TARGET_ADDRESS entry using the targetParams values. For example, community daniel is mapped with trapparms2, which, under the TARGET_ADDRESS entry, maps to IP address 9.3.149.49. Community daniel and IP address 9.3.149.49 were originally a trap entry in the /etc/snmpd.conf file. See the following example:

```
#---------------------------------------------------------------------
# TARGET_PARAMETERS
#    Defines the message processing and security parameters
```

```
#    to be used in sending notifications to a particular management target.
# Format is:
#  paramsName mpModel securityModel securityName securityLevel storageType
TARGET_PARAMETERS trapparms1 SNMPv1  SNMPv1  public  noAuthNoPriv -
TARGET_PARAMETERS trapparms2 SNMPv1  SNMPv1  daniel  noAuthNoPriv -
TARGET_PARAMETERS trapparms3 SNMPv1  SNMPv1  vasu    noAuthNoPriv -
TARGET_PARAMETERS trapparms4 SNMPv1  SNMPv1  david   noAuthNoPriv -
#--------------------------------------------------------------------
```

3. The `trapmask` information in the `/etc/snmpd.conf` file does not migrate to the `/etc/snmpdv3.conf` file.

Step 4. Migrate the smux information

If you have smux information that you need to migrate, you can copy those lines directly into the new file. In this scenario, the `sampled` smux entry was configured in the `/etc/snmpd.conf` file. That line must be copied to the `/etc/snmpdv3.conf` file.

```
#-------------------------------------------------------------------------------------------
#       smux <client OIdentifier> <password> <address> <netmask>
smux            1.3.6.1.4.1.2.3.1.2.3.1.1         sampled_password  # sampled
#-------------------------------------------------------------------------------------------
```

Step 5. Stop and Start the snmpd daemon

After the migration of the `/etc/snmpd.conf` file to the `/etc/snmpdv3.conf` file is complete, stop and then start the **snmpd** daemon. You will need to stop and start the **snmpd** daemon each time you make changes to the `/etc/snmpdv3.conf` file.

1. Type the following command to stop the daemon:
   ```
   stopsrc -s snmpd
   ```
2. Type the following command to restart the daemon:
   ```
   startsrc -s snmpd
   ```

Note: Simply refreshing the **SNMPv3** agent will not work as it did in **SNMPv1**. If you make changes to the `/etc/snmpdv3.conf` file, you must stop and start the daemon as instructed above. The dynamic configuration function supported in **SNMPv3** will not allow you to refresh.

Creating users in SNMPv3

This scenario shows how to create a user in SNMPv3 by manually editing the `/etc/snmpdv3.conf` and `/etc/clsnmp.conf` files.

User `u1` will be created in this scenario. User `u1` will be given authorization keys, but will not be given privacy keys (which are available only if you have the `snmp.crypto` fileset installed). The HMAC-MD5 protocol will be used to create u1's authorization keys. After u1 is configured, it will be put into a group, after which that group will have its view and access permissions defined. Finally, trap entries for u1 will be created.

Each individual value used in the `/etc/snmpdv3.conf` and `/etc/clsnmp.conf` files must not exceed 32 bytes.

Things to Consider

- The information in this how-to scenario was tested using specific versions of AIX. The results you obtain might vary significantly depending on your version and level of AIX.

Step 1. Create the user

1. Decide which security protocols you want to use, either HMAC-MD5 or HMAC-SHA. In this scenario, HMAC-MD5 will be used.

2. Generate the authentication keys by using the pwtokey command. Your output may look different based on the authentication protocol you are using and if you are using privacy keys. These keys will be used in the /etc/snmpdv3.conf and /etc/clsnmp.conf files. The command used for user u1 follows:

   ```
   pwtokey -p HMAC-MD5 -u auth anypassword 9.3.230.119
   ```

   The IP address specified is the IP address where the agent is running. The password can by any password, but be sure to save it in a secure place for future use. The output should look similar to the following:

   ```
   Display of 16 byte HMAC-MD5 authKey:
     63960c12520dc8829d27f7fbaf5a0470

   Display of 16 byte HMAC-MD5 localized authKey:
     b3b6c6306d67e9c6f8e7e664a47ef9a0
   ```

3. With root authority, open the /etc/snmpdv3.conf file with your favorite text editor.
4. Create a user by adding a USM_USER entry following the format given in the file. The *authKey* value will be the localized authentication key that was generated using the **pwtokey** command. The entry for user u1 follows:

   ```
   #------------------------------------------------------------------------
   # USM_USER entries
   #    Defines a user for the User-based Security Model (USM).
   # Format is:
   #  userName engineID authProto authKey privProto privKey keyType storageType
   #
   USM_USER u1 - HMAC-MD5 b3b6c6306d67e9c6f8e7e664a47ef9a0 - - L -
   #------------------------------------------------------------------------
   ```

   - *userName* is the name of the user. In this case, it is u1.
   - *authProto* must be the protocol that you used when you created the keys. In this case, it is HMAC-MD5.
   - *authKey* is the localized authentication key that was created using the **pwtokey** command.
   - *privProto* and *privkey* are not specified because we are not using the privacy keys in this scenario.
   - *keyType* is L because we are using the localized authentication key.
5. Save and close the /etc/snmpdv3.conf file.
6. Open the /etc/clsnmp.conf file on the SNMP manager with your favorite text editor.
7. Add the new user according to the format given in the file. The entry for u1 follows:

   ```
   #-----------------------------------------------------------------------------------------------
   #
   # Format of entries:
   # winSnmpName targetAgent admin secName password context secLevel authProto authKey privProto privKey
   #
   user1  9.3.230.119  SNMPv3  u1  -  -  AuthNoPriv  HMAC-MD5  63960c12520dc8829d27f7fbaf5a0470  -  -
   #-----------------------------------------------------------------------------------------------
   ```

   - *winSnmpName* can be any value. This value will be used when making SNMP requests using the **clsnmp** command.
   - *targetAgent* is the IP address where the agent is running, which was also used in creating the authentication keys.
   - *admin* is set to SNMPv3 because we will be sending SNMPv3 requests.
   - *secName* is the name of the user that you are creating. In this case, it is u1.
   - *seclevel* is set to AuthNoPriv because it is being configured to use authentication but not privacy (as a result, there are no values for *privProto* and *privKey*).
   - *authproto* is set to the authentication protocol that was used in creating the authentication keys.
   - *authKey* is the non-localized key that was generated by the **pwtokey** command.
8. Save and close the /etc/clsnmp.conf file.

Step 2. Configure the group

The user must now be placed in a group. If you already have a group that is configured with all of the view and access permissions that you want to give this user, you can put this user in that group. If you want to give this user view and access permissions that no other groups have, or if you do not have any groups configured, create a group and add this user to it.

To add the user to a new group, create a new VACM_GROUP entry in the /etc/snmpdv3.conf file. The group entry for u1 follows:

```
#-------------------------------------------------------------
# VACM_GROUP entries
#    Defines a security group (made up of users or communities)
#    for the View-based Access Control Model (VACM).
# Format is:
#  groupName securityModel securityName storageType
VACM_GROUP group1 USM u1 -
#-------------------------------------------------------------
```

- *groupName* can be any name. It becomes that name of your group. In this case, it is group1.
- *securityModel* is set to USM, which takes advantage of the SNMPv3 security features.
- *securityName* is the name of the user. In this case, it is u1.

Step 3. Configure view and access permissions

The view and access permissions must be set for the new group that was just created. These permissions are set by adding VACM_VIEW and VACM_ACCESS entries to the /etc/snmpdv3.conf file.

1. Decide what view and access permissions you want the new group to have.
2. Add VACM_VIEW entries to the /etc/snmpdv3.conf file to define what MIB objects the group can access. In this scenario, group1 will have access to the interfaces, tcp, icmp, and system MIB subtrees. However, we will restrict group1's access to the sysObjectID MIB variable within the system MIB subtree.

   ```
   #-------------------------------------------------------------
   # VACM_VIEW entries
   #    Defines a particular set of MIB data, called a view, for the
   #    View-based Access Control Model.
   # Format is:
   #  viewName viewSubtree viewMask viewType storageType
   VACM_VIEW group1View  interfaces    - included -
   VACM_VIEW group1View  tcp        - included -
   VACM_VIEW group1View  icmp        - included -
   VACM_VIEW group1View  system     - included -
   VACM_VIEW group1View  sysObjectID    - excluded -
   #-------------------------------------------------------------
   ```

   - *viewName* is the name of the view. In this scenario, it is group1View.
   - *viewSubtree* is the MIB subtree that you want to give access to.
   - *viewType* determines whether the MIB subtrees defined are included in the view. In this case, all subtrees are included, but the MIB variable sysObjectID, which is part of the system subtree, is excluded.
3. Add a VACM_ACCESS entry to the /etc/snmpdv3.conf file to define the permissions that the group has to the MIB objects specified above. For group1, read only access is given.

   ```
   #-----------------------------------------------------------------------------------------------------
   # VACM_ACCESS entries
   #    Identifies the access permitted to different security groups
   #    for the View-based Access Control Model.
   # Format is:
   # groupName contextPrefix contextMatch securityLevel securityModel readView writeView notifyView storageType
   VACM_ACCESS  group1 - - AuthNoPriv USM group1View - group1View -
   #-----------------------------------------------------------------------------------------------------
   ```

   - *groupName* is the name of the group. In this case, it is group1.

- *securityLevel* is the level of security that is being used. In this scenario, authentication keys are used but not privacy keys. The value is therefore set to AuthNoPriv.
- *securityModel* is the security model that you are using (SNMPv1, SNMPv2c, or USM). In this scenario, it is set to USM to allow the SNMPv3 security features to be used.
- *readView* determines which VACM_VIEWs the group has read access to. In this scenario, group1View is given, which gives group1 read access to the group1View VACM_VIEW entries.
- *writeView* determines which VACM_VIEWs the group has write access to. In this scenario, no write access is given to group1.
- *notifyView* specifies the name of the view to be applied when a trap is performed under control of the entry in the access table.

Note: In some cases, multiple VACM_ACCESS entries for one group may be necessary. If users in the group have different authentication and privacy settings (noAuthNoPriv, AuthNoPriv, or AuthPriv) multiple VACM_ACCESS entries are required with the securityLevel parameter set accordingly.

Step 4. Configure trap entries for the user

Trap entries in SNMPv3 are created by adding NOTIFY, TARGET_ADDRESS and TARGET_PARAMETERS entries to the /etc/snmpdv3.conf file. The TARGET_ADDRESS entry will specify where you want the traps to be sent, and the TARGET_PARAMETERS entry will map the TARGET_ADDRESS information to group1.

The NOTIFY entry has been configured by default. Following is the default NOTIFY entry:

```
NOTIFY notify1 traptag trap -
```

In this scenario, we use the value that is specified in the default entry, traptag.

1. Add a TARGET_ADDRESS entry to specify where you want traps to be sent.

```
#-------------------------------------------------------------------------------
# TARGET_ADDRESS
#    Defines a management application's address and parameters
#    to be used in sending  notifications.
# Format is:
#  targetAddrName tDomain tAddress tagList targetParams timeout retryCount storageType
#-------------------------------------------------------------------------------
TARGET_ADDRESS Target1 UDP 9.3.207.107     traptag trapparms1 - - -
```

   - *targetAddrName* can be any name. In this scenario, we used Target1.
   - *tAddress* is the IP address where the traps for the group should be sent.
   - *tagList* is the name configured in the NOTIFY entry. In this scenario, it is traptag.
   - *targetParams* can be any value. We used is trapparms1, which will be used in the TARGET_PARAMETERS entry.

2. Add a TARGET_PARAMETERS entry.

```
#-------------------------------------------------------------------------
# TARGET_PARAMETERS
#    Defines the message processing and security parameters
#    to be used in sending notifications to a particular management target.
# Format is:
#  paramsName mpModel securityModel securityName securityLevel storageType
#-------------------------------------------------------------------------
TARGET_PARAMETERS trapparms1 SNMPv3  USM     u1        AuthNoPriv -
```

   - *paramsName* is the same as the targetParams value in the TARGET_ADDRESS entry, which, in this case, is trapparms1.
   - *mpModel* is the version of SNMP being used.
   - *securityModel* is the security model that you are using (SNMPv1, SNMPv3, or USM). In this scenario, it is set to USM to allow the SNMPv3 security features to be used.
   - *securityName* is the user name specified in the USM_USER entry, which, in this case, is u1.

- *securityLevel* is set to AuthNoPriv because we are using authentication keys but not privacy keys.

Step 5. Stop and start the snmpd daemon

After making the changes the /etc/snmpdv3.conf file, stop and the start the **snmpd** daemon.

1. Type the following command to stop the **snmpd** daemon:
   ```
   stopsrc -s snmpd
   ```
2. Type the following command to start the **snmpd** daemon:
   ```
   startsrc -s snmpd
   ```

The new settings will now take effect.

Note: Simply refreshing the SNMPv3 agent using refresh -s snmpd will not work as it did in SNMPv1. If you make changes to the /etc/snmpdv3.conf file, you must stop and start the daemon as instructed above. The dynamic configuration function supported in SNMPv3 will not allow you to refresh.

Step 6. Test your configuration

To verify that your configuration is correct, you can run the following command on the SNMP manager .

```
clsnmp -h user1 walk mib
```

where *mib* is a MIB subtree to which the user has access. In this scenario, it could be interfaces, tcp, icmp, or system. If the configuration is correct, you will see the information from the specified subtree.

If you did not get the correct output, review the steps in this document and verify that you have entered all information correctly.

SNMPv3 troubleshooting

These problems might be encountered using **SNMPv3**.

- While migrating, you need to migrate the community and SMUX entries defined in the /etc/snmpd.conf file to the /etc/snmpdv3.conf file. For information on migrating this information, see "Migrating from SNMPv1 to SNMPv3" on page 451.
- Requests are not generating any responses.

  The most likely cause of this problem is a configuration error in either the /etc/snmpdv3.conf file or the /etc/clsnmp.conf file, or both. Carefully review these files to ensure that all information is entered correctly. For information on editing these files when creating new users, see "Creating users in SNMPv3" on page 454.
- A new user was configured using both authentication and privacy keys, but an error message is returned when using this user.

  The most likely cause is that you are not running the **SNMPv3** encrypted version. Follow these steps to determine what version you are running:

  1. Run ps -e|grep snmpd.
     - If you receive no output, you probably need to start the **snmpd** daemon. Run startsrc -s snmpd.
     - If your output included snmpdv1, you are running **SNMPv1**. You will be able to make **SNMPv1** requests when running this version.
     - If your output included snmpdv3ne, you are running the **SNMPv3** non-encrypted version. After installing the AIX operating system, this version will be running by default. This version does not allow you to use privacy keys.
     - If your output included snmpdv3e, you are running the **SNMPv3** encrypted version, which is a separately-installable product. The **SNMPv3** encrypted version is available on the AIX Expansion Pack where allowed. The **SNMPv3** encrypted version allows the use of privacy keys.

2. Determine if the version that you are running is the intended version. If it is not, use the **snmpv3_ssw** command to change the version as follows:
   - `snmpv3_ssw -1` will switch to **SNMPv1**
   - `snmpv3_ssw -n` will switch to **SNMPv3** non-encrypted
   - `snmpv3_ssw -e` will switch to **SNMPv3** encrypted if it is installed

- After making changes to the `/etc/snmpdv3.conf` file and refreshing the daemon, my changes are not taking effect.

  After making changes to the `/etc/snmpdv3.conf` file, the **SNMP** daemon must be stopped and started. Refreshing the daemon will not work. Use the following procedure:

  1. Stop the **SNMP** daemon by running `stopsrc -s snmpd`.
  2. Start the **SNMP** daemon by running `startsrc -s snmpd`.

- The DPI2 subagent is started, but no MIB variables can be queried from it.

  The most likely cause is that the `public` community is not configured in the `/etc/snmpdv3.conf` file. By default, and DPI2 subagent shipped with AIX uses the community name `public` to connect itself to the **SNMP** agent. The `public` community is configured in the `/etc/snmpdv3.conf` file by default. If you have removed the `public` community from the `/etc/snmpd.conf` file, add the following lines to the file:

  ```
  VACM_GROUP group1 SNMPv1 public -
  VACM_VIEW defaultView 1.3.6.1.4.1.2.2.1.1.1.0 - included -
  VACM_ACCESS  group1 - - noAuthNoPriv SNMPv1  defaultView - defaultView -
  COMMUNITY public    public     noAuthNoPriv 0.0.0.0     0.0.0.0         -
  ```

  `1.3.6.1.4.1.2.2.1.1.1.0` is the OID for dpiPortForTCP.0.

- MIB variables managed by the SMUX peer that could be queried before migration can no longer be queried.

  Ensure that your SMUX entry is present in the `/etc/snmpdv3.conf` file and the `/etc/snmpd.peers` file. If you configure new SMUX peers, ensure that they are entered in both of these files as well.

- A personal set of MIB variables was implemented, but the variables cannot be included or excluded from viewing by other users.

  In the VACM_VIEW entry in the `/etc/snmpdv3.conf` file, you must specify the OID of the MIB variable rather than the MIB variable name.

- Receiving traps are not being received.

  Ensure that you have configured the trap entries correctly in the `/etc/snmpdv3.conf` file. In addition, if the trap is an **SNMPv3** trap, the `/etc/clsnmp.conf` file must also be configured. For instructions on configuring traps, see "Creating users in SNMPv3" on page 454.

  In addition, make sure that the machine specified to receive traps (in the `/etc/snmpdv3.conf` file) is listening for them. You can start this process by running `clsnmp trap` on the command line of the receiving machine.

- Why is the DPI2 server not running in the **SNMPv3** environment?

  In the **SNMPv3** architecture, the **SNMPv3** agent itself runs the DPI2 server. See "SNMPv3 architecture" on page 443 for more information.

SNMPv1

This information is specific to **SNMPv1**. When using **SNMPv1**, the **snmpd** agent uses a simple authentication scheme to determine which **Simple Network Management Protocol (SNMP)** manager stations can access its Management Information Base (MIB) variables.

This authentication scheme involves the specification of **SNMP** access policies for **SNMPv1**. An **SNMP** access policy is an administrative relationship involving an association among an **SNMP** community, an access mode, and an MIB view.

An *SNMP community* is a group of one or more hosts and a community name. A community name is a string of octets that an **SNMP** manager must embed in an **SNMP** request packet for authentication purposes.

The *access mode* specifies the access the hosts in the community are allowed with respect to retrieving and modifying the MIB variables from a specific **SNMP** agent. The access mode must be one of the following: *none*, *read-only*, *read-write*, or *write-only*.

A *MIB view* defines one or more MIB subtrees that a specific **SNMP** community can access. The MIB view can be the entire MIB tree or a limited subset of the entire MIB tree.

When the **SNMP** agent receives a request, the agent verifies the community name with the requesting host IP address to determine if the requesting host is a member of the **SNMP** community identified by the community name. If the requesting host is a member of the **SNMP** community, the **SNMP** agent then determines if the requesting host is allowed the specified access for the specified MIB variables as defined in the access policy associated with that community. If all criteria are met, the **SNMP** agent attempts to honor the request. Otherwise, the **SNMP** agent generates an *authenticationFailure* trap or returns the appropriate error message to the requesting host.

The **SNMPv1** access policies for the **snmpd** agent are user-configurable and are specified in the `/etc/snmpd.conf` file. To configure the **SNMP** access policies for the **snmpd** agent, see the `/etc/snmpd.conf` file in *Files Reference*.

SNMP daemon configuration

The **Simple Network Management Protocol (SNMP)** daemon is a background server process that can be run on any **Transmission Control Protocol/Internet Protocol (TCP/IP)** workstation host.

The daemon, acting as **SNMP** agent, receives, authenticates, and processes **SNMP** requests from manager applications. See Simple Network Management Protocol, How a Manager Functions, and How an Agent Functions in *Communications Programming Concepts* for more detailed information on agent and manager functions.

Note: The terms **SNMP** daemon, **SNMP** agent, and agent are used interchangeably.

The **snmpd** daemon requires the loopback **TCP/IP** interface to be active for minimal configuration. Enter the following command before starting **TCP/IP**:

```
ifconfig lo0 loopback up
```

The **SNMP** daemon will attempt to bind sockets to certain well-known **User Datagram Protocol (UDP)** and **Transmission Control Protocol (TCP)** ports, which must be defined in the `/etc/services` file as follows:

```
snmp            161/udp
snmp-trap       162/udp
smux            199/tcp
```

The `snmp` service must be assigned port `161`, as required by RFC 1157. The `/etc/services` file assigns ports 161, 162, and 199 to these services. If the `/etc/services` file is being serviced off another machine, these assigned ports must be made available in the served `/etc/services` file on the server before the **SNMP** daemon can run.

The **SNMP** daemon reads the configuration file on the running **SNMP** version on startup and when a **refresh** command (if the **snmpd** daemon is invoked under System Resource Controller control) or **kill -1** signal is issued.

/etc/snmpd.conf file:

The `/etc/snmpd.conf` configuration file specifies community names and associated access privileges and views, hosts for trap notification, logging attributes, **snmpd**-specific parameter configurations, and single multiplexer (SMUX) configurations for the **SNMP** daemon for **SNMPv1**.

See the /etc/snmpd.conf file in *Files Reference* for more information.

SNMP daemon processing

The **Simple Network Management Protocol (SNMP)** daemon processes **SNMP** requests from manager applications.

Read Simple Network Management Protocol (SNMP), How a Manager Functions, and How an Agent Functions in *Communications Programming Concepts* for more detailed information on agent and manager functions.

SNMP message processing and authentication:

All requests, traps, and responses are transmitted in the form of ASN.1-encoded messages.

A message, as defined by RFC 1157, has the following structure:

Version Community PDU

where *Version* is the SNMP version (currently version 1), *Community* is the community name, and *PDU* is the protocol data unit that contains the **SNMP** request, response, or trap data. A PDU is also encoded according to ASN.1 rules.

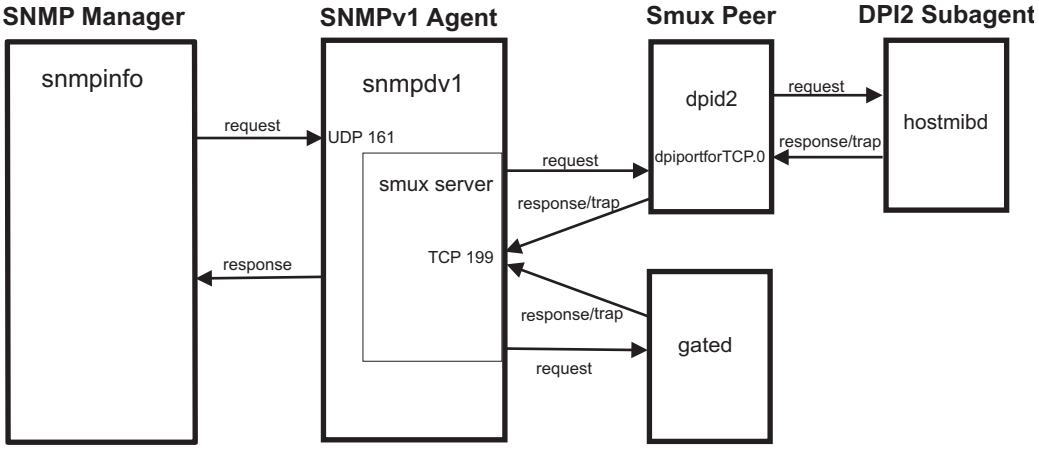


Figure 28. The primary parts of the SNMPv1 architecture

This illustration shows and example of the **SNMPv1** architecture. The DPI2 subagent, smux peer, **SNMP** manager, and **SNMP** agent are shown. In addition, how they communicate with each other is shown.

The **SNMP** daemon receives and transmits all **SNMP** protocol messages through the **Transmission Control Protocol/Internet Protocol (TCP/IP) User Datagram Protocol (UDP)**. Requests are accepted on well-known port 161. Traps are transmitted to the hosts listed in the trap entries in the `/etc/snmpd.conf` file that are listening on well-known port 162.

When a request is received, the source IP address and the community name are checked against a list containing the IP addresses, community names, permissions, and views as specified in the community and view entries in the `/etc/snmpd.conf` file. The **snmpd** agent reads this file at startup and on a **refresh**

command or a **kill -1** signal. If no matching entry is found, the request is ignored. If a matching entry is found, access is allowed according to the permissions specified in the community and view entries for that IP address, community, and view name association in the `/etc/snmpd.conf` file. Both the message and the PDU must be encoded according to the ASN.1 rules.

This authentication scheme is not intended to provide full security. If the **SNMP** daemon is used only for get and get-next requests, security might not be a problem. If set requests are allowed, the set privilege can be restricted.

See the `/etc/snmpd.conf` file in *Files Reference* for further information. See Management Information Base (MIB) in *Communications Programming Concepts* for further information.

SNMP request processing:

There are three types of request PDUs that can be received by the **SNMP** daemon.

The request types are defined in RFC 1157, and the PDUs all have the following format:

Table 83. Request PDU format

request-ID	error-status	error-index	variable-bindings
GET	0	0	*VarBindList*
GET-NEXT	0	0	*VarBindList*
SET	0	0	*VarBindList*

The request-ID field identifies the nature of the request; the error-status field and error-index field are unused and must be set to 0 (zero); and the variable-bindings field contains a variable-length list of numeric-format instance IDs whose values are being requested. If the value of the request-ID field is SET, the variable-bindings field is a list of pairs of instance IDs and values.

Read Using the Management Information Base (MIB) Database in *Communications Programming Concepts* for a discussion of the three request types.

SNMP response processing:

Response PDUs have nearly the same format as request PDUs.

Table 84. Response PDU format

request-ID	error-status	error-index	variable-bindings
GET-RESPONSE	*ErrorStatus*	*ErrorIndex*	*VarBindList*

If the request was successfully processed, the value for both the error-status and error-index field is 0 (zero), and the variable-bindings field contains a complete list of pairs of instance IDs and values.

If any instance ID in the variable-bindings field of the request PDU was not successfully processed, the SNMP agent stops processing, writes the index of the failing instance ID into the error-index field, records an error code in the error-status field, and copies the partially completed result list into the variable-bindings field.

RFC 1157 defines the following values for the error-status field:

Table 85. Values for the error-status field

Value	Value	Explanation
noError	0	Processing successfully completed (error-index is 0).
tooBig	1	The size of the response PDU would exceed an implementation-defined limit (error-index is 0).
noSuchName	2	An instance ID does not exist in the relevant MIB view for GET and SET request types or has no successor in the MIB tree in the relevant MIB view for GET-NEXT requests (nonzero error-index).
badValue	3	For SET requests only, a specified value is syntactically incompatible with the type attribute of the corresponding instance ID (nonzero error-index).
readOnly	4	Not defined.
genErr	5	An implementation-defined error occurred (nonzero error- index); for example, an attempt to assign a value that exceeds implementation limits.

SNMP trap processing:

Trap PDUs are defined by RFC 1157 to have the format shown in this table.

Table 86. Trap PDU format

enterprise	agent-address	generic-trap	specific-trap	time-stamp	variable-bindings
Object ID	*Integer*	*Integer*	*Integer*	*TimeTicks*	*VarBindList*

The fields are used as follows:

Item	Description
enterprise	The object identifier assigned to the vendor implementing the agent. This is the value of the **sysObjectID** variable, and it is unique for each implementer of an **SNMP** agent. The value assigned to this implementation of the agent is **1.3.6.1.4.1.2.3.1.2.1.1.3**, or **risc6000snmpd.3**.
agent-address	IP address of the object generating the trap.
generic-trap	Integer, as follows: **0** *coldStart* **1** *warmStart* **2** *linkDown* **3** *linkUp* **4** *authenticationFailure* **5** *egpNeighborLoss* **6** *enterpriseSpecific*
specific-trap	Unused, reserved for future development.
time-stamp	Elapsed time, in hundredths of a second, from the last reinitialization of the agent to the event generating the trap.
variable-bindings	Extra information, dependent on *generic-trap* type.

The following generic-trap values indicate that certain system events have been detected:

Item	Description
coldStart	The agent is reinitializing. Configuration data or MIB variable values, or both, might have changed. Restart the measurement epochs.
warmStart	The agent is reinitializing but configuration data or MIB variable values have not changed. In this implementation of the **SNMP** agent, a *warmStart* trap is generated when the `/etc/snmpd.conf` file is reread. The configuration information in the `/etc/snmpd.conf` file is for agent configuration that has no side effects on SNMP manager databases. Measurement epochs should not be restarted.
linkDown	The agent has detected that a known communications interface has been disabled.
linkUp	The agent has detected that a known communications interface has been enabled.
authenticationFailure	A message was received that could not be authenticated.
egpNeighborLoss	An **Exterior Gateway Protocol (EGP)** neighbor was lost. This value is only generated when the agent is running on a host that runs the **gated** daemon using **EGP**.
enterpriseSpecific	Not implemented; reserved for future use.

The *linkDown* and *linkUp* traps contain a single instance ID/value pair in the variable-bindings list. The instance ID identifies the **ifIndex** of the adapter that was disabled or enabled, and the value is the **ifIndex** value. The trap for *egpNeighborLoss* also contains a binding consisting of the instance ID and value of *egpNeighAddr* for the lost neighbor.

SNMP daemon support for the EGP family of MIB variables

If the agent host is running the **gated** daemon with the **Exterior Gateway Protocol (EGP)** enabled, there are several Management Information Base (MIB) variables in the **EGP** group supported by the **gated** daemon which the **snmpd** agent can access.

The following **EGP** MIB variables have a single, unique instance:

Item	Description
egpInMsgs	Number of **EGP** messages received without error.
egpInErrors	Number of **EGP** messages received in error.
egpOutMsgs	Total number of **EGP** messages transmitted by the **gated** daemon running on the agent's host.
egpOutErrors	Number of **EGP** messages that could not be sent by the agent host **gated** daemon because of resource limitations.
egpAs	Autonomous system number of the agent host **gated** daemon.

The following **EGP** MIB variables have an instance for each **EGP** peer or neighbor acquired by the agent host **gated** daemon:

Item	Description
egpNeighState	The state of this **EGP** peer: **1** idle **2** acquisition **3** down **4** up **5** cease.
egpNeighAddr	IP address of this **EGP** peer.
egpNeighAs	Autonomous system number of this **EGP** peer. Zero (0) indicates the autonomous system number of this peer is not yet known.
egpInNeighMsgs	Number of **EGP** messages received without error from this **EGP** peer.
egpNeighInErrs	Number of **EGP** messages received in error from this **EGP** peer.
egpNeighOutMsgs	Number of locally generated **EGP** messages to this **EGP** peer.
egpNeighOutErrs	Number of locally generated **EGP** messages not sent to this **EGP** peer because of resource limitations.
egpNeighInErrMsgs	Number of **EGP**-defined error messages received from this **EGP** peer
egpNeighOutErrMsgs	Number of **EGP**-defined error messages sent to this **EGP** peer.

Item	Description
egpNeighStateUp	Number of **EGP** state transitions to the UP state with this **EGP** peer.
egpNeighStateDowns	Number of **EGP** state transitions from the UP state to any other state with this **EGP** peer.
egpNeighIntervalHello	Interval between **EGP** Hello command retransmissions in hundredths of a second.
egpNeighIntervalPoll	Interval between **EGP** poll command retransmissions in hundredths of a second.
egpNeighMode	Polling mode of this **EGP** peer. The mode can be either active (1) or passive (2).
egpNeighEventTrigger	Control variable triggers operator-initiated start and stop events to this **EGP** peer. This MIB variable can be set to start (1) or stop (2).

If the **gated** daemon is not running, or if the **gated** daemon is running but is not configured to communicate with the **snmpd** agent, or if the **gated** daemon is not configured for **EGP**, get and set requests for the values of these variables will return the *noSuchName* error response code.

The **gated** daemon configuration file, `/etc/gated.conf`, should contain the following statement:

```
snmp      yes;
```

The **gated** daemon is internally configured to be an Simple Network Management Protocol (SNMP) single multiplexer (SMUX) protocol peer, or proxy agent, of the **snmpd** daemon. When the **gated** daemon starts up, it registers the *ipRouteTable* MIB variable tree with the **snmpd** agent. If the **gated** daemon is configured for **EGP**, then the**gated** daemon also registers the **EGP** MIB variable tree. After this registration is complete, an SNMP manager can successfully make requests to the **snmpd** agent for the *ipRouteTable* an **EGP** MIB variables supported by this agent host **gated** daemon. When the **gated** daemon is running, all MIB routing information is obtained using the **gated** daemon. In this case, set requests to the *ipRouteTable* are not allowed.

The SMUX communication between the **gated** daemon and the **snmpd** daemon takes place over the well-known Transmission Control Protocol (TCP) port 199. If the **gated** daemon terminates, **snmpd** immediately unregisters the trees the **gated** daemon previously registered. If the **gated** daemon is started before the **snmpd** daemon, the **gated** daemon periodically checks for the **snmpd** daemon until the SMUX association can be established.

To configure the **snmpd** agent to recognize and allow the SMUX association with the **gated** daemon client, the user must add a SMUX entry to the `/etc/snmpd.conf` file. The client object identifier and password specified in this SMUX entry for the **gated** daemon must match those specified in the `/etc/snmpd.peers` file.

The **snmpd** agent supports set requests for the following read-write MIB I and MIB II variables:

sysContact
The textual identification of the contact person for this agent host. This information includes the name of this person and how to contact this person: for example, "Bob Smith, 555-5555, ext 5." The value is limited to 256 characters. If, for a set request, the string for this MIB variable is greater than 256 characters, the **snmpd** agent will return the error *badValue*, and the set operation is not performed. The initial value of *sysContact* is defined in `/etc.snmp.conf`. If nothing is defined, the value is null string.

Instance	Value	Action
0	"string"	The MIB variable is set to "string".

sysName

The host name for this agent host. Typically this is the fully qualified domain name of the node. The value is limited to 256 characters. If, for a set request, the string for this MIB variable is greater than 256 characters, the **snmpd** agent returns the error *badValue*, and the set operation is not performed.

Instance	Value	Action
0	"string"	The MIB variable is set to "string".

sysLocation

A textual string stating the physical location of the machine on which this **snmpd** agent resides: for example, "Austin site, building 802, lab 3C-23." The value is limited to 256 characters. If, for a set request, the string for this MIB variable is greater than 256 characters, the **snmpd** agent returns the error *badValue*, and the set operation is not performed. The initial value of *sysLocation* is defined in `/etc/snmp.conf`. If nothing is defined, the value is null string.

Instance	Value	Action
0	"string"	The MIB variable is set to "string".

ifAdminStatus

The desired state of an interface adapter on the agent's host. Supported states are up and down. The state can be set to testing but such an action has no effect on the operational state of the interface.

Instance	Value	Action
f	1	The interface adapter with **ifIndex f** is enabled.

Note: It is possible that the *ifAdminStatus* value can be set to up or down, yet the actual operational change of the interface failed. In such a case, a get request of the *ifAdminStatus* might reflect *up* while an *ifOperStatus* for that interface might reflect *down*. If such a situation occurs, the network administrator would issue another set request to set *ifAdminStatus* to up to attempt the operational change again.

atPhysAddress

The hardware address portion of an address table binding on the agent host (an entry in the Address Resolution Protocol table). This is the same MIB variable as *ipNetToMediaPhysAddress*.

Instance	Value	Action
f.1.n.n.n.n	hh:hh:hh:hh:hh:hh	For the interface with **ifIndex f,** any existing ARP table binding for IP address n.n.n.n is replaced with the binding (n.n.n.n, hh:hh:hh:hh:hh:hh). If a binding did not exist, the new binding is added. hh:hh:hh:hh:hh:hh is a twelve-hexadecimal-digit hardware address.

atN0etAddress

The IP address corresponding to the hardware or physical address specified in *atPhysAddress*. This is the same MIB variable as *ipNetToMediaNetAddress*.

Instance	Value	Action
f.1.n.n.n.n	m.m.m.m	For the interface with **ifIndex f,** an existing ARP table entry for IP address n.n.n.n is replaced with IP address m.m.m.m.

ipForwarding

Indicates whether this agent host is forwarding datagrams.

Table 87. ipforwarding

Instance	Value	Action
0	1	If the agent host has more than one active interface, then the **TCP/IP** kernel is configured to forward packets. If the agent host has only one active interface, the set request fails.
0	2	The **TCP/IP** kernel on the agent host is configured to not forward packets.

ipDefaultTTL

The default time-to-live (TTL) value inserted into IP headers of datagrams originated by the agent host.

Instance	Value	Action
0	n	The default time-to-live value used by IP protocol support is set to the integer n.

ipRouteDest

The destination IP address of a route in the route table.

Instance	Value	Action
n.n.n.n	m.m.m.m	The destination route for route n.n.n.n is set to the IP address m.m.m.m.

ipRouteNextHop

The gateway by which a destination IP address can be reached from the agent host (an entry in the route table).

Instance	Value	Action
n.n.n.n	m.m.m.m	A route table entry to reach network n.n.n.n using gateway m.m.m.m is added to the route table. The host portion of the IP address n.n.n.n must be 0 to indicate a network address.

ipRouteType

The state of a route table entry on the agent host (used to delete entries).

Instance	Value	Action
h.h.h.h	1	Any route to host IP address h.h.h.h is deleted.
n.n.n.n	2	Any route to host IP address n.n.n.n is deleted.

ipNetToMediaPhysAddress

The hardware address portion of an address table binding on the agent host (an entry in the ARP table). This is the same MIB variable as *atPhysAddress*.

Instance	Value	Action
f.1.n.n.n.n	hh:hh:hh:hh:hh:hh	For the interface with **ifIndex f,** any existing ARP table binding for IP address n.n.n.n is replaced with the binding (n.n.n.n, hh:hh:hh:hh:hh:hh). If a binding did not exist, the new binding is added. hh:hh:hh:hh:hh:hh is a 12-hexadecimal-digit hardware address.

ipNetToMediaNetAddress

The IP address corresponding to the hardware or physical address specified in *ipNetToMediaPhysAddress*. This is the same MIB variable as *atNetAddress*.

Instance	Value	Action
f.1.n.n.n.n	m.m.m.m	For the interface with **ifIndex f,** an existing ARP table entry for IP address n.n.n.n is replaced with IP address m.m.m.m.

ipNetToMediaType

The type of mapping from the IP address to the physical address.

Instance	Value	Action
f.1.n.n.n.n	1	For the interface with **ifIndex f,** for an existing ARP binding from IP address to physical address, the mapping type is set to 1, or other.
f.1.n.n.n.n	2	For the interface with **ifIndex f,** for an existing ARP binding from IP address to physical address, the mapping type is set to 2, or invalid. As a side effect, the corresponding entry in the **ipNetMediaTable** is invalidated; that is, the interface is disassociated from this **ipNetToMediaTable** entry.
f.1.n.n.n.n	3	For the interface with **ifIndex f,** for an existing ARP binding from IP address to physical address, the mapping type is set to 3, or dynamic.
f.1.n.n.n.n	4	For the interface with **ifIndex f,** for an existing ARP binding from IP address to physical address, the mapping type is set to 4, or static.

snmpEnableAuthenTraps

Indicates whether the **snmpd** agent is configured to generate *authenticationFailure* traps.

Instance	Value	Action
0	1	The **snmpd** agent will generate authentication failure traps.
0	2	The **snmpd** agent will not generate authentication failure traps.

smuxPstatus

The status of an SMUX protocol peer (used to delete SMUX peers).

Instance	Value	Action
n	1	**snmpd** agent does nothing.
n	2	**snmpd** agent stops communicating with SMUX peer n.

smuxTstatus
The status of a SMUX MIB tree (used to delete MIB tree mounts).

Instance	Value	Action
l.m.m.m._ _ _ .p	1	**snmpd** agent does nothing.
l.m.m.m._ _ _ .p	2	Unmounts SMUX mounting of MIB tree m.m.m... where *l* is the length of MIB tree instance and *p* is the smuxTpriority.

The following variables are the settable variables as defined in RFC 1229. The **snmpd** daemon allows the user to set these variables. The underlying device might not allow the setting of such variables. Check with each device to see what is and is not supported.

ifExtnsPromiscuous
The status of the promiscuous mode on a given device. This is used to enable and disable promiscuous mode on a given device. The **snmpd** action is final and complete. When **snmpd** is told to turn off, promiscuous mode is turned completely off regardless of the other applications on the machine.

Instance	Value	Action
n	1	Turns on the promiscuous mode for device n.
n	2	Turns off the promiscuous mode for device n.

ifExtnsTestType
The test initiation variable. When this variable is set, the appropriate test is run for that device. An Object Identifier is the value of the variable. The specific value is dependent on the device type and the test that is to be run. Currently, the only defined test that **snmpd** knows to run is the testFullDuplexLoopBack test.

Instance	Value	Action
n	oid	Start the test specified by oid.

ifExtnsRcvAddrStatus
The address status variable. When this variable is set, the specified address comes into existence with the appropriate level of duration. **snmpd** only allows the setting of temporary addresses because it is not able to set device Object Data Manager (ODM) records and it is only allowed to set multicast or broadcast addresses.

Instance	Value	Action
n.m.m.m.m.m.m	1	Add the address as something other than a temporary or permanent address.
n.m.m.m.m.m.m	2	Remove the address from usage.
n.m.m.m.m.m.m	3	Add the address as a temporary address.
n.m.m.m.m.m.m	4	Add the address as a permanent address.

The variables listed below are the settable variables as defined in RFC 1231. The **snmpd** daemon allows the user to set these variables. The underlying device might not allow the setting of such variables. You should check with each device to see what is supported.

dot5Commands
The command the token-ring device is to perform.

Instance	Value	Action
n	1	Does nothing. Returned.
n	2	Tells the token-ring device to open.
n	3	Tells the token-ring to reset.
n	4	Tells the token-ring device to close.

dot5RingSpeed
The current ring speed or bandwidth.

Instance	Value	Action
n	1	An unknown speed.
n	2	1 megabit ring speed.
n	3	4 megabit ring speed.
n	4	16 megabit ring speed.

dot5ActMonParticipate
The object specifies whether the device participates in the active monitor selection process.

Instance	Value	Action
n	1	Participates.
n	2	Not participate.

dot5Functional
The functional mask that allows the token-ring device to specify what addresses it receives frames from.

Instance	Value	Action
n	m.m.m.m.m.m	Functional mask to be set.

The following complex timer manipulations variables are defined in the RFC as read-only but you are encouraged to make them read-write. Review the RFC to gain a full understanding of their interactions. **snmpd** allows the requestor to set them, but the device might not. Check the device driver documentation for more information. The variables are:

- dot5TimerReturnRepeat
- dot5TimerHolding
- dot5TimerQueuePDU
- dot5TimerValidTransmit
- dot5TimerNoToken
- dot5TimerActiveMon
- dot5TimerStandbyMon
- dot5TimerErrorReport
- dot5TimerBeaconTransmit
- dot5TimerBeaconReceive.

The SNMP daemon allows the user to set the following variables. The daemon uses the FDDI Station Management (SMT) 7.2 protocol standard to get the information and is determined at the microcode level. Check the microcode on the FDDI documentation to ensure that the SMT 7.2 microcode is being used.

fddimibSMTUserData
A variable holding 32 bytes of user information.

Instance	Value	Action
n	string	Stores 32 bytes of user information.

fddimibSMTConfigPolicy

The status of the configuration policies, specifically the hold policy usage.

Instance	Value	Action
n	0	Do not use the hold policy.
n	1	Use the hold policy.

fddimibSMTConnectionPolicy

The status of the connection policies in the FDDI node. See RFC 1512 for more information about the specific settable values.

Instance	Value	Action
n	k	Defines the connection policies.

fddimibSMTTNotify

The timer, expressed in seconds, used in the Neighbor Notification protocol. It has a range of 2 seconds to 30 seconds, and its default value is 30 seconds.

Instance	Value	Action
n	k	Defines the timer value.

fddimibSMTStatRptPolicy

The status of the status reporting frame generation.

Instance	Value	Action
n	1	Indicates that the node generates status reporting frames for implemented events.
n	2	Indicates that the node does not create status reporting frames.

fddimibSMTTraceMaxExpiration

This variable defines the maximum timer expiration value for trace.

Instance	Value	Action
n	k	Defines the maximum timer expiration in milliseconds.

fddimibSMTStationAction

This variable causes the SMT entity to take a specific action. See the RFC to get specific information about this variable.

Instance	Value	Action
n	k	Defines an action on the SMT entity. Values range from 1 to 8.

fddimibMACRequestedPaths

Defines the paths the medium access control (MAC) should be inserted.

Instance	Value	Action
n.n	k	Defines the requested path for the MAC.

fddimibMACFrameErrorThreshold
Threshold for when a MAC status report is generated. Defines the number of error that must occur before a report is generated.

Instance	Value	Action
n.n	k	Defines the number of errors that must be observed before a MAC status report is generated.

fddimibMACMAUnitdataEnable
This variable determines the value of the **MA_UNITDATA_Enable** flag in RMT. The default and initial value of this flag is true (1).

Instance	Value	Action
n.n	1	Marks the MA_UNITDATA_Enable flag true.
n.n	2	Marks the MA_UNITDATA_Enable flag false.

fddimibMACNotCopiedThreshold
A threshold for determining when a MAC condition report is generated.

Instance	Value	Action
n.n	k	Defines the number of errors that must be observed before a MAC condition report is generated.

The following three variables are timer variables that are interactive among themselves. Before changing any of these variables, you should have a good understanding of their meaning as defined in **RFC 1512**.

- fddimibPATHTVXLowerBound
- fddimibPATHTMaxLowerBound
- fddimibPATHMaxTReq

fddimibPORTConnectionPolicies
Specifies the connection policies for the specified port.

Instance	Value	Action
n.n	k	Defines the connection policies for the specified port.

fddimibPORTRequestedPaths
This variable is a list of permitted paths where each list element defines the port permitted paths. The first octet corresponds to `none', the second octet to `tree', and the third octet to `peer'.

Instance	Value	Action
n.n	ccc	Defines the port paths.

fddimibPORTLerCutoff

The link error rate estimate at which a link connection is broken. It ranges from 10**-4 to 10**-15 and is reported as the absolute value of the base 10 logarithm (default of 7).

Item	Description	
Instance	**Value**	**Action**
n.n	k	Defines the port LerCutoff.

fddimibPORTLerAlarm

The link error rate estimate at which a link connection generates an alarm. It ranges from 10**-4 to 10**-15 and is reported as the absolute value of the base 10 logarithm of the estimate (default is 8).

Instance	Value	Action
n.n	k	Defines the port LerAlarm.

fddimibPORTAction

This variable causes the port to take a specific action. See the RFC to get specific information about this variable.

Instance	Value	Action
n	k	Defines an action on the defined port. The values range from 1 to 6.

Note: RFC 1213 describes all variables in the *atEntry* and *ipNetToMediaEntry* tables as read-write. Set support is implemented only for the *atEntry* variables *atPhysAddress* and *atNetAddress,* and the *ipNetToMediaEntry* variables *ipNetToMediaPhysAddress, ipNetToMediaNetAddress,* and *ipNetToMediaType.* To accept set requests that might specify the remaining unsupported attributes in these two tables, set requests for the remaining variables are accepted in *atIfIndex* and *ipNetToMediaIfIndex.* No error response is returned to the set request originator, but a subsequent get request will show that the original values are retained.

In the *ipRouteEntry* table, RFC 1213 describes all variables except *ipRouteProto*as read-write. As mentioned above, set support is implemented only for the variables *ipRouteDest*, *ipRouteNextHop*, and *ipRouteType*. To accept set requests that might specify several unsupported route attributes, set requests for the remaining variables in the *ipRouteEntry* table are accepted: *ipRouteIfIndex*, *ipRouteMetric1*, *ipRouteMetric2*, *ipRouteMetric3*, *ipRouteMetric4*, *ipRouteMetric5*, *ipRouteAge*, and *ipRouteMask*. No error response is returned to the set request originator, but a subsequent get request will show that the original values are retained. The **snmpd** daemon does not coordinate routing with the **routed** daemon. If the **gated** daemon is running and has registered the *ipRouteTable* with the **snmpd** daemon, set requests to the *ipRouteTable* are not allowed.

RFC 1229 describes settable variables that **snmpd** allows. See the previous entries for actual deviations.

The following examples use the **snmpinfo** command. It is assumed that the **snmpinfo** default community name, public, has read-write access for the respective MIB subtree.

```
snmpinfo -m set sysContact.0="Primary contact: Bob Smith, office phone: 555-5555,
beeper: 9-123-4567. Secondary contact: John Harris, phone: 555-1234."
```

This command sets the value of `sysContact.0` to the specified string. If an entry for `sysContact.0` already exists, it is replaced.

```
snmpinfo -m set sysName.0="bears.austin.ibm.com"
```

This command sets the value of `sysName.0` to the specified string. If an entry for `sysName.0` already exists, it is replaced.

```
snmpinfo -m set sysLocation.0="Austin site, building 802, lab 3C-23, southeast
corner of the room."
```

This command sets the value of `sysLocation.0` to the specified string. If an entry for `sysLocation.0` already exists, it is replaced.

```
snmpinfo -m set ifAdminStatus.2=2
```

This command disables the network interface adapter which has the ifIndex of 2. If the assigned value is 1, the interface adapter is enabled.

```
snmpinfo -m set atPhysAddress.2.1.192.100.154.2=02:60:8c:2e:c2:00
snmpinfo -m set ipNetToMediaPhysAddress.2.1.192.100.154.2=02:60:8c:2e:c2:00
```

These two commands change the hardware address in the ARP table entry for `192.100.154.2` to `02:60:8c:2e:c2:00`. These two commands affect the same ARP table entry. The MIB variable *atPhysAddress* is a deprecated variable and is being replaced with the MIB variable *ipNetToMediaPhysAddress*. Thus, *atPhysAddress* and *ipNetToMediaPhysAddress* access the same structure in the TCP/IP kernel ARP table.

```
snmpinfo -m set atNetAddress.2.1.192.100.154.2=192.100.154.3
snmpinfo -m set ipNetToMediaNetAddress.2.1.192.100.154.2=192.100.154.3
```

These commands change the IP address in the ARP table entry for `192.100.154.2` to `192.100.154.3`. These two commands affect the same ARP table entry. The MIB variable *atNetAddress* is a deprecated variable and is being replaced with the MIB variable *ipNetToMediaNetAddress*. Thus, *atNetAddress* and *ipNetToMediaNetAddress* access the same structure in the TCP/IP kernel ARP table.

```
snmpinfo -m set ipForwarding.0=1
```

This command sets the **TCP/IP** kernel so that it can forward packets if the agent host has more than one interface that is up. If the host has only one active interface, then the set request fails and the **snmpd** agent returns the error, *badValue*.

```
snmpinfo -m set ipDefaultTTL=50
```

This command allows an IP datagram using default time-to-live (TTL) to pass through up to 50 gateways before being discarded. When each gateway processes a datagram, the gateway subtracts 1 from the time-to-live field. In addition, each gateway decrements the time-to-live field by the number of seconds the datagram waited for service at that gateway before passing the datagram on to the next destination.

```
snmpinfo -m set ipRouteDest.192.100.154.0=192.100.154.5
```

This command sets the destination IP address of the route associated with `192.100.154.0` to the IP address `192.100.154.5`, assuming route `192.100.154` already existed.

```
snmpinfo -m set ipRouteNextHop.192.100.154.1=129.35.38.47
```

This command sets a route to host `192.100.154.1` using the gateway host `129.35.38.47`, assuming route `192.100.154.1` already existed.

```
snmpinfo -m set ipRouteNextHop.192.100.154.0=192.100.154.7
```

This command sets a route to the class C network `192.100.154` using the gateway host `192.100.154.7`, assuming route `192.100.154.0` already existed. Note that the host part of the address must be 0 to indicate a network address.

```
snmpinfo -m set ipRouteType.192.100.154.5=2
```

This command deletes any route to host `192.100.154.5`.

```
snmpinfo -m set ipRouteDest.129.35.128.1=129.35.128.1
                ipRouteType.129.35.128.1=3
             ipRouteNextHop.129.35.128.1=129.35.128.90
```

This command creates a new route from host 129.35.128.90 to 129.35.128.1 as a gateway.

```
snmpinfo -m set ipNetToMediaType.2.1.192.100.154.11=4
```

This command sets the ARP table entry for 192.100.154.11 to static.

```
snmpinfo -m set snmpEnableAuthenTraps=2
```

This command causes the **snmpd** agent on the specified host to not generate *authenticationFailure* traps.

```
snmpinfo -m set smuxPstatus.1=2
```

This command invalidates the SMUX peer 1. The result is that the connection between the **snmpd** agent and this SMUX peer is terminated.

```
snmpinfo -m set smuxTstatus.8.1.3.6.1.2.1.4.21.0=2
```

This command invalidates or removes the mounting of the SMUX tree 1.3.6.1.2.1.4.21, the *ipRoute* Table. The first number in the instance indicates the number of levels in the SMUX tree identifier. The final number in the instance indicates the smuxTpriority. In this example, there are 8 levels in the SMUX tree identifier: 1.3.6.1.2.1.4.21. The priority, 0, is the highest priority.

```
snmpinfo -m set ifExtnsPromiscuous.1=1 ifExtnsPromiscuous.2=2
```

This command turns on promiscuous mode for the first device in the interfaces table and turns off promiscuous mode for the second device in the interfaces table.

```
snmpinfo -m set ifExtnsTestType.1=testFullDuplexLoopBack
```

This command starts the testFullDuplexLoopBack test on interface 1.

```
snmpinfo -m set ifExtnsRcvAddrStatus.1.129.35.128.1.3.2=2
```

This command tells interface 1 to remove physical address 129.35.128.1.3.2 from its list of acceptable addresses.

```
snmpinfo -m set dot5Commands.1=2
```

This command tells the first interface to do an open.

```
snmpinfo -m set dot5RingSpeed.1=2
```

This command tells the first interface to set it ring speed to 1 megabit.

```
snmpinfo -m set dot5ActMonParticipate.1=1
```

This command tells the first interface to participate in the active monitor selection process.

```
snmpinfo -m set dot5Functional.1=255.255.255.255.255.255
```

This command sets the functional address mask to allow everything.

```
snmpinfo -m set fddimibSMTUserData.1="Greg's Data"
```

This command sets the user data on the first SMT entity to "Greg's Data".

```
snmpinfo -m set fddimibMACFrameErrorThreshold.1.1=345
```

This command sets the threshold for frame errors to 345 on the first MAC of the first SMT entity.

Note: All of the variables described previously fall into one of the listed methods used to set the variable.

See “Address Resolution Protocol” on page 138 and “Internet addresses” on page 164 for more information on protocols and Internet addresses.

SNMP daemon troubleshooting

Troubleshooting tips for the **SNMP** daemon include solving termination problems, MIB variable access problems, MIB variable access in community entry problems, noSuchName problems, no response from agent problems, and daemon failures.

Daemon termination problem

If the **snmpd** agent is not behaving as expected, the following are some hints to help determine and correct the problem. It is strongly recommended that you start up the **snmpd** agent with some type of logging. If invoking the **snmpd** daemon causes problems, it is strongly recommended that the **syslogd** daemon be set up for logging at the daemon facility and DEBUG severity level. See the **snmpd** command in *Commands Reference, Volume 5* and the **snmpd.conf** file in *Files Reference*for more information on **snmpd** logging.

If the **snmpd** daemon terminates as soon as it is invoked, the following are possible reasons for failure and probable solutions:

- The reason the **snmpd** daemon terminated will be logged in the **snmpd** log file or the configured **syslogd** log file. Check the log file to see the **FATAL** error message.

  Solution: Correct the problem and restart the **snmpd** daemon.
- The **snmpd** command line usage was incorrect. If the **snmpd** command was invoked without the System Resource Controller (SRC), the required usage statement is echoed to the screen. If the **snmpd** daemon was invoked under SRC control, the usage message is not echoed to the screen. Check the log file to see the usage message.

  Solution: Invoke the **snmpd** command with the correct usage statement.
- The **snmpd** daemon must be invoked by the root user.

  Solution: Switch to the root user and restart the **snmpd** daemon.
- The **snmpd.conf** file must be owned by the root user. The **snmpd** agent verifies the ownership of the configuration file. If the file is not owned by the root user, the **snmpd** agent terminates with a fatal error.

  Solution: Make sure you are the root user, change the ownership of the configuration file to the root user, and restart the **snmpd** daemon.
- The **snmpd.conf** file must exist. If the **-c** flag is not specified in the configuration file on the **snmpd** command line, the **/etc/snmpd.conf** file does not exist. If the **/etc/snmpd.conf** file is accidentally removed, reinstall the **bos.net.tcp.client** image or else reconstruct the file with the appropriate configuration entries as defined in the **snmpd.conf** file management page. If the configuration file is specified with the**-c** flag on the **snmpd** command line, make sure that the file exists and that the file is owned by the root user. The full path and file name of the configuration file must be specified or else the default **/etc/snmpd.conf** file will be used.

  Solution: Make sure the specified configuration file exists and that this file is owned by the root user. Restart the **snmpd** daemon.
- The `udp port 161` is already bound. Make sure that the **snmpd** daemon is not already running. Issue the **ps -eaf | grep snmpd** command to determine if an **snmpd** daemon process is already executing. Only one **snmpd** agent can bind to `udp port 161`.

  Solution: Either kill the existing **snmpd** agent or do not try to start up another **snmpd** daemon process.

Daemon failure problem

If the **snmpd** daemon fails when you issue a **refresh** or a **kill -1** signal, the following are possible reasons for failure and probable solutions:

- The reason the **snmpd** daemon terminated is logged in the **snmpd** log file or the configured **syslogd** log file. Check the log file to see the FATAL error message.

  Solution: Correct the problem and restart the **snmpd** daemon.
- Make sure that the complete path and file name of the configuration file is specified when the **snmpd** daemon is invoked. The **snmpd** daemon forks and changes to the root directory at invocation. If the complete path name of the configuration file is not specified, the **snmpd** agent cannot find the file on a refresh. This is a fatal error and will cause the **snmpd** agent to terminate.

  Solution: Specify the complete path and file name of the **snmpd** configuration file. Make sure the configuration file is owned by the root user. Restart the **snmpd** daemon.
- Make sure that the **snmpd** configuration file still exists. The file may have been accidentally removed after the **snmpd** agent was invoked. If the **snmpd** agent cannot open the configuration file, the **snmpd** agent terminates.

  Solution: Recreate the **snmpd** configuration file, make sure the configuration file is owned by the root user, and restart the **snmpd** daemon.

MIB variable access problem

If Management Information Base (MIB) variables cannot be accessed from the **snmpd** agent; if the **snmpd** agent is running, but the **Simple Network Management Protocol (SNMP)** manager application times out waiting for a response from the **snmpd** agent, try the following:

- Check the network configuration of the host on which the **snmpd** agent is running using the **netstat -in** command. Verify the lo0, loopback, device is up. If the device is down, an * (asterisk) displays to the left of the lo0. The lo0 must be up for the **snmpd** agent to service requests.

  Solution: Issue the following command to start up the loopback interface:

  ```
  ifconfig lo0 inet up
  ```
- Verify that the **snmpd** daemon has a route to the host where the requests are issued.

  Solution: On the host where the **snmpd** daemon is running, add a route to the host where the **route add** command is issued. See the **route** command in *Commands Reference, Volume 4*for more information.
- Check to see that the host name and the host IP address are the same value.

  Solution: Reset the host name to correspond to the host IP address.
- Check to see that *localhost* is defined to be the lo0 IP address.

  Solution: Define *localhost* to be the same address used by the lo0 IP address (usually `127.0.0.1`).

MIB variable access in community entry problem

If a community entry is specified in the configuration file with a MIB view name, but MIB variables cannot be accessed, check the following:

- Make sure that you have correctly specified the community entry. If you have specified a view name in the community entry, all fields in the community are absolutely required.

  Solution: Specify all fields in the community entry in the configuration file. Refresh the **snmpd** agent and try your request again.
- Make sure the access mode in the community entry corresponds with your request type. If you are issuing a **get** or **get-next** request, make sure that the community has read-only or read-write permission. If you are issuing a **set** request, make sure that the community has read-write permission.

  Solution: Specify the correct access mode in the community entry. Refresh the **snmpd** agent and try your request again.
- Make sure that a view entry for the specified view name is specified in the community entry in the configuration file. If there is a specified view name in the community entry, but there is no corresponding view entry, the **snmpd** agent does not allow access for that community. A view entry is absolutely required for a view name specified in a community entry in the configuration file.

Solution: Specify a view entry for the view name specified in the community entry. Refresh the **snmpd** agent and try your request again.

- If `iso` is specified as the MIB subtree for the view entry, verify that `iso.3` is specified. The instance of `3` is required for the **snmpd** agent to access the `org` portion of the `iso` tree.

  Solution: Specify the MIB subtree as `iso.3` in the view entry. Refresh the **snmpd** agent and try your request again.
- Check the *IP address* and *network mask* in the community entry. Verify that the host issuing the SNMP request is included in the community being specified with the community name.

  Solution: Change the *IP address* and *network mask* fields in the community entry in the configuration file to include the host that is issuing the SNMP request.

No response from agent problem

If the *IP address* in the community is specified as 0.0.0.0, but there is no response from the **snmpd** agent, try the following:

- Check the *network mask* field in the community entry. For general access to this community name, the *network mask* must be **0.0.0.0**. If the *network mask* is specified to be `255.255.255.255`, the **snmpd** agent is configured to not allow any requests with the specified community name.

  Solution: Specify the *network mask* in the community entry to 0.0.0.0. Refresh the **snmpd** agent and try the request again.
- Make sure the access mode in the community entry corresponds with the request type. When issuing a **get** or **get-next** request, make sure that the community has read-only or read-write permission. If you are issuing a **set** request, make sure that the community has read-write permission.

  Solution: specify the correct access mode in the community entry. Refresh the **snmpd** agent and try your request again.

noSuchName Problem

If, in attempting to set an MIB variable that the **snmpd** agent is supposed to support, a `noSuchName` error message is returned, the following might be the reason:

The set request issued did not include a community name for a valid community with write access. The **SNMP** protocol dictates that a set request with a community with inappropriate access privileges be answered with the `noSuchName` error message.

Solution: Issue the set request with a community name for a community that has write privileges and includes the host from which the set request is issued.

Network File System

The Network File System (NFS) is a mechanism for storing files on a network. It is a distributed file system that allows users to access files and directories located on remote computers and treat those files and directories as if they were local.

For example, users can use operating system commands to create, remove, read, write, and set file attributes for remote files and directories.

The NFS software package includes commands and daemons for NFS, Network Information Service (NIS), and other services. Although NFS and NIS are installed together as one package, each is independent and each is configured and administered individually.

AIX 5.3 and later supports the NFS version 2, 3, and 4 protocols. NFS version 4 is the most recently-defined version of NFS, and it is described by RFC 3530. Additional details on AIX support of NFS version 4 will be discussed later in this section. NFS clients use the NFS version 3 protocol by default.

NFS services

NFS provides its services through a client-server relationship.

The computers that make their *file systems*, or *directories*, and other resources available for remote access are called *servers.* The act of making file systems available is called *exporting*. The computers and their processes that use server resources are considered *clients*. After a client mounts a file system that a server exports, the client can access the individual server files (access to exported directories can be restricted to specific clients).

The major services provided by NFS are:

Table 88. NFS services

Service	Description
Mount service	Mounts from the **/usr/sbin/rpc.mountd** daemon on the server and the **/usr/sbin/mount** command on the client. This service is only available on NFS version 2 and version 3.
Remote File access	Accesses from the **/usr/sbin/nfsd** daemon on the server and the **/usr/sbin/biod** daemon on the client.
Remote execution service	Executes from the **/usr/sbin/rpc.rexd** daemon on the server and the **/usr/bin/on** command on the client.
Remote System Statistics service	Compiles from the **/usr/sbin/rpc.rstatd** daemon on the server and the **/usr/bin/rup** command on the client.
Remote User Listing service	Lists from the **/usr/lib/netsvc/rusers/rpc.rusersd** daemon on the server and the **/usr/bin/rusers** command on the client.
Boot Parameters service	Provides startup parameters to Sun Operating System diskless clients from the **/usr/sbin/rpc.bootparamd** daemon on the server.
Remote Wall service	Protects from the **/usr/lib/netsvc/rwall/rpc.rwalld** daemon on the server and the **/usr/sbin/rwall** command on the client.
Spray service	Sends a one-way stream of Remote Procedure Call (RPC) packets from the **/usr/lib/netsvc/spray/rpc.sprayd** daemon on the server and the **/usr/sbin/spray** command on the client.
PC authentication service	Provides a user authentication service for PC-NFS from the **/usr/sbin/rpc.pcnfsd** daemon on the server.
Enhanced security service	Provides access on both the client and server to more advanced security services, such as Kerberos 5. The **/usr/sbin/gssd** daemon provides NFS with access to security services provided by the Network Authentication Service. The Network Authentication Service and the Cryptographic Library filesets (`krb5.client.rte`, **krb5.server.rte**, and **modcrypt.base**) must be installed. These filesets can be installed from the AIX Expansion Pack.
Identity translation service	Performs translation between security principals, NFS version 4 identity strings, and their corresponding numeric system IDs. In addition, mapping of identity information from foreign NFS version 4 domains is provided. These services are provided by the **/usr/sbin/nfsrgyd** daemon.

Note: A computer can be both an NFS server and an NFS client simultaneously.

NFS version 2 and 3 servers are *stateless*, meaning that the server does not retain any transaction information about its clients. A single NFS transaction corresponds to a single, complete file operation. NFS requires that the client remember any information needed for later NFS use.

An NFS version 4 server is stateful because of the file open and file locking operations defined in the NFS version 4 protocol.

NFS Access Control Lists support

The AIX NFS version 4 implementation supports two ACL types: NFS4 and AIXC.

The authoritative source for access checking lies in the underlying file system exported by the NFS server. The file system takes into consideration the file's access controls (ACLs or permission bits), the caller's credentials, and other local system restrictions that might apply. Applications and users should not assume that examination of UNIX mode bits or ACLs alone can be used to conclusively predict access.

The **aclget**, **aclput**, and **acledit** commands can be used on the client to manipulate either NFS or AIX ACLs. For more information, see Access Control Lists in *Security*.

NFS RBAC

NFS provides support to Role Based Access Control (RBAC). The NFS client and server commands are RBAC enabled.

This allows non-root users to execute NFS commands once the administrator assigns the RBAC role of the command to the user. To see the list of authorization strings and privileges associated with NFS commands, refer /etc/security/privcmds file on the system.

NFS4 ACL

NFS4 ACL is the ACL defined by the NFS version 4 protocol.

NFS4 ACL is platform-independent, so it can be supported by other vendors' clients or servers. NFS version 4 clients and servers are not required to support NFS4 ACL.

In an AIX server, if an underlying physical file system instance supports NFS4 ACL, then the AIX NFS4 server supports NFS4 ACL for that file system instance. Most physical file systems types on AIX do not support NFS4 ACL. These file system types include but are not limited to CFS, UDF, JFS, and JFS2 with extended attribute version 1. All instances of JFS2 with extended attribute version 2 support NFS4 ACL.

NFS version 4 client file systems are able to read and write the NFS4 ACL if the exported NFS version 4 file system instance on the server supports NFS4 ACL.

AIX ACL

The AIXC ACL is an access control list that is a proprietary of AIX servers.

It is not defined by the NFS version 4 protocol, and it is understood only by AIX servers and clients.

On an NFS version 4 server, AIXC ACL is supported when the underlying file system instance support AIXC ACL. All instances of JFS and JFS2 support the AIXC ACL.

An NFS version 4 client has a mount option that enables or disables support for AIX ACL. The default is to not support AIXC ACL. A user on an NFS version 4 client file system is able to read and write AIXC ACL when both the client and the server are running AIX, the underlying physical file system instance on the server supports AIXC ACL, and the AIX client mounts the file system instance with AIXC ACL enabled. AIXC ACL support in NFS version 4 is similar to the AIXC ACL support in AIX NFS version 2 and NFS version 3 implementations.

All instances of a JFS2 file system with extended attribute version 2 support both AIXC ACL and NFS4 ACL. A file in this type of file system may have mode bits only (no ACL), an NFS4 ACL, or an AIXC ACL. But, it cannot have NFS4 ACL and AIXC ACL at the same time.

The **aclgettypes** command can be used to determine the ACL types the can be read and written on a file system instance. This command may return different output when it runs against a physical file system on an NFS version 4 server locally than when it runs against the same file system on a NFS version 4 client. For example, a NFS version 4 file system instance on and NFS version 4 server may support NFS4

ACL and AIXC ACL, but the client is only configured to send and receive NFS4 ACL. In this case, when the **aclgettypes** command is executed from an NFS version 4 client file system, only NFS4 is returned. Also, if a user on the client requests an AIXC ACL, an error is returned.

Cache File System support

The Cache File System (CacheFS) is a general-purpose file system caching mechanism that 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 client-per-server ratio, reduces server and network loads and improves performance for clients on slow links, such as Point-to-Point Protocol (PPP).

A cache is created on the client machine so file systems specified to be mounted in the cache can be accessed locally instead of across the network. Files are placed in the cache when a user first requests access to them. The cache does not get filled until the user requests access to a file or files. Initial file requests may seem slow, but subsequent uses of the same files may be faster.

Note:

1. You cannot cache the / (root) or `/usr` file systems.
2. You can mount only file systems that are shared. (See the **exportfs** command in *Commands Reference, Volume 2*.)
3. There is no performance gain in caching a local Journaled File System (JFS) disk file system.
4. You must have root or system authority to do the tasks in the following table.

Table 89. CacheFS tasks

Task	SMIT fast path	Command or file
Set up a cache	`cachefs_admin_create`	**cfsadmin -c** *MountDirectoryName*[1].
Specifying Files for Mounting	`cachefs_mount`	**mount -F cachefs -o backfstype=***FileSysType*, **cachedir=***CacheDirectory*[,*options*] BackFileSystem *MountDirectoryName*[2] or edit `/etc/`**filesystems**.
Modify the Cache	`cachefs_admin_change`	remove the cache, then recreate it using appropriate **mount** command options.
Display Cache Information	`cachefs_admin_change`	**cfsadmin -l** *MountDirectoryName.*
Remove a Cache	`cachefs_admin_remove`	1. Unmount the file system: **umount** *MountDirectoryName* 2. Determine the cache ID: **cfsadmin -l** *MountDirectoryName* 3. Delete the file system: **cfsadmin -d** *CacheID CacheDirectory*
Check File System Integrity	`cachefs_admin_check`	**fsck_cachefs***CacheDirectory*[3].

Notes:

1. After you have created the cache, do not perform any operations within the cache directory (`cachedir`) itself. This causes conflicts within the CacheFS software.
2. If you use the **mount** command option to specify files for mounting, the command must be reissued each time the system is restarted.
3. Use the **-m** or **-o** options of the **fsck_cachefs** command to check the file systems without making any repairs.
4. After migrating the system to AIX Version 6.1 or later from previous versions of AIX, old cache file systems that are created in the older version of AIX must be removed and recreated.

NFS mapped file support

NFS mapped file support allows programs on a client to access a file as though it were in memory.

By using the **shmat** subroutine, users can map areas of a file into their address space. As a program reads and writes into this region of memory, the file is read into memory from the server or updated as needed on the server.

Mapping files over NFS is limited in three ways:

- Files do not share information well between clients.
- Changes to a file on one client using a mapped file are not seen on another client.
- Locking and unlocking regions of a file is not an effective way to coordinate data between clients.

If an NFS file is to be used for data sharing between programs on different clients, use record locking and the regular **read** and **write** subroutines.

Multiple programs on the same client can share data effectively using a mapped file. Advisory record locking can coordinate updates to the file on the client, provided that the entire file is locked. Multiple clients can share data-using mapped files only if the data never changes, as in a static database.

NFS proxy serving

AIX supports Network File System (NFS) proxy serving. An AIX server can concurrently export locally accessible file systems and proxy exports. The exported proxy view can be mounted by NFS clients.

AIX NFS proxy serving uses disk caching of accessed data to serve similar subsequent requests locally with reduced network traffic to the back-end server. Proxy serving can potentially extend NFS data access over slower or less reliable networks with improved performance and reduced network traffic to the primary server where the data resides. Depending on availability and content management requirements, proxy serving can provide a solution for extending NFS access to network edges without the need for copying data. You can configure AIX NFS proxy serving using the **mknfsproxy** command.

Proxy caching can be used with both the NFS v3 and the NFS v4 protocols. The protocol between the proxy and the connected clients can be either NFS v3 or NFS v4 when the NFS v4 protocol is used between the proxy and the back-end server. However, when the NFS v3 protocol is used, the protocol between the proxy and the connected clients must be the NFS v3 protocol. Both reads and writes of data are supported in addition to byte range advisory locks.

The krb5, krb5i, and krb5p security methods can be used between the proxy server and its connected clients. These methods can also be used between the proxy server and the main server. Using ticket forwarding technology through proxy, you can authenticate on the client and be authenticated to the main server. To take advantage of this technology, use the **kinit** command with the **-f** option when you are performing Kerberos authentication. If **auth_sys** security is used between the proxy and the back-end server, when you access the back-end server, the proxy server maps Kerberos client accesses into **auth_sys** attributes. For best results, the proxy server and the back-end server should share the same user and group identity definitions.

The following restrictions apply to NFS proxy serving:

- Proxy serving requires connected clients using TCP.
- Because proxy serving provides a way for NFS v3 clients to browse through the NFS v4 exported namespace without using the **mount** and **unmount** commands, you must use the **mknfsproxy** command with the **mfsid** option when you are constructing the proxy file system.
- The cache file system used with proxy serving must be an Enhanced JFS file system (JFS2).
- Proxy serving runs CacheFS on top of an AIX client mounted to the back-end NFS server. The concurrent I/O (CIO) feature, available with the AIX NFS client, enhances CacheFS performance. Attempts to directly access the underlying NFS client mount might fail because of conflicts with open attempts of CIO.

Types of NFS mounts

There are three types of NFS mounts: predefined, explicit, and automatic.

Predefined mounts are specified in the `/etc/filesystems` file. Each stanza (or entry) in this file defines the characteristics of a mount. Data such as the host name, remote path, local path, and any mount options are listed in this stanza. Predefined mounts are used when certain mounts are always required for proper operation of a client.

Explicit mounts serve the needs of the root user. Explicit mounts are usually done for short periods of time when there is a requirement for occasional unplanned mounts. Explicit mounts can also be used if a mount is required for special tasks and that mount is not generally available on the NFS client. These mounts are usually fully qualified on the command line by using the **mount** command with all needed information. Explicit mounts do not require updating the `/etc/filesystems` file. File systems mounted explicitly remain mounted unless explicitly unmounted with the **umount** command or until the system is restarted.

Automatic mounts are controlled by the **automount** command, which causes the **AutoFS** kernel extension to monitor specified directories for activity. If a program or user attempts to access a directory that is not currently mounted, then **AutoFS** intercepts the request, arranges for the mount of the file system, then services the request.

NFS exporting and mounting

Exporting and mounting directories must be understood in order to administer NFS.

An NFS server must export a file or directory, after which an NFS client may mount that file or directory. More details about these concepts are included in this section.

NFS directory exports

Exporting a directory is done on the NFS server. Exporting a directory declares that a directory in the server's namespace is available to client machines.

The exported directory is referred to as an *export*, and it includes all files under the directory that reside on the exported directory's file system.

Each export also defines access restrictions. For example, the following restrictions may be defined:

- which clients may access the exported directory
- which NFS versions the client must use to access the directory
- whether the client can write files in the export
- what security methods the client must use to access directories and files in the export

For a full description of allowable export restrictions and export semantics, see the **exportfs** command description in *Commands Reference, Volume 2* and the `/etc/exports` file description in *Files Reference*.

Note: When an export's attributes are changed, the directory must be re-exported in order for the change to take effect. A directory might need to be re-exported because of changes to other files or changes external to the server. For instance, if a client name specified in an access list is a netgroup defined in the `/etc/netgroup` file, and the definition of the client group is changed, all exports that use that netgroup in an access list must be re-exported for the change to take effect.

Similarly, if a client's IP address is changed, all exports that specify that client in an access list must be re-exported. The reason for this is because the NFS server maintains a cache of client access rights on each export. The cache is flushed on each unexport or re-export. If the access rights of an export are modified, particularly if the IP address of a client changes or if a client is removed from the access list, an unexport or re-export must be done so the client's access is correctly reflected in the cache. The NFS

server calls the **rpc.mountd** daemon to get the access rights of each client, so the **rpc.mountd** daemon must be running on the server even if the server only exports file systems for NFS version 4 access.

NFS directory mounts

An NFS client can mount a directory that has been exported by an NFS server. Mounting a directory makes the files that reside on the NFS server available to an NFS client.

A client can access files on a server if the files were exported by the server and the export restrictions allow the client to have access to the export's files. Once a client has successfully mounted a server's export onto a mount point in its namespace, the server's files for that export will exist in the client's namespace and appear as files on the local file system.

For example, assume you want to export the `/tmp` directory on server `diamond` and mount that directory on client `clip` as the `/mnt` directory. On the server, type the following command:

```
exportfs -i -o access=clip /tmp
```

This makes the `/tmp` directory available to the client.

On the client, type the following command:

```
mount diamond:/tmp /mnt
```

The directories and files in the server's `/tmp` directory will now appear in the client's `/mnt` directory.

Note:

1. There are some differences between NFS versions 2 and 3 and NFS version 4 in how mounts are handled. In NFS versions 2 and 3, the server exported the directories it wanted to make available for mounting. The NFS version 2 or 3 client then had to explicitly mount each export to which it wanted access.

   With NFS version 4, the server still specifies export controls for each server directory or file system to be exported for NFS access. From these export controls, the server renders a single directory tree of all the exported data filling in gaps between the exported directories. This tree is known as a pseudo file system and it starts at the NFS version 4 server's pseudo root. The NFS version 4 pseudo file system model allows an NFS version 4 client, depending on its implementation, to perform a single mount of the server's pseudo root in order to access all the server's exported data. The AIX NFS client supports this feature. The actual content seen by the client is dependent on the server's export controls.
2. NFS version 4 does not allow file-to-file mounting.

Mounting NFS

Clients access files on the server by first mounting server exported directories. When a client mounts a directory, it does not make a copy of that directory. Rather, the mounting process uses a series of remote procedure calls to enable a client to transparently access the directories on the server.

The following describes the mounting process:

1. When the server starts, the `/etc/rc.nfs` script runs the **exportfs** command, which reads the server `/etc/exports` file, and then tells the kernel which directories are to be exported and what access restrictions they require.
2. The **rpc.mountd** daemon and several **nfsd** daemons are then started by the `/etc/rc.nfs` script.
3. Then the `/etc/rc.nfs` script executes the **mount** command, which reads the file systems listed in the `/etc/filesystems` file.
4. The **mount** command locates one or more servers that export the information the client wants and sets up communication between itself and that server. This process is called *binding*.
5. The **mount** command then requests that one or more servers allow the client to access the directories in the client `/etc/filesystems` file.

6. The server daemon receives the client mount requests and either grants or denies them. If the requested directory is available to that client, the server daemon sends the client kernel an identifier called a *file handle*.
7. The client kernel then ties the file handle to the mount point (a directory) by recording certain information in a *mount record*.

Client communication with the **rpc.mountd** daemon does not occur with NFS version 4 mount processing. Operations in the core NFS version 4 protocol are used to service client side mount operations. The NFS version 4 server implementation does utilize support in the **rpc.mountd** daemon as part of handling NFS version 4 access.

/etc/exports file

The `/etc/exports` file indicates all directories that a server exports to its clients.

Each line in the file specifies a single directory. A directory can be specified twice in the `/etc/exports` file: once for NFS version 2 or NFS version 3, and once for NFS version 4. The server automatically exports the listed directories each time the NFS server is started. These exported directories can then be mounted by clients. The syntax of a line in the `/etc/exports` file is:

directory *-option*[,*option*]

The *directory* is the full path name of the directory. Options can designate a simple flag such as **ro** or a list of host names. See the specific documentation of the `/etc/exports` file in *Files Reference* and the **exportfs** command in *Commands Reference, Volume 2* for a complete list of options and their descriptions. The `/etc/rc.nfs` script does not start the **nfsd** daemons or the **rpc.mountd** daemon if the `/etc/exports` file does not exist.

The following example illustrates entries from an `/etc/exports` file:

```
/usr/games    -ro,access=ballet:jazz:tap
/home         -root=ballet,access=ballet
/var/tmp
/usr/lib      -access=clients
/accounts/database -vers=4,sec=krb5,access=accmachines,root=accmachine1
/tmp          -vers=3,ro
/tmp          -vers=4,sec=krb5,access=accmachines,root=accmachine1
```

The first entry in this example specifies that the `/usr/games` directory can be mounted by the systems named `ballet`, `jazz`, and `tap`. These systems can read data and run programs from the directory, but they cannot write in the directory.

The second entry in this example specifies that the `/home` directory can be mounted by the system `ballet` and that root access is allowed for the directory.

The third entry in this example specifies that any client can mount the `/var/tmp` directory. (Notice the absence of an access list.)

The fourth entry in this example specifies an access list designated by the netgroup `clients`. In other words, these machines designated as belonging to the netgroup `clients` can mount the `/usr/lib` directory from this server. (A *netgroup* is a network-wide group allowed access to certain network resources for security or organizational purposes. Netgroups are controlled by using NIS.

The fifth entry allows access to the directory `/accounts/database` only to clients in the `accmachines` netgroup using NFS version 4 protocol and accessing the directory using Kerberos 5 authentication. Root access is allowed only from `accmachine1`.

The sixth and the seventh entries export the `/tmp` directory using different versions and options. If two entries for the same directory with different NFS versions exist in the `/etc/exports` file, the **exportfs**

command will export both. If a directory has the same options for NFS version 4 and NFS version 3, you can have one entry in the /etc/exports file specifying -vers=3:4.

/etc/xtab file

The /etc/xtab file has a format similar to the /etc/exports file and lists the currently exported directories.

Whenever the **exportfs** command is run, the /etc/xtab file changes. This allows you to export a directory temporarily without having to change the /etc/exports file. If the temporarily exported directory is unexported, the directory is removed from the /etc/xtab file.

Note: The /etc/xtab file is updated automatically, and is not to be edited.

/etc/nfs/hostkey file

This file is used by the NFS server to specify the Kerberos host principal and the location of the keytab file.

For instructions on how to configure and administer this file, see the **nfshostkey** command description in *Commands Reference, Volume 4*.

/etc/nfs/local_domain file

This file contains the local NFS domain of the system.

It is implied that systems that share the same NFS local domain also share the same user and group registries. For instructions on how to configure and administer this file, see the **chnfsdom** command description in *Commands Reference, Volume 1*.

/etc/nfs/realm.map file

This file is used by the NFS registry daemon to map incoming Kerberos principals of the form *name@kerberos-realm* to the form *name@nfs-domain*.

It then can resolve the *name@nfs-domain* to a local UNIX credential. This file provides a simple way to map Kerberos principals into the server's user registry. It is appropriate when clients in different Kerberos realms will be accessing the server, but the user namespace is global. The file should contain lines in the following format:

```
realm1 nfs-domain
realm2 nfs-domain
```

for all Kerberos realms the server supports. If the Kerberos realm name is always the same as the server's NFS domain, this file is not needed. If you need the more general capability of mapping *userA@kerberos-realm* to *userB@nfs-domain*, use the Enterprise Identity Mapping (EIM) service. For additional information, see “Identity mapping” on page 502.

To add, edit, or remove entries in this file, use the **chnfsrtd** command. See the **chnfsrtd** command description in *Commands Reference, Volume 1* for more details.

/etc/nfs/princmap file

This file maps host names to Kerberos principals when the principal is not the fully qualified domain name of the server.

It consists of any number of lines of the following format:

```
<host part of principal> alias1 alias2 ...
```

To add, edit, or remove entries from this file, use the **nfshostmap** command. See the **nfshostmap** command description in *Commands Reference, Volume 4* for more information.

/etc/nfs/security_default file

The `/etc/nfs/security_default` file contains the list of security flavors that may be used by the NFS client, in the order in which they should be used.

Use the **chnfssec** command to manage this file. See the **chnfssec** command description in *Commands Reference, Volume 1* for more information.

Remote Procedure Call Protocol

NFS is implemented on a wide variety of machine types, operating systems, and network architectures. NFS achieves this independence using the **Remote Procedure Call (RPC)** protocol.

RPC is a library of procedures. The procedures allow one process (the client process) to direct another process (the server process) to run procedure calls as if the client process had run the calls in its own address space. Because the client and the server are two separate processes, they need not exist on the same physical system (although they can).

NFS is implemented as a set of **RPC** calls in which the server services certain types of calls made by the client. The client makes such calls based on the file system operations that are done by the client process. NFS, in this sense, is an RPC application.

Because the server and client processes can reside on two different physical systems which may have completely different architectures, **RPC** must address the possibility that the two systems might not represent data in the same way. For this reason, **RPC** uses data types defined by the **eXternal Data Representation (XDR)** protocol.

eXternal Data Representation Protocol

The **eXternal Data Representation (XDR)** protocol is the specification for a standard representation of various data types.

By using a standard data type representation, a program can be confident that it is interpreting data correctly, even if the source of the data is a machine with a completely different architecture.

In practice, most programs do not use **XDR** internally. Rather, they use the data type representation specific to the architecture of the computer on which the program is running. When the program needs to communicate with another program, it converts its data into **XDR** format before sending the data. Conversely, when it receives data, it converts the data from **XDR** format into its own specific data type representation.

portmap daemon

The **portmap** daemon helps clients map program number and version number pairs to the port number of a server.

Each RPC application has associated with it a program number and a version number. These numbers are used to communicate with a server application on a system. When making a request from a server, the client needs to know what port number the server is accepting requests on. This port number is associated with the **User Datagram Protocol (UDP)** or **Transmission Control Protocol (TCP)** that is being used by the service. The client knows the program number, the version number, and the system name or host name where the service resides. The client needs a way to map the program number and version number pair to the port number of the server application. This is done with the help of the **portmap** daemon.

The **portmap** daemon runs on the same system as the NFS application. When the server starts running, it registers with the **portmap** daemon. As a function of this registration, the server supplies its program number, version number, and **UDP** or **TCP** port number. The **portmap** daemon keeps a table of server applications. When the client tries to make a request of the server, it first contacts the **portmap** daemon to find out which port the server is using. The **portmap** daemon responds to the client with the port of the server that the client is requesting. Upon receipt of the port number, the client is able to make all of its future requests directly to the server application.

NFS applications and control

The NFS and NIS daemons are controlled by the System Resource Controller (SRC).

This means you must use SRC commands such as **startsrc**, **stopsrc**, and **lssrc** to start, stop, and check the status of the NFS and NIS daemons.

Some NFS daemons are not controlled by the SRC; specifically, **rpc.rexd**, **rpc.rusersd**, **rpc.rwalld**, and **rpc.rsprayd** are not controlled by the SRC. These daemons are started and stopped by the **inetd** daemon.

The following table lists the SRC-controlled daemons and their subsystem names.

Table 90. Daemons and their subsystems

File path	Subsystem name	Group name
/usr/sbin/nfsd	**nfsd**	nfs
/usr/sbin/biod	**biod**	nfs
/usr/sbin/rpc.lockd	**rpc.lockd**	nfs
/usr/sbin/rpc.statd	**rpc.statd**	nfs
/usr/sbin/rpc.mountd	**rpc.mountd**	nfs
/usr/sbin/nfsrgyd	**nfsrgyd**	nfs
/usr/sbin/gssd	**gssd**	nfs
/usr/lib/netsvc/yp/ypserv	**ypserv**	yp
/usr/lib/netsvc/yp/ypbind	**ypbind**	yp
/usr/lib/netsvc/rpc.yppasswdd	**yppasswdd**	yp
/usr/lib/netsvc/rpc.ypupdated	**ypupdated**	yp
/usr/sbin/keyserv	**keyserv**	keyserv
/usr/sbin/portmap	**portmap**	portmap

Related information:

System Resource Controller Overview

Changing the number of biod and nfsd daemons

The **chnfs** command can be used to change the maximum number of **biod** or **nfsd** daemons that will run on a system.

For example, to set the number of maximum **nfsd** daemons to 1000 and the maximum number of **biod** daemons to 4, run the following command:

```
chnfs -n 1000 -b 4
```

Note: This command will stop the currently running daemons, update the SRC configuration information, and then restart the daemons. As a result, NFS service will be temporarily unavailable. The maximum number of **biod** daemons may also be specified on a per mount basis using the `biods=`*n* mount option.

Note: If the number of **nfsd** daemons is not sufficient to serve the client, a nonidempotent operation error is returned to the client. For example, if the client removes a directory, an `ENOENT` error is returned even though the directory on the server is removed.

Changing command line arguments for daemons controlled by SRC

Many NFS and NIS daemons have command-line arguments that can be specified when the daemon is started. Because these daemons are not started directly from the command line, you must update the SRC database so that the daemons can be started correctly.

To do this, use the **chssys** command. The **chssys** command has the format:

```
chssys -s Daemon -a 'NewParameter'
```

For example:

```
chssys -s nfsd -a '10'
```

changes the **nfsd** subsystem so that when the daemon is started, the command line looks like `nfsd 10`. The changes made by the **chssys** command do not take effect until the subsystem has been stopped and restarted.

Starting the NFS daemons

The file size limit for files located on an NFS server is defined by the process environment when **nfsd** is started.

To use a specific value, edit the `/etc/rc.nfs` file. Use the **ulimit** command with the desired limit before the **startsrc** command for the **nfsd** daemon.

The NFS daemons can be started individually or all at once. To start NFS daemons individually, run:

```
startsrc -s Daemon
```

where *Daemon* is any one of the SRC-controlled daemons. For example, to start the **nfsd** daemons, run:

```
startsrc -s nfsd
```

To start all of the NFS daemons, run:

```
startsrc -g nfs
```

Note: If the `/etc/exports` file does not exist, the **nfsd** and the **rpc.mountd** daemons will not be started. You can create an empty `/etc/exports` file by running the `touch /etc/exports` command. This will allow the **nfsd** and the **rpc.mountd** daemons to start, although no file systems will be exported.

Stopping the NFS daemons

The NFS daemons can be stopped individually or all at once.

To stop NFS daemons individually, run:

```
stopsrc -s Daemon
```

where *Daemon* is anyone of the SRC-controlled daemons. For example, to stop the **rpc.lockd** daemon, run:

```
stopsrc -s rpc.lockd
```

To stop all NFS daemons at once, run:

```
stopsrc -g nfs
```

Getting the current status of the NFS daemons

You can get the current status of the NFS daemons individually or all at once.

To get the current status of the NFS daemons individually, run:

```
lssrc -s Daemon
```

where *Daemon* is any one of the SRC-controlled daemons. For example, to get the current status of the **rpc.lockd** daemon, run:

```
lssrc -s rpc.lockd
```

To get the current status of all NFS daemons at once, run:

```
lssrc -a
```

NFS version 4 support

Beginning with AIX 5.3, support for NFS version 4 protocol features is included.

The mandatory features of the protocol are supported as described in RFC 3530 with the following exceptions:

- The LIPKEY and SPKM-3 security mechanisms are not supported with RPCSEC-GSS RPC authentication. Only the Kerberos V5 mechanism is supported.
- The UTF-8 requirements are not fully supported. Specifically, the transmission of file names and file system strings such as symbolic link contents and directory entry names are not guaranteed to be in UTF-8 format. Transmission of NFS attribute strings, such as owner and owner group, are always in UTF-8 format. The NFS server and client do perform UTF-8 validation on incoming string data as defined RFC 3530. This checking can be administratively disabled using the **nfso** command. Disabling UTF-8 checking may be necessary to use NFS version 4 in environments with non UTF-8 configurations and data.
- Diskless client, NIM, and UDP are not supported over NFS version 4.

The following optional features of NFS version 4 are supported:

- NFS version 4 ACLs are supported by both the NFS client and server. The NFS client supports management of NFS version 4 ACLs using the **acledit**, **aclget**, and **aclput** utilities. The NFS server is capable of storing and retrieving NFS version 4 ACLs in underlying file systems that support the NFS version 4 ACL model. For more information, see “NFS Access Control Lists support” on page 480.
- Support is provided to map principals and file ownership attributes from one NFS version 4 domain into another. This support is primarily intended for use at AIX NFS servers. It requires deployment of LDAP. The NFS mappings are managed using the **chnfsim** utility.

There are several considerations when using concurrent access with NFS versions 2 and 3, and NFS version 4. NFS version 3 access may receive errors due to the NFS version 4 granted state. Also, NFS version 3 performance may be impacted when data is exported for NFS version 4 access.

NFS server grace period

The NFS version 4 (NFSv4) protocol provides functionality that lets system administrators enable a grace period on the NFSv4 server for special handling of specific operations.

Within this grace period, administrators can manage locking, read operations, and write operations for the entire duration of the server lease. Locks and their associated states can be recovered by clients through reclaim-type locking requests.

Note: Not all states reclaimed by clients in the grace period can be guaranteed to be the state held by the server in the previous instance. The state that is reclaimed during the grace period is guaranteed to be the correct one as defined by the NFSv4 RFC.

Beginning with AIX 5L Version 5.3 with the 5300-05 Technology Level, administrators can use the grace period on NFSv4 servers. The grace period is disabled by default. To enable the grace period on the server, use the SMIT menu or the **chnfs** command line interface.

When the grace period is enabled, the NFSv4 server records state information to disk in the /var file. The recorded state is reclaimed automatically when the server restarts.

NFS DIO and CIO support

AIX 5L Version 5.3 with the 5300-03 Recommended Maintenance package supports direct I/O and concurrent I/O in the NFS client for the version 3 and 4 protocols. DIO and CIO only involve the client.

Using DIO and CIO, data center workloads such as databases and high performance computing applications can experience higher levels of performance, coupled with reduced system CPU and memory resources, while maintaining the benefits of centralizing file-based storage and its associated management of back-end systems.

I/O is often nonsequential, and often applications do not benefit from data caching at the NFS client, or the applications do all the advanced caching. These applications benefit when NFS does not cache, perform read prediction, or use write-behind mechanisms. In addition, some applications such as databases are not dependent on POSIX single-site semantics, which serialize reads against writes. These applications issue concurrent reads and writes, but these applications are responsible for the consistency and coordination of such operations.

Direct I/O for NFS

DIO allows applications to perform reads and writes directly to the NFS server without going through the NFS client caching layer (Virtual Memory Manager), or incurring the associated overhead of caching data.

Under DIO, application I/O requests are serviced using direct Remote Procedure Calls (RPC) to the NFS server. You can set DIO using the AIX *dio* mount option. Without the mount option, you can also enable DIO per-file by using the **AIX O_DIRECT open()** flag.

Servicing NFS direct I/Os might require multiple RPCs to the server, depending on the I/O request size and the maximum wire transfer size permitted by the server and client. For more information about DIO, see the **-o** option for the **mount** command.

Concurrent I/O for NFS

With CIO, application reads and writes that are issued concurrently run concurrently without reads blocking for the duration of writes, or the reverse.

Multiple writes also run concurrently. POSIX atomicity guarantees are not provided. When CIO is in effect, direct I/O is implied. Use the AIX *cio* mount option or the **O_CIO open()** flag to set CIO. For more information about CIO, see the **-o** option for the **mount** command.

In AIX Version 6.1 with the 6100-04 Technology Level and later, you can run the **mount** command, the **nfs4cl** command, or the **open()** subroutine to allow you to open read only files when those file have already been opened in CIOR. The **cior** mount option and the **O_CIOR open ()** flag can only be used in conjunction with CIO.

Related information:

mount command

Interaction of DIO, CIO, regular opens, and mapped files for NFS

The following behaviors exist between the different access modes that can occur with DIO and CIO.

When existing DIO opens are in effect:

- A regular open causes DIO to be turned off until there are no longer regular opens. When a close reduces the regular opens to 0, DIO is re-enabled if there are still DIO opens outstanding.
- Mapping a file with shmat() or mmap() will deactivate DIO on the file until the number of mappings drops to 0. Then if there are still DIO opens, DIO will be re-enabled.

- Attempts to open the file for CIO will fail with the **EINVAL** error.

When there are regular opens (no CIO or DIO) in effect:
- DIO open attempts succeed, but DIO is not activated until the count of regular opens drops to 0.
- Opens for CIO will fail with the **EINVAL** error.

When CIO opens are in effect:
- Regular, DIO, and attempts to map the file will all fail with the **EINVAL** error.

When CIO|CIOR opens are in effect:
- Regular, DIO, and attempts to map the file will all fail with the **EINVAL** error except Read-only and CIO|CIOR opens.

Note: When there is a transition to DIO or CIO, client-cached modifications will first write back to the NFS server prior to deleting all cached information.

NFS replication and global namespace

The NFS version 4 (NFSv4) protocol provides functions that allows you, as the system administrator, to distribute data across multiple servers in a way that is transparent to the users of that data.

You can use two features provided beginning with AIX 5L Version 5.3 with the 5300-03 Recommended Maintenance package. The first is a global namespace feature called a *referral*. The second feature is a means of specifying locations where copies of data can be found, which is called a *replica*.

A *referral* is a special object you can create in the namespace of a server to which location information is attached. The server uses NFSv4 protocol features to redirect clients to the server specified in the location information. The referral forms a building block for integrating data on multiple NFS servers into a single file namespace tree that referral-aware NFSv4 clients can navigate.

A *replica* is a copy of a file system on one NFS server that has been placed on several other NFS servers (or at an alternate location, such as a different disk on the same server). If a given replica location in use by a replica-aware NFSv4 client becomes unavailable, then the client will switch to another available replica. For more information about replicas, refer to "NFS replicas" on page 494.

NFS referrals

The following examples provide scenarios to help you understand referrals.

In the following examples, there are four servers:
- The server called `publications` contains documentation files.
- The server called `projects` contains user work directories.
- The server called `data` contains information databases.
- The server called `account1` is the main NFS server that exports all other files, and is the server that all the clients know about.

Allowing all clients to access files on the main NFS server

Server `account1` exports the directory `/work` to all clients using the following statement in the `/etc/exports` file:

```
/work -vers=4
```

All clients can access the files in the `/work` remote directory by mounting / from the `account1` server onto the `/mnt` directory using the following command:

```
mount -o vers=4 account1:/ /mnt
```

When the user on the client lists the contents of the /mnt directory, they see the remote directory work at the path /mnt/work. The contents of the /mnt/work directory on the client is the same as the contents of the /work directory on the account1 server.

Allowing a client to access files on a specific server

The client user also wants to access the /usr/doc directory on the publications server.

In previous releases, you must export the directory from the server and mount the directory at the client.

Using referrals to build a distributed namespace

You can set up a server so that clients can access the data on other servers without the client knowing where the data is. Only the administrator on the referring server needs to know where the data is. The referring server can redirect clients to the location of the /usr/doc directory using a referral. On the server publications, the /usr/doc directory can be exported by adding the following statement to the export file:

```
/usr/doc -vers=4
```

This makes the directories available to NFSv4 clients.

The account1 server can now use referrals to make those directories available to clients by adding the following statement to the export file:

```
/usr/doc -vers=4,refer=/usr/doc@publications
```

You then export the directory. At this point, the client that mounted the /mnt directory from the / directory on the account1 server has access to the directory usr when the client lists the /mnt directory. The client does not have to perform any mounts to other servers. The client user does not even need to be aware that the files there are not being provided by the account1 server. For example, you can make the directories of /databases/db on server data and /home/accts on server projects available through account1 by exporting the directories from the data and projects servers, and creating referrals on account1 to those directories.

Because a client user is unaware of the actual location of the data, the administrator can redirect clients from one server to another simply by changing the referral statement in the exports file on the server. The administrator is responsible for the placement and correctness of data that referrals reference with their location specifications.

Administrators should make sure the second server will not refer the request back to the first server, creating a circular referral. In the above example, if the administrator had created a referral on the publications server at /usr/doc that referred to /usr/doc on the account1 server, the resulting circular referral would be undesirable.

Although referrals are created using exportfs, they are different from data exports. Locations specified for referrals should correspond to the root directories of NFSv4-exported file systems. You can create a referral within exported namespaces or in unexported namespaces. In the above example, the /usr/doc referral can be created on the account1 server even if /usr is not exported. This places the referral within the NFSv4 pseudospace. If account1 had exported /usr, the referral export would still have been allowed, in contrast to exporting a directory called doc, which would have failed if it were in the same file system. In either case, the referral export would have failed if a file or directory had existed at /usr/doc. There is no restriction on the number of referrals that can be created within either the server's NFSv4 pseudospace or within an exported file system.

Since a referral does not export any data and only has meaning to the NFSv4 protocol, referrals are available in NSFv4 only. Exporting a referral without the vers=4 option will fail. Although this example only specifies one location, up to 8 locations could have been specified.

Creating a referral creates a special referral object at the location specified by the directory parameter. Since client access to the object is determined by the client's access to the object's parent directory, most other export options have no meaning and are allowed and are ignored. The only exception is the `exname` option, which will have the expected behavior. For example, if the server creates the referral `/n4root/special/users -vers=4,exname=/exported/users,refer=/restricted/users@secrethost`, the clients that mount / from that server would see the path `/mnt/exported/users`, which would redirect the clients to the `/restricted/users` directory on `secrethost`. On the exporting server, the referral object would actually be created in the local namespace at `/n4root/special/users`, so no file or directory can exist there when the export is done. A special object gets created on the server to hold the referral location information. Any directories along the path to the referral will also be created if they do not exist. If the referral is unexported, the referral information will be removed from the object, but the object itself will not be removed. The NSFv4 server will not allow clients to access the resulting *stale* or *orphan* referral object. It will return an access error to clients that attempt to access the object. The object can be removed using **rm**, if desired. A referral can be re-exported with new referral information. This is not recommended as a frequent practice because it can take some time for clients that have accessed the referral to realize that the location information has changed. The server does touch the referral's parent directory to indicate that information in the directory has been changed. This helps the clients to realize that any information that the client has cached about the directory (and the referral within the directory) has changed and needs to be refetched, but there is no guarantee how long it will take for the clients to notice.

For information about using the **refer** option to change the order of the locations specified in the file system location list, see Reordering the file system location list using the **scatter** option.

NFS replicas

Replication allows you, as the NFSv4 administrator, to place copies of data on multiple NFSv4 servers and inform NFSv4 clients where the replicas reside.

In the event that the primary data server becomes inaccessible to the clients, the clients can use one of the replica servers to continue operations on the replicated file system. The replica file systems are assumed to be exact copies of the data on the primary server. You can set up to 8 replica locations. The AIX server does not specify how the replica file systems are created from the primary file system or how the data is kept coherent. If you are going to specify replicas as read-write, you must keep the data on the replicas coherent with the primary file system.

A replica is a server that contains a copy of the directory or directories of another server. If the primary server becomes unavailable to the client, the client can access the same files from a replica location. The following scenario is an example:

If the files in the `/data` directory on the `account1` server are also available in the `/backup/data` directory on the server `inreserve`, the NSFv4 clients could be made aware of this by specifying replica locations on the export. By adding a statement similar to the following one in the export file, you can export the `/data` directory and specify the location of the replica copy:

```
/data -vers=4,replicas=/data@account1:/backup/data@inreserve
```

If the `account1` server becomes unavailable, client users using files under the `/data` directory of the `account1` server can begin using files in the `/backup/data` directory on the `inreserve` server, without being aware that the client has switched to a different server.

For information about using the **replicas** option to change the order of the locations specified in the file system location list, see Reordering the file system location list using the **scatter** option.

NFS configuration requirements to allow specification of replicas:

You must be an administrator to enable, disable, or specify root replicas.

To enable, disable, and specify root replicas, use the following command:

```
chnfs -R {on|off|host[+host]}
```

In order to specify replicas, the server must be configured with **chnfs -R** (**chnfs -R** on) to issue volatile NFSv4 file handles. A file handle is an identifier that NFS servers issue to clients to identify a file or directory on the server. By default, the server issues persistent file handles. Switching between file handle types can result in errors to applications at NFSv4 clients that are actively using the server when the switch is performed. In order to change the file handle mode with **chnfs -R**, no file systems can be exported for NFSv4 access. Setting the file handle disposition should be done with a newly-provisioned NFS server or done when NFS activity can be minimized or stopped. For clients actively connected to servers when the mode is changed, it may be necessary to unmount and remount NFSv4 mounts on those clients. To minimize this action, the number of client mounts can be reduced to a small number of mounts that mount the top-level directories of an NFSv4 server's exported file space.

The NFSv4 client cannot fail over to replicas with different export access properties. Administrators must make sure that all replicas are specified with the same export access controls and access mode (read-only or read-write). With the possible exception of exported GPFS™, it is expected that replicated data will be exported read-only. It is also the administrator's responsibility to maintain the data content at all replica locations. Directory trees and all data content should be kept identical. Updates to data content will need to be performed in a manner that is most compatible with the applications that will be using the data.

With replicas, you can use the **exname** export option to hide details of the server's local file system namespace from NFSv4 clients. For more details, see the **exportfs** command description in *Commands Reference, Volume 2* and the `/etc/exports` file description in *Files Reference*.

You can use the **replicas** option with the export cluster file systems such as General Parallel File System (GPFS) to specify multiple NFS server nodes that see the same GPFS view. This is a configuration where exporting the data for read-write access may be valid. However, with read-write replicas, if a replica failover occurs while write operations are in progress, applications performing the write may encounter unrecoverable errors. Similarly, a **mkdir** or exclusive file create operation running during a failover may encounter an **EXISTS** error.

A replicated export must export an entire file system. This means that the directory being exported must be the root of the local file system. The server exporting a replicated file system should specify itself as one of the locations for the export. For servers with multiple interfaces, this must include the server's primary host name. If the server exporting a replicated file system does not specify itself as one of the locations for export, the exporting server will be silently added to the list of replica locations as the first replica location. The order of replica locations in the replica list specifies the order of preference the clients should use when failing over. For example, if the user at `serverA` wants to export `/webpages`, and there is a replica of `/webpages` on `serverB` in the `/backup/webpages` directory, the following entry in the `/etc/exports` file will export `/webpages` from `serverA` and inform clients that there is a copy of the file system on `serverB` at `/backup/webpages`:

```
/webpages -vers=4,ro,replicas=/webpages@serverA:
/backup/webpages@serverB
```

Both `/webpages` on `serverA` and `/backup/webpages` on `serverB` are assumed to be the root directories of their file systems. If `serverA` had not been listed in the export, it would have been silently added as the first replica location. This is because the server exporting the data is assumed to be the preferred server for the data it is exporting.

Replicas are only used by the NFSv4 protocol. The above export could have specified NFSv3 (vers=3:4), but the replication information would not be available to NFSv3 clients. Clients using NFSv3 can, however, access the information in /webpages on serverA, but they will not fail over to the replica if serverA becomes unavailable.

NFS client-side support for multiple locations:

When the client can no longer access replicated data from its current server, the client attempts to access the data from the next-most-favored server.

The order in which the replicas are specified in the replicas list is taken by the client to be the order of preference.

The client administrator can override the replica preference using the **prefer** subcommand of the **nfs4cl** command. The **nfs4cl** command displays all the file system information on the client or modifies file system options of a file system and displays or modifies current NFSv4 statistics and properties.

NFS common considerations for replicas and referrals:

If the client encounters two different paths leading to the same data (file system), the client treats the second path as a symbolic link to the file.

For example, server A exports:

```
/tmp/a              -vers=4,replicas=/tmp/a@B:/tmp/a@A
/tmp/b              -vers=4,refer=/tmp/a/b@B
```

And server B exports:

```
/tmp/a              -vers=4
/tmp/a/b            -vers=4
```

In this example, the client mounts / on server A onto /mnt by using the command mount -o vers=4 A:/ /mnt. The client user accesses /tmp/a/b on server B through cd /mnt/tmp/a/b or cd /mnt/tmp/b. If the user changes the directory to cd /mnt/tmp/a/b first, then the path /mnt/tmp/b acts as a symbolic link to /mnt/tmp/a/b. In this scenario, if the user is in /mnt/tmp/b and uses the command /bin/pwd, /bin/pwd > will return /mnt/tmp/a/b.

Note: The above practice is not recommended. The administrator should set up export specifications that result in only one possible namespace path to exported data.

You can list multiple locations in referrals if the target data of the referral is actually replicated. Clients will only use the referral locations to find the referral target at an available server. Once the client establishes access to the referral target, it will obtain new location information for the found data.

Since clients may not immediately detect changes on referral location information, removing or changing referral location frequently is not recommended. When relocating the target of a reference location, it is suggested that the new location be populated along with changing the location information in the export's referral specification. The data at the old location should be kept for several hours or even days to give clients time to see and use the new location.

Both replication and referrals can only run on servers running the 64-bit kernel. Clients can run on both the 32- and 64-bit kernels.

If you are going to specify replicas as read-write, you must keep the data on the replicas coherent with the primary fileset.

How NFS clients fail over:

A *fail-over* is when the client switches from one replica location to another after it determines that the current server it is communicating with is no longer accessible.

The following tunables influence the NFS client fail-over behavior:

NFS mount option *timeo*
: This mount option specifies the time that the TCP/IP layer must wait before it returns with a timeout response.

NFS mount option *retrans*
: This mount option specifies the number of times the NFS RPC layer should retry the client's request before returning an RPC timeout error (**ETIMEDOUT**).

nfso option *nfs_v4_fail_over_timeout*
: You can use this **nfso** option to specify the minimum amount of time the client must wait before failing over to a replica. This option is global to the NFS client and overrides the default per mount behavior. By default, the **nfs_v4_fail_over_timeout** is not active. It's value is `0`.

When **nfs_v4_fail_over_timeout** is not active, the fail-over threshold is set to twice the mount **timeo** option value. When no successful RPC calls have occurred for this duration, the client will begin fail-over processing to find another available replica. However, the actual time the client will wait is influenced by the **retrans** option. If **retrans** is greater than 2, the client will likely wait until it receives an RPC timeout based on the **retrans** value times the **timeo** value (`retrans` × `timeo`). Therefore, the combination of the **timeo** and **retrans** options can be adjusted to control fail-over behavior on a per-NFS mount basis. You can also set these options at a more granular level by using the **nfs4cl** command.

When **nfso nfs_v4_fail_over_timeout** is set to a non-zero value, it represents the number of seconds the client will wait on an unavailable server before considering replica fail-over. If the **timeo** and **retrans** options result in RPC timeout behavior beyond the **nfso** setting, fail-over processing may not start until the RPC timeout is generated.

For more information about the **retrans**, **timeo**, and **nfs_v4_fail_over_timeout** options, refer to the NFS-specific options of the **mount**, **nfs4cl**, and **nfso** commands.

In addition to replica fail-over in the event of an unavailable server, there are cases where the client will voluntarily switch from one replica location to another. One case is when you use the **nfs4cl** command to establish a preferred replica. In this case, the client initiates a switch to the preferred server, if that is not the current server the client is using. The client will also refetch replica location information from the NFS server on approximately 30 minute intervals when there has been recent activity on the associated data. If the ordering of the locations has changed, the client attempts to switch to the first location, if that is different from the current server the client is using and you have not set a replica preference with the **nfs4cl** command.

Soft NFS mounts versus failover behavior:

The default mount model for NFS is hard mounts and the replica failover behavior applies to hard mounts. Failover behavior is different if NFS soft mounts are used.

If the soft mount settings result in an RPC timeout prior to the established wait period for replica failover, then the timeout will result in an **ETIMEDOUT** error to the calling application. The use of soft mounts with replicated data is not recommended. If soft mounts are used and the **nfso nfs_v4_fail_over_timeout** value has been set, it is suggested that the **retrans** and **timeo** mount options be set to exceed the **nfso** setting. This will avoid **ETIMEDOUT** returns to applications for replicated data.

Reordering the file system location list using the scatter option

The **scatter** option of the **exportfs** command enables you to change the order of the locations specified in the file system location list that is set with either the **refer** option or the **replicas** option of the **exportfs** command.

Using this option generates different combinations of server locations so that different lists have different servers in the order of preference. Consequently, different clients have server location lists that are different. This reordering helps load balancing because the first server in the location list of different clients is a different server. Also, if a server goes down, the fail-over load is distributed across multiple servers because the server in the next location in the server locations list is different. The **scatter** option applies only to directories exported for access by the NFS version 4 protocol.

The **scatter** option can have the following values:

- **full** - All of the servers are reordered to form combinations of alternate locations. The total number of combinations is limited to either 12 or the number of servers, whichever number is higher.
- **partial** - The first location for all of the server combinations generated is fixed to the first server in the server list. The rest of the locations are listed as if a full reordering is used.
- **none** - No reordering of the file system location list is done. This is the default value for the scatter **option**. Use this value to disable any previous reordering of the location list.

Note: If the **noauto** flag is not specified when you are using the **exportfs** command, then the location list includes the primary host name as one of the replica locations. For more information about the **noauto** flag, see the **exportfs** command in *Commands Reference, Volume 2*.

To specify referrals for the `/common/documents` directory at hosts s1, s2, and s3, and then reorder them using the **full** option, add the following line to the `/etc/exports` file and then export the `/common/documents` directory:

```
/common/documents -ver=4, refer=/common/documents@s1:/common/document@s2a:/common/
documents@s3,scatter=full
```

To specify replicas for the `/common/documents` directory at hosts s1, s2, s3, and s4, and reorder them partially (first fail-over server is s1 for all combinations), add the following line to the `/etc/exports` file and then export the `/common/documents` directory:

```
/common/documents -vers=4, replicas=/common/documents@s1:/common/documents@s2:/common/
   documents@s3:/common/documents@s4,scatter=partial
```

NFS server-client delegation

Delegation is the ability of the server to delegate certain responsibilities to the client.

Beginning with AIX 5L Version 5.3 with the 5300-03 Recommended Maintenance package, you can use delegation. When the server grants a delegation for a file to a client, the client is guaranteed certain semantics with respect to sharing that file with other clients. When a file is opened, the server can provide the client a read delegation for the file. If the client is granted a read delegation, it is assured that no other client has the ability to write to the file for the duration of the delegation. If the client is granted a write delegation, the client is assured that no other client has read or write access to the file. The AIX server only grants read delegations. The AIX server only supports delegation with the 64-bit AIX kernel. The AIX client supports both read and write delegations.

In order for the server to grant a delegation to the client, the client must first provide a callback address to the server. When a delegation is recalled, the server will send the recall request to this address. By default, the client will indicate the IP address that is being used for normal communication with the server. For clients with multiple network interfaces, a specific address can be specified in the `/etc/nfs/nfs4_callback.conf` file. The format of the entries in this file is:

server-host client-ip-address

Where *server-host* is the name or address of an NFSv4 server and *client-ip-address* is the client address to be used when providing the server callback information. If the *server-host* name is the IPv4 address 0.0.0.0 or the IPv6 address 0::0, the specified *client-ip-address* will be used for all servers that are not listed in the file. If this file does not exist, or if an entry for the server (or a default entry) is not found, the client selects an address based on the existing connection to the server.

Delegations can be recalled by the server. If another client requests access to the file in such a way that the access conflicts with the granted delegation, the server is able to notify the initial client and recall the delegation. This requires that a callback path exists between the server and client. If this callback path does not exist, then delegations cannot be granted. If a file delegation has been granted, access from other NFSv4 clients, NFS versions 2 and 3 clients, and local accesses to the file at the file server can cause the delegation to be recalled. If GPFS is being NFSv4 exported, an access at a GPFS node in the network may cause the delegation to be recalled.

The essence of a delegation is that it allows the client to locally service operations such as OPEN, CLOSE, LOCK, LOCKU, READ, and WRITE without immediate interaction with the server.

Server delegation is enabled by default. Server delegation can be disabled with the `nfso -o server_delegation=0` command. Administrators can use the **exportfs deleg=yes | no** option to disable or enable the granting of delegations on a per-file system basis, which will override the **nfso** setting.

Client delegation can be disabled with the `nfso -o client_delegation=0` command. Client delegation must be set before any mounts take place on the client.

If the administrator is exporting a file system where many clients will be writing to many common files, the administrator may want to disable delegations for that file system.

If the client cannot be contacted (for example, if the network or client is experiencing an outage) other clients may be delayed in accessing the data.

Establishing generic host principals for Kerberos-protected callback paths

You can set up a callback path for IBM Network Authentication Service (Kerberos).

The client receiving the delegation must be a full client with its own host principal. However, you can establish a generic host principal for all clients to use for callbacks.

To establish a generic host principal for all clients to use for callbacks, perform these steps:

1. To create a service principal (for example, `nfs/client`) using the same method used to create a host principal, refer to Creating a Kerberos principal in *Security*.
2. Create a keytab entry for that service principal. For example, to create a keytab called slapd_krb5.keytab, do the following:

```
kadmin.local: ktadd -k /etc/security/slapd_krb5.keytab ldap/plankton.austin.ibm.com
Entry for principal ldap/plankton.austin.ibm.com with kvno 2,
encryption type Triple DES cbc mode with HMAC/sha1 added to keytab
WRFILE:/etc/security/slapd_krb5.keytab.
Entry for principal ldap/plankton.austin.ibm.com with kvno 2,
encryption type ArcFour with HMAC/md5 added to keytab WRFILE:/etc/security/slapd_krb5.keytab.
Entry for principal ldap/plankton.austin.ibm.com with kvno 2,
encryption type AES-256 CTS mode with 96-bit SHA-1 HMAC added to keytab
WRFILE:/etc/security/slapd_krb5.keytab.
Entry for principal ldap/plankton.austin.ibm.com with kvno 2,
encryption type DES cbc mode with RSA-MD5 added to keytab WRFILE:/etc/security/slapd_krb5.keytab.
kadmin.local:
```

3. Distribute this keytab to all clients that will use it.
4. Configure the clients with the **nfshostkey** command.

This process is identical to the process for configuring a server for use with Kerberos, but the generic principal cannot be used for servers; each server must have its own principal of the form `nfs/hostname`.

STNFS short-term network file systems

An Short Term Network File Systems (STNFS) file system is a file system backed up by a Network File System (NFS), and the STNFS file system allows local modification to files. The modifications are not saved on the server.

Notes:

1 Many STNFS clients can share the same file system image from a server, but any modifications are seen only by the client making the modification.

2 Any modifications made by a client are lost when the file system is unmounted or when the client reboots.

3 Write operations from STNFS fail when the system memory falls below the predetermined threshold. This threshold is internal to STNFS and cannot be configured externally.

Mounting a NFS short-term file system

The **mount** command is used to mount an NFS file system on a short-term basis. For example, type the following command:

```
mount -v stnfs -o options server:/remote-path /local-path
```

The available options are:

vers=3 Use NFS version 3 to communicate with the server.

vers=4 Use NFS version 4 to communicate with the server.

rsize=size
Set the read-size bytes.

proto=udp
Use UDP to communicate with the NFS server.

proto=tcp
Use TCP to communicate with the NFS server.

hard Use NFS hard mounts.

soft Use NFS soft mounts.

sec Use the specified security classification.

The default options are:

vers=3

rsize=32768

proto=tcp

hard

sec=sys

Checklist for configuring NFS

After the NFS software is installed on your systems, you are ready to configure NFS. Follow these steps to configure NFS.

The CryptoLite in C (CLiC) kernel library must be installed before configuring NFS to use the following types of security:

- krb5

- krb5i
- krb5p

Each step is described in greater detail below.

1. Determine which systems in the network are to be servers and which are to be clients (a system can be configured as both a server and a client).
2. Determine what version of NFS you are going to use.
3. Decide whether you are going to use RPCSEC-GSS security. If so, refer to considerations in "Setting up a network for RPCSEC-GSS" on page 504.
4. For each system (whether client or server), follow the instructions in "Start the NFS daemons at system startup."
5. For each NFS server, follow the instructions in "Configuring an NFS server."
6. For each NFS client, follow the instructions in "Configuring an NFS client."
7. If you want personal computers on your network to have access to your NFS servers (beyond being able to mount file systems), configure PC-NFS by following the instructions in "PC-NFS" on page 514.
8. If NFS version 4 use is planned, refer to considerations in "NFS version 4 support" on page 490.

Start the NFS daemons at system startup

The NFS daemons, by default, are not started during installation.

When installed, all of the files are placed on the system, but the steps to activate NFS are not taken. You can start the NFS daemons at system startup through:

- The SMIT fast path, `smit mknfs`
- The **mknfs** command.

All of these methods place an entry in the `inittab` file so that the `/etc/rc.nfs` script is run each time the system restarts. This script, in turn, starts all NFS daemons required for a particular system.

Configuring an NFS server

Use this procedure to configure an NFS server.

To configure an NFS server:

1. Create the `/etc/exports` file. See "/etc/exports file" on page 485.
2. If you are using Kerberos, set up the NFS server as a Kerberos client. See "Setting up a network for RPCSEC-GSS" on page 504.
3. If you are using NFS version 4, establish the NFS version 4 domain using the **chnfsdom** command. See the **chnfsdom** command description in *Commands Reference, Volume 1* for details.

   Initially, you can specify the server's internet domain in the file. It is possible, however, to define an NFS version 4 domain that is different from the server's internet domain. For clarification on this, see the documentation for the NFS registry daemon, **nfsrgyd** in *Commands Reference, Volume 4*.
4. If you are using NFS version 4 with Kerberos, you may need to create the `/etc/nfs/realm.map` file. See "/etc/nfs/realm.map file" on page 486.
5. If you want to use Kerberos authentication on the server, you must enable enhanced security on the server. You can enable enhanced security through using SMIT, or by using the **chnfs -S -B** command. For more information on **chnfs**, refer to the **chnfs** command description in *Commands Reference, Volume 1*.

Configuring an NFS client

Use this procedure to configure an NFS client.

1. Start NFS using the instructions in "Starting the NFS daemons" on page 489.

2. Establish the local mount point using the **mkdir** command. For NFS to complete a mount successfully, a directory that acts as the mount point (or place holder) of an NFS mount must be present. This directory should be empty. This mount point can be created like any other directory, and no special attributes are needed.

   Note: With one exception, the mount points for all NFS mounts must exist on your system before mounting a file system. If the **automount** daemon is used, it is not necessary to create mount points. See the **automount** daemon description in *Commands Reference, Volume 1* for details.
3. If using Kerberos, follow these steps:
   a. Configure the NFS client into a Kerberos realm. This is done with the **config.krb5** command. Refer to the *IBM Network Authentication Service Administrator's and User's Guide* for configuration details.
   b. Create Kerberos principals for all users on the client who will be accessing files via Kerberos mounts. This is done with the **kadmin** command. Refer to the *Network Authentication Service Administrator's and User's Guide* for a description of how to create Kerberos principals.
   c. Establishing a Kerberos principal for the client machine itself is optional. A client without a principal is known as a *slim client*, and a client with a principal is referred to as a *full client*. Slim clients use weaker NFS RPC security when performing certain NFS version 4 client-to-server context management operations used for state management. A full client, dependent on configuration, can use the stronger Kerberos-based RPC security. Slim client configurations require less administrative overhead and may be sufficient for many environments. Deployments requiring the highest levels of security may choose to run full client configurations.
4. If you are using NFS version 4, you must also establish an NFS version 4 domain using the **chnfsdom** command. Initially, you can specify the client's internet domain in the file. It is possible, however, to define an NFS version 4 domain that is different from the client's internet domain. For clarification on this, see the documentation for the NFS registry daemon, **nfsrgyd**.
5. If you want to use Kerberos authentication on the client, you must enable enhanced security on the client. You can enable enhanced security using SMIT, or by using the **chnfs -S -B** command. For more information on **chnfs**, refer to the **chnfs** command reference page.
6. Establish and mount the predefined mounts by following the instructions in "Establishing predefined NFS mounts" on page 510.

Identity mapping

Identity mapping provides a method for the local NFS server and client to translate foreign users and groups to local users and groups.

AIX uses the EIM technology, which is based on LDAP, to perform its identity mapping. All NFS identity mapping data is stored on an LDAP server.

In order to set up an EIM client, the `bos.eim.rte` and `ldap.client` filesets must be installed. The EIM server also requires the `ldap.server` fileset. After the appropriate filesets are installed, the `/usr/sbin/chnfsim` is used to configure EIM. The minimum setup options are as follows:

```
/usr/sbin/chnfsim -c -a -t [type] -h [EIM server] -e [LDAP/EIM domain] -f [LDAP suffix] -w [administrator password]
```

This configures both EIM clients and servers to use a specific EIM server for identity mapping. If the host name specified in the command is the local host name, then an LDAP server will also be setup.

After the configuration step is complete, the EIM administrator can populate the LDAP server with NFS identity mapping data. An individual user or group, such as John Doe, is known as a mapping identity. The NFS owner string of that user, *johndoe@austin.ibm.com*, is known as an identity mapping. To input the LDAP server with this data, the following command should be run:

```
/usr/sbin/chnfsim -a -u -i "John Doe" -n johndoe -d austin.ibm.com
```

The mapping identity is the descriptive name of the user or group, and the identity mapping is the *name@domain* NFS owner string. Realm to domain mappings are also stored in the LDAP server. To input that the Kerberos realm `kerb.austin.ibm.com` maps to the NFS domain `austin.ibm.com`, the following command should be run:

```
/usr/sbin/chnfsim -a -r kerb.austin.ibm.com -d austin.ibm.com
```

In order to configure NFS to use the mapping data in EIM, the NFS registry daemon needs to be restarted. The NFS registry daemon checks for the availability of an EIM server upon startup, and if one is found, all mapping functions will go through EIM and all local mappings will no longer be used.

For information on EIM, see Enterprise identity mapping in the *Security*.

Exporting an NFS file system

You can export an NFS file system using the following procedures.

- To export an NFS file system using the SMIT:
  1. Verify that NFS is already running by typing the command `lssrc -g nfs`. The output should indicate that the **nfsd** and the **rpc.mountd** daemons are active. If they are not, start NFS using the instructions in "Starting the NFS daemons" on page 489.
  2. At a command line, type the following and press Enter:
     ```
     smit mknfsexp
     ```
  3. Specify appropriate values in the PATHNAME of directory to export, MODE to export directory, and EXPORT directory now, system restart or both fields.
  4. Specify any other optional characteristics you want, or accept the default values by leaving the remaining fields as they are.
  5. When you have finished making your changes, SMIT updates the `/etc/exports` file. If the `/etc/exports` file does not exist, it is created.
  6. Repeat steps 3 through 5 for each directory you want to export.
- To export an NFS file system using a text editor:
  1. Open the `/etc/exports` file with your favorite text editor.
  2. Create an entry for each directory to be exported using the full path name of the directory. List each directory to be exported starting in the left margin. No directory should include any other directory that is already exported. See the `/etc/exports` file in *Files Reference* for a description of the full syntax for entries in the `/etc/exports` file.
  3. Save and close the`/etc/exports` file.
  4. If NFS is running, type the following command and press Enter:
     ```
     /usr/sbin/exportfs -a
     ```

     The **-a** option tells the **exportfs** command to send all information in the `/etc/exports` file to the kernel. If NFS is not running, start NFS using the instructions in "Starting the NFS daemons" on page 489.
- To temporarily export an NFS file system (without changing the `/etc/exports` file), type the following command and press Enter:

  `exportfs -i` *`/dirname`*

  where *dirname* is the name of the file system you want to export. The **exportfs -i** command specifies that the `/etc/exports` file is not to be checked for the specified directory, and all options are taken directly from the command line.

  The AIX NFS version 4 support allows the administrator to create and control an alternate namespace that is rendered by the NFS server to clients. This is done using the **exname** export option. This support can also be used to hide details of the server's local file system namespace from NFS clients. For more details, see the **exportfs** command description in *Commands Reference, Volume 2* and the `/etc/exports` file description in *Files Reference*.

Setting up a network for RPCSEC-GSS

The network that is being set up in this scenario contains five servers and is configured for RPCSEC-GSS.

The five servers on the network are as follows:

- kdc.austin.ibm.com
- alpha.austin.ibm.com
- beta.austin.ibm.com
- gamma.austin.ibm.com
- zeta.austin.ibm.com

The system kdc.austin.ibm.com will be configured as the Key Distribution Center (KDC) server, and the Kerberos realm AUSTIN.IBM.COM will be created, on which all of the systems except kdc.austin.ibm.com and zeta.austin.ibm.com will be NFS servers offering file systems exported with RPCSEC-GSS.

Systems alpha.austin.ibm.com and beta.austin.ibm.com have an additional link between them; across that link, they appear to each other as fast_alpha.test.austin.com and fast_beta.test.austin.ibm.com. For this reason, an additional configuration step will be required.

In addition, this network has the following users, which have been configured on some of the systems:

- adam
- brian
- charlie
- dave
- eric

Note: The following setup is provided only as an example, and may not be appropriate for all environments. See the Administrator's and User's Guide for the Network Authentication Service before attempting to set up a new Kerberos realm.

Note: Kerberos requires that the system time be reasonably close throughout the network. Before beginning this procedure, you should set up a mechanism to automatically synchronize time throughout the network, such as the AIX **timed** daemon or an NTP setup.

1. Set up the KDC server.

   Note: The KDC server should ideally not be used for any other purpose; if the KDC is compromised, all Kerberos principals are compromised.

   In this scenario, kdc.austin.ibm.com will be configured as the KDC server. The following configuration is for **des3**. If **des** would be preferred for performance reasons, add the argument -e des-cbc-crc:normal to the addprinc call and the ktadd call for the **kadmin** below.

   To configure your network with **aes** encryption, add the -e aes256-cts:normal argument to the addprinc call and the ktadd call for the **kadmin** command.

   a. Install the krb5.server.rte fileset on kdc.austin.ibm.com.
   b. Set up the KDC server. In this scenario, the following command was used:

      ```
      config.krb5 -S -d austin.ibm.com -r AUSTIN.IBM.COM
      ```

      After running this command, the system will ask for a Master Database password and a password for the administrative principal.
   c. Create principals for each user and host by running the /usr/krb5/sbin/kadmin.local command on the KDC server. This example creates Kerberos principals that match the UNIX user name of the associated user. The principal name will be mapped to the user name by NFS to determine the

UNIX credential associated with the principal. For a description of how to use more general mappings between principals and user names, see "Identity mapping" on page 502. For this network, we created the following principals:

- adam
- brian
- charlie
- dave
- eric
- nfs/alpha.austin.ibm.com
- nfs/beta.austin.ibm.com
- nfs/gamma.austin.ibm.com

Note: The chosen user principal names must match the corresponding user names in the system's configured user registry (/etc/passwd, **LDAP**, **NIS**, and so on). NFS uses the principal name as a user name to obtain user and group IDs on the local system. If the names do not match, the access will be treated as an anonymous access.

The KDC is now configured.

2. Each NFS client and server will now be configured as Kerberos clients by using the **config.krb5** command. How this is done will depend on how the KDC was configured. In this scenario we ran the following command on each NFS system:

   ```
   config.krb5 -C -d austin.ibm.com -r AUSTIN.IBM.COM -c kdc.austin.ibm.com -s kdc.austin.ibm.com
   ```

   It is now possible to **kinit** as any of the user principals on any of the configured systems . For example, to **kinit** as user adam, run the following command:

   ```
   /usr/krb5/bin/kinit adam
   ```

   You will need to specify adam's Kerberos, not AIX, password.

   This example uses **kinit** to authenticate the user. It is possible to configure AIX to use Kerberos authentication during system login. For more information, see Authenticating to AIX Using Kerberos in *Security*.

3. Each NFS server will now be configured with the appropriate keytab entry. In this scenario, we configured the keytab entry for alpha.austin.ibm.com as an example; the exact same process will be used on beta.austin.ibm.com and gamma.austin.ibm.com.

   a. From alpha.austin.ibm.com, run the **kadmin** command. Then, run the following command:

      ```
      ktadd nfs/alpha.austin.ibm.com
      ```

      This creates a keytab file.

   b. Next, set up the **gssd** daemon to use the keytab file you just created with the **nfshostkey** command. In this scenario, we ran the following:

      ```
      nfshostkey -p nfs/alpha.austin.ibm.com -f /etc/krb5/krb5.keytab
      ```

   c. Set up the **gssd** daemon to start up automatically by running the following command:

      ```
      chnfs -S -B
      ```

   Repeat this setup for each system.

4. At this point, the NFS server will work, although, all users will come across as nobody. It is advisable to have all of the users exist on all servers with the same uid and gid; any users that do not exist will have access to the exported directory only as nobody. To get user names to map properly, you must configure the NFS registry daemon.

   a. Set up the domain using the **chnfsdom** command. In this scenario, the following command was run on all the NFS servers to set up austin.ibm.com as the domain:

      ```
      chnfsdom austin.ibm.com
      ```

b. Set up the /etc/nfs/realm.map file; this file should contain one line, with the realm name followed by the local domain. For our example network, these two files should look like this on all of the NFS servers:

```
realm.map AUSTIN.IBM.COM   austin.ibm.com
```

The realm entry in this file is not case-sensitive, so technically, this entry is not required.

c. For zeta.austin.ibm.com, which will not be an NFS server, start up the **gssd** daemon using the chnfs -S -B command. Before trying any Kerberos client operations, the user must use **kinit** to obtain valid credentials.

5. In this scenario, there is fast network link configure between alpha.austin.ibm.com and beta.austin.ibm.com. Across this link, beta.austin.ibm.com will see alpha.austin.ibm.com as fast_alpha.test.austin.ibm.com, and alpha.autsin.ibm.com will see beta.austin.ibm.com as fast_beta.test.austin.ibm.com. Because neither nfs/fast_alpha.test.austin.ibm.com nor nfs/fast_beta.test.austin.ibm.com are valid principals, they will be unable to use this link for mounts.

   To correct this, the **nfshostmap** command will be used, which will map principal to handle this situation.

   a. On alpha.austin.ibm.com, we ran the following command:

   ```
   nfshostmap -a beta.austin.ibm.com fast_beta.test.austin.ibm.com
   ```

   This tells alpha.austin.ibm.com that fast_beta.test.austin.ibm.com's principal is for beta.austin.ibm.com.

   b. On beta, we ran the following command:

   ```
   nfshostmap -a alpha.austin.ibm.com fast_alpha.test.austin.ibm.com
   ```

   Servers can have multiple host principals. Assuming that the IP address for fast_alpha is 10.0.0.1 and the IP address for fast_beta is 10.0.0.2, complete the following steps to add multiple host principals:

   a. Add the nfs/fast_alpha.test.austin.ibm.com and nfs/fast_beta.test.austin.ibm.com principals to the appropriate keytab files.

   b. Run the **nfshostkey** command on the alpha server, as follows:

   ```
   nfshostkey -a -p nfs/fast_alpha.test.austin.ibm.com -i 10.0.0.1
   ```

   c. Run the **nfshostkey** command on the beta server, as follows:

   ```
   nfshostkey -a -p nfs/fast_beta.test.austin.ibm.com -i 10.0.0.2
   ```

Unexporting an NFS file system

You can unexport an NFS directory using the following procedures.

- To unexport an NFS directory using SMIT:
  1. Type the following at a command prompt and press Enter:

     ```
     smit rmnfsexp
     ```

  2. Enter the appropriate path name in the PATHNAME of exported directory to be removed field.

     The directory is now removed from the /etc/exports file and is unexported.

     If the directory was exported to clients using NFS version 4, the unexport may fail due to file state on the server. File state means files in the exported directories are open by a client. You can either take actions to stop applications using that data, or you can forcefully unexport (**exportfs -F**) the data, which may result in failures for applications that are actively using the data.
- To unexport an NFS directory by using a text editor:
  1. Open the /etc/exports file with your favorite text editor.
  2. Find the entry for the directory you wish to unexport, and the delete that line.
  3. Save and close the /etc/exports file.
  4. If NFS is currently running, enter:

```
exportfs -u dirname
```

where *dirname* is the full path name of the directory you just deleted from the /etc/exports file. If the unexport fails due to access by NFS V4 clients, you can add a -F option to force the directory to be unexported.

Changing an exported file system

Change an exported NFS file system using the following procedures.

- To change an exported NFS file system using SMIT:
  1. To unexport the file system, type:
     ```
     exportfs -u /dirname
     ```
     where *dirname* is the name of the file system you want to change.
  2. Type:
     ```
     smit chnfsexp
     ```
  3. Enter the appropriate path name in the PATHNAME of exported directory field.
  4. Make whatever changes you want.
  5. Exit SMIT.
  6. Reexport the file system by entering:
     ```
     exportfs /dirname
     ```
     where *dirname* is the name of the file system you just changed.
- To change an exported NFS file system by using a text editor:
  1. To unexport the file system, type:
     ```
     exportfs -u /dirname
     ```
     where *dirname* is the name of the file system you want to change.
  2. Open the /etc/exports file with your favorite text editor.
  3. Make whatever changes you want.
  4. Save and close the /etc/exports file.
  5. Reexport the file system by entering:
     ```
     exportfs /dirname
     ```
     where *dirname* is the name of the file system you just changed.

Root user access to an exported file system

When a file system is exported, by default, the root user is not granted root access to that exported file systems.

When a root user on one host requests access to a particular file from NFS, the user ID of the requester is mapped by NFS to the user ID of user nobody (nobody is one of the user names placed in the /etc/password file by default). The access rights of user nobody are the same as those given to the public (*others*) for a particular file. For example, if *others* only has run permission for a file, then user nobody can only run the file.

To enable root user access to an exported file system, follow the instructions in "Changing an exported file system." If you use the SMIT method, specify in the HOSTS allowed root access field the name of the host to which you want to grant root access. If you edit the file with a text editor, add the qualifier -root=hostname to the file system entry. For example,

```
/usr/tps -root=hermes
```

specifies that the root user on host hermes may access the /usr/tps directory with root privileges.

Mounting an NFS file system explicitly

To mount an NFS directory explicitly, use the following procedure:

1. Verify that the NFS server has exported the directory:
   ```
   showmount -e ServerName
   ```
   where *ServerName* is the name of the NFS server. This command displays the names of the directories currently exported from the NFS server. If the directory you want to mount is not listed, export the directory from the server.

   Note: The **showmount** command will not work for file systems that were exported only as NFS version 4 file systems. For NFS version 4, the client can mount the root filesystem for the server and traverse the exported directory structure. Individual exported file systems do not have to be explicitly mounted to be accessed by the client.
2. Establish the local mount point using the **mkdir** command. A null (empty) directory that acts as the mount point (or place holder) of an NFS mount must be present for NFS to complete a mount successfully. This mount point can be created like any other directory, and no special attributes are needed.
3. Type:
   ```
   mount ServerName:/remote/directory /local/directory
   ```
   where *ServerName* is the name of the NFS server, /remote/*directory* is the directory on the NFS server you want to mount, and /local/*directory* is the mount point on the NFS client.
4. On the client machine, type the following SMIT fast path:
   ```
   smit mknfsmnt
   ```
5. Make changes to the following fields that are appropriate for your network configuration. Your configuration might not require completing all of the entries on this screen.

   Note: If the SMIT interface is being used, press the Tab key to change to the correct value for each field, but do *not* press Enter until completing step 7.
   - PATHNAME of mount point.
   - PATHNAME of remote directory.
   - HOST where remote directory resides.
   - MOUNT now, add entry to /etc/filesystems or both?
   - /etc/filesystems entry will mount the directory on system RESTART.
   - MODE for this NFS file system.
6. Change or use the default values for the remaining entries, depending on your NFS configuration.
7. When you finish making all the changes on this screen, SMIT mounts the NFS file system.
8. When the **Command:** field shows the OK status, exit SMIT.

The NFS file system is now ready to use.

Automount subsystem

The **automount** subsystem allows non-root users to mount remote file systems once the initial mount points are specified by the root user.

The /etc/auto_master file specifies this information. These mount points, known as keys, have a corresponding maps that determine which remote file system is mounted over it. The format of the /etc/auto_master file is as follows:

```
/key map
```

Note: The `/etc/auto_master` file is read when the **automount** command is initially executed, and changes to it will not take effect until the **automount** command is run again.

The most common maps are direct maps, indirect maps, and host maps.

Direct maps

Direct maps require a special key (`/-`) in the `/etc/auto_master` file.

The map is a file with the following format:

/directkey [*-options*] *server*:*/dir*

When a user accesses the */directkey* directory, the **automount** daemon will mount *server*:*/dir* over */directkey*.

Indirect maps

Another kind of map that determines which remote file system is mounted over a mount point is an indirect map.

Indirect maps have the following format:

indirectkey [*-options*] *server*:*/dir*

When a user accesses the `/key/`*indirectkey* directory, the **automount** daemon will mount *server*:*/dir* over `/key/`*indirectkey*.

Host maps

Host maps require a special map (`-hosts`) in the `/etc/auto_master` file.

The **automount** daemon will create a subdirectory under the `/key` directory for every server listed in the `/etc/hosts` file. When a user accesses the `/key/server` directory, the **automount** daemon will mount the server's exported directories over the `/key/server` directory.

Using AutoFS to automatically mount a file system

AutoFS relies on the use of the **automount** command to propagate the automatic mount configuration information to the **AutoFS** kernel extension and start the **automountd** daemon.

Through this configuration propagation, the extension automatically and transparently mounts file systems whenever a file or a directory within that file system is opened. The extension informs the **automountd** daemon of mount and unmount requests, and the **automountd** daemon actually performs the requested service.

Because the name-to-location binding is dynamic within the **automountd** daemon, updates to a Network Information Service (NIS) map used by the **automountd** daemon are transparent to the user. Also, there is no need to premount shared file systems for applications that have hard-coded references to files and directories, nor is there a need to maintain records of which hosts must be mounted for particular applications.

AutoFS allows file systems to be mounted as needed. With this method of mounting directories, all file systems do not need to be mounted all of the time; only those being used are mounted.

For example, to mount an NFS directory automatically:

1. Verify that the NFS server has exported the directory by entering:

   `showmount -e` *ServerName*

   where *ServerName* is the name of the NFS server. This command displays the names of the directories currently exported from the NFS server.

2. Create an **AutoFS** master file and map file. **AutoFS** mounts and unmounts the directories specified in these map files. For example, suppose you want **AutoFS** to mount the `/local/dir1` and `/local/dir2` directories as needed from the **serve1** server onto the `/remote/dir1` and `/remote/dir2` directories, respectively. The `auto_master` file entry would be as follows:

   ```
   /remote  /tmp/mount.map
   ```

   The `/tmp/mount.map` file entry would be as follows:

   ```
   dir1 -rw serve1:/local/dir1
   dir2 -rw serve1:/local/dir2
   ```

3. Ensure that the **AutoFS** kernel extension is loaded and the **automountd** daemon is running. This can be accomplished in two ways:
   a. Using the **automount** command: Issue `/usr/bin/automount -v`.
   b. Using **SRC**: Issue `lssrc -s automountd`. If the **automountd** subsystem is not running, issue `startsrc -s automountd`.

      Note: Starting the **automountd** daemon with the **startsrc** command will ignore any changes that have been made to the `auto_master` file.

4. To stop the **automount** daemon, issue the `stopsrc -s automountd` command.

   If, for some reason, the **automountd** daemon was started without the use of **SRC**, issue:

   ```
   kill automountd_PID
   ```

   where *automountd_PID* is the process ID of the **automountd** daemon. (Running the **ps -e** command displays the process ID of the **automountd** daemon.) The **kill** command sends a SIGTERM signal to the **automountd** daemon.

Establishing predefined NFS mounts

You can establish predefined NFS mounts using the one of the following procedures.

Note: Define the **bg** (background) and **intr** (interruptible) options in the `/etc/filesystems` file when establishing a predefined mount that is mounted during system startup. Mounts that are noninterruptible and running in the foreground can hang the client if the network or server is down when the client system starts up. If a client cannot access the network or server, the user must start the machine again in maintenance mode and edit the appropriate mount requests.

- To establish predefined mounts through SMIT:
  1. Type:

     ```
     smit mknfsmnt
     ```

  2. Specify values in this screen for each mount you want to predefine. Specify a value for each required field (those marked with an asterisk (*) in the left margin). Also specify values for the other fields or accept their default values. This method creates an entry in the `/etc/filesystems` file for the desired mount and attempts the mount.
- To establish the NFS default mounts by editing the `/etc/filesystems` file:
  1. Open the `/etc/filesystems` file with a text editor.
  2. Add entries for each of the remote file systems to be mounted when the system is started. For example:

     ```
     /home/jdoe:
     dev = /home/jdoe
     mount = false
     vfs = nfs
     nodename = mach2
     options = ro,soft
     type = nfs_mount
     ```

     This stanza directs the system to mount the `/home/jdoe` remote directory over the local mount point of the same name. The file system is mounted as read-only (`ro`). Because it is also mounted as `soft`,

an error is returned in the event the server does not respond. By specifying the *type* parameter as `nfs_mount`, the system attempts to mount the `/home/jdoe` file (along with any other file systems that are specified in the `type = nfs_mount` group) when the **mount -t nfs_mount** command is issued.

The example stanza below directs the system to mount the `/usr/games` file system at system startup time. If the mount fails, the system continues to attempt to mount in the background.

```
/usr/games:
dev = /usr/games
mount = true
vfs = nfs
nodename = gameserver
options = ro,soft,bg
type = nfs_mount
```

The following parameters are required for stanzas pertaining to NFS mounts:

Item	Description
`dev=`*`filesystem_name`*	Specifies the path name of the remote file system being mounted.
`mount=[true|false]`	If `true` the NFS file system is mounted when the system boots. If `false`, the NFS file system is not be mounted when the system boots.
`nodename=`*`hostname`*	Specifies the host machine on which the remote file system resides.
`vfs=nfs`	Specifies that the virtual file system being mounted is an NFS file system.

The following parameters are optional for stanzas pertaining to NFS mounts:

Item	Description
`type=`*`type_name`*	Defines the file system being mounted as part of the *type_name* mount group. This parameter is used with the **mount -t** command, which mounts groups of specified file systems at the same time.
`options=`*`options`*	Specifies one or more of the following *options* parameters: **`biods=`***`N`* Specifies the maximum number of **biod** daemons to use. The default is seven for NFS version 2, and four for NFS version 3 and version 4. **`bg`** Specifies to try the mount again in the background if the first mount attempt fails. **`fg`** Specifies to try the mount again in the foreground if the first mount attempt fails. **`noacl`** Disables, for this mount only, the Access Control List (ACL) support provided by the NFS journaled file system. When used between two systems, NFS supports access control lists. If the `noacl` option is used when mounting a file system, NFS does not use ACLs. The effect of the `noacl` option equals what happens when an NFS client on a system mounts from an NFS server that does not support ACLs. For more information about ACLs, refer to "NFS Access Control Lists support" on page 480. **`retry=`***`n`* Sets the number of times to try the mount. **`rsize=`***`n`* Sets the read buffer size to the number of bytes specified by *n*. **`wsize=`***`n`* Sets the write buffer size to the number of bytes specified by *n*.

Item	Description
	timeo=*n* Sets the NFS time out to the tenths of a second specified by *n*. Use this variable to avoid situations that can occur in networks where the server load can cause inadequate response time. **retrans=***n* Sets the number of NFS retransmissions to the number specified by *n*. **port=***n* Sets the server port to the number specified by *n*. **soft** Returns an error if the server does not respond. **hard** Continues to try the request until the server responds. **Note:** When you specify a `hard` mount, it is possible that the process can hang while waiting for a response. To be able to interrupt the process and end it from the keyboard, use the `intr` variable in the mount variables. **intr** Allows keyboard interrupts on hard mounts. **ro** Sets the read-only variable. **rw** Sets the read-write variable. Use the `hard` variable along with this variable to avoid error conditions that can conflict with applications if a `soft` mount is attempted as read-write. See "NFS troubleshooting" on page 520 for information on hard- and soft-mounted problems. **secure** Specifies to use a more secure protocol for NFS transactions. **sec** The **sec** option specifies the security flavor list for the NFS mount. The available flavors are **des**, **unix**, **sys**, **krb5**, **krb5i**, and **krb5p**. This option only applies to AIX 5.3 or later. **actimeo=***n* Extends flush time by *n* seconds for both regular files and directories. **Note:** The attribute cache retains file attributes on the client. Attributes for a file are assigned a time to be erased. If the file is modified before the flush time, then the flush time is extended by the time since the previous modification (under the assumption that recently changed files are likely to change again soon). There are minimum and maximum flush time extensions for regular files and for directories. **vers** Specifies NFS version. The default is the version of NFS protocol used between the client and server and is the highest one available on both systems. If the NFS server does not support NFS Version 3, the NFS mount will use NFS Version 2. Use the **vers** option to select the NFS version. By default, the NFS mount will never use NFS Version 4 unless specified. **acregmin=***n* Holds cached attributes for at least *n* seconds after file modification. **acregmax=***n* Holds cached attributes for no more than *n* seconds after file modification. **acdirmin=***n* Holds cached attributes for at least *n* seconds after directory update. **acdirmax=***n* Holds cached attributes for no more than *n* seconds after directory update.

Item	Description

	cio Specifies the file system to be mounted for concurrent readers and writers. I/O on files in this file system will behave as if they had been opened with **O_CIO** specified in the **open()** system call. Using this option will prevent access in any manner other than CIO. It is impossible to use cached I/O on a file system mounted with the **cio** option. This means that mapping commands such as **mmap()** and **shmat()** will fail with **EINVAL** when used on any file in a file system mounted with the **cio** option. One side effect of this is that it is impossible to run binaries out of a **cio**-mounted file system, since the loader may use **mmap()**.
	dio Specifies that I/O on the file system will behave as if all the files had been opened with **O_DIRECT** specified in the **open()** system call.
	Note: Using the **-odio** or **-ocio** flags can help performance on certain workloads, but users should be aware that using these flags will prevent file caching for these file systems. Because readahead is disabled for these file systems, this may decrease performance for large sequential reads.
	maxpout=*n*
	Specifies the page-out level for files on this file system at which threads should be slept. If **maxpout** is specified, you must also specify **minpout**. This value must be nonnegative and greater than **minpout**. The default is the kernel **maxpout** level.
	minpout=*n*
	Specifies the page-out level for files on this file system at which threads should be readied. If **minpout** is specified, you must also specify **maxpout**. This value must be nonnegative. The default is the kernel **minpout** level.
	rbr Uses the release-behind-when-reading capability. When sequential reading of a file in this file system is detected, the real memory pages used by the file are released once the pages are copied to internal buffers.
	Note: If you do not set the following options, the kernel automatically sets them to these default values:
	`fg`
	`retry=10000`
	`rsize=8192`
	`wsize=8192`
	`timeo=7`
	`retrans=5`
	`port=NFS_PORT`
	`hard`
	`secure=off`
	`acregmin=3`
	`acregmax=60`
	`acdirmin=30`
	`acdirmax=60`

3. Remove any directory entries that you do not want to mount automatically at system startup.
4. Save and close the file.
5. Run the **mount -a** command to mount all the directories specified in the `/etc/filesystems` file.

Unmounting an explicitly or automatically mounted file system

The following procedure can be used to unmount an explicitly or automatically mounted NFS directory.

To unmount an explicitly or automatically mounted NFS directory, type:

```
umount /directory/to/unmount
```

Removing predefined NFS mounts

You can remove a predefined NFS mount using the following procedures.

- To remove a predefined NFS mount through SMIT:
  1. Type:

```
smit rmnfsmnt
```

- To remove a predefined NFS mount by editing the `/etc/filesystems` file:
  1. Enter the command: `umount /directory/to/unmount`.
  2. Open the `/etc/filesystems` file with your favorite editor.
  3. Find the entry for the directory you just unmounted, and then delete it.
  4. Save and close the file.

PC-NFS

PC-NFS is a program for personal computers that enables the personal computer to mount file systems exported by a Network File System (NFS) server.

The personal computer can also request network addresses and host names from the NFS server. Additionally, if the NFS server is running the **rpc.pcnfsd** daemon, the personal computer can access authentication and print-spooling services.

You might want to configure the **rpc.pcnfsd** daemon on the following:

- Systems that perform user authentication services
- Systems that offer print-spooling
- All Network Information Service (NIS) master and slave servers.

Note: Because NIS networks are typically configured so that PC-NFS can pick any NIS server as the default server, it is important that all servers have the **rpc.pcnfsd** daemon running. If running this daemon on all NIS servers is not practical, or if you want to limit requests to a specific server, add a **net pcnfsd** command to the `autoexec.bat` file on each personal computer to force it to use a specific NIS server.

Related information:

Network Information Services (NIS)

PC-NFS authentication service

By default, PC-NFS presents itself to NFS servers as the `nobody` user. With `nobody` privileges, all personal computer user files appear as owned by `nobody`, and consequently you cannot distinguish between different personal computer users.

The authentication capability of the **rpc.pcnfsd** daemon allow you to monitor system resources and security by recognizing individual users and assigning them different privileges.

With the **rpc.pcnfsd** daemon running, a PC-NFS user can issue the **net name** command from a personal computer to log in to PC-NFS in the same manner as a user can log in to this operating system. The user name and password are verified by the **rpc.pcnfsd** daemon. This authentication procedure does not make a server more secure, but it does provide more control over access to files that are available through NFS.

PC-NFS print-spooling service

The print-spooling service of the **rpc.pcnfsd** daemon enables personal computers running PC-NFS to print to printers not directly attached to the personal computer.

Specifically, PC-NFS redirects files intended for personal computer printers to a file on an NFS server. This file is placed in a spool directory on the NFS server. The **rpc.pcnfsd** daemon then invokes the server printing facility. (The spooling directory must be in an exported file system so that PC-NFS clients can mount it.) When PC-NFS requests that the **rpc.pcnfsd** daemon print the file, it provides the following information:

- Name of the file to be printed
- Login ID of the user on the client

- Name of the printer to be used.

Configuring the rpc.pcnfsd daemon

For the best performance, configure the **rpc.pcnfsd** daemon using these steps.

To configure the **rpc.pcnfsd** daemon:

1. Install PC-NFS program on your personal computer.
2. Select a location for the spool directory on the NFS server. The default spool directory is `/var/tmp`. The spool directory must have at least 100K bytes of free space.
3. Export the spool directory. Do not put access restrictions on the exported directory that could cause access problems in your network. For details of this procedure, see "Exporting an NFS file system" on page 503.
4. Start the **rpc.pcnfsd** daemon by following the instructions in "Starting the rpc.pcnfsd daemon."
5. Verify that the **rpc.pcnfsd** daemon is accessible by following the instructions in "Verifying the rpc.pcnfsd daemon is accessible".

Note: Because printer-redirection requests sometimes cause file listings of zero length to be left in the PC-NFS spool directories, periodically clear spooling directories of these entries.

Starting the rpc.pcnfsd daemon

To start the **rpc.pcnfsd** daemon using the default spooling directory, use the following procedure.

1. With a text editor, uncomment the following entry in the `/etc/inetd.conf` file:
   ```
   pcnfsd sunrpc_udp udp wait root /usr/sbin/rpc.pcnfsd pcnfsd 150001 1
   ```
2. Save the file and exit the text editor.

To start the **rpc.pcnfsd** daemon using a directory that is different from the default:

1. Use a text editor to add the following entry to the `/etc/rc.nfs` file:
   ```
   if [ -f /usr/sbin/rpc.pcnfsd ] ; then
   /usr/sbin/rpc.pcnfsd -s spooldir ; echo ' rpc.pcnfsd\c'
   fi
   ```
   where *spooldir* specifies the full path name of the spool directory.
2. Save the file and exit the text editor.
3. Using a text editor, comment the following entry in the `/etc/inetd.conf` file:
   ```
   #pcnfsd sunrpc_udp udp wait root /usr/sbin/rpc.pcnfsd pcnfsd 150001 1
   ```
   Placing a pound sign (#) at the beginning of the line prevents the **inetd** daemon from starting the **rpc.pcnfsd** daemon using the default spool directory.
4. Start the **rpc.pcnfsd** daemon print spooler by typing the following at the command line:
   ```
   /usr/sbin/rpc.pcnfsd -s spooldir
   ```
   where *spooldir* specifies the full path name of the spool directory.

For more information on updating the **inetd** configuration database, see "Configuring the inetd daemon" on page 333.

Note: The default directory that the **rpc.pcnfsd** daemon uses cannot be changed from the `inetd.conf` file.

Verifying the rpc.pcnfsd daemon is accessible

Follow this procedure to determine whether the **rpc.pcnfsd** daemon is accessible.

To verify that the **rpc.pcnfsd** daemon is accessible, type:

```
rpcinfo -u host 150001
```

where *host* specifies the host name of the system on which you are configuring **rpc.pcnfsd**, and 15001 is the RPC program number of the **rpc.pcnfsd** daemon. After you enter the command, you will receive a message that the program is ready and waiting.

LDAP automount maps

You can configure the automount subsystem to retrieve its maps from an LDAP server.

To administer automount maps in LDAP, add the following line to the `/etc/irs.conf` file:

```
automount nis_ldap
```

In order to administer automount maps in LDAP, you need to create the appropriate LDIF files. You can convert local automount map files to LDIF format using the **nistoldif** command. As an example, if the LDAP server is named `ldapserver`, then its base suffix is `dc=suffix`, and the `/etc/auto_home` map file contains the following lines:

```
user1   server1:/home/user1
user2   server1:/home/user2
user3   server1:/home/user3
```

Use the following commands to create the LDIF file for the `/etc/auto_home` map file, and add it to the LDAP server:

```
nistoldif -d dc=suffix -sa -f /etc/auto_home > /tmp/auto_home.ldif
ldapadd -D cn=admin -w passwd -h ldapserver -f /tmp/auto_home.ldif
```

In order to edit or remove existing automount entries from an LDAP server, the LDIF files must be created manually. For example, if the home directory of `user2` is now on `server2`, the following LDIF should be created:

```
# cat /tmp/ch_user2.ldif
dn: automountKey=user2,automountMapName=auto_home,dc=suffix
changetype: modify
replace: automountInformation
automountInformation: server2:/home/user2
```

After creating the above LDIF, run the following command:

```
ldapmodify -D cn=admin -w passwd -h ldapserver -f /tmp/ch_user2.ldif
```

You must also create an LDIF file to remove a user. For example, to remove `user3`, create the following LDIF:

```
# cat /tmp/rm_user3.ldif
dn: automountKey=user3,automountMapName=auto_home,dc=suffix
changetype: delete
```

After creating the above LDIF, run the following command:

```
ldapmodify -D cn=admin -w passwd -h ldapserver -f /tmp/rm_user3.ldif
```

WebNFS

The operating system provides NFS server capability for WebNFS.

Defined by Oracle, WebNFS is a simple extension of the NFS protocol that allows easier access to servers and clients through Internet firewalls.

A WebNFS-enhanced web browser can use an NFS universal resource locator (URL) to access data directly from the server. An example NFS URL is:

```
nfs://www.YourCompany.com/
```

WebNFS works in tandem with existing web-based protocols to provide data to clients.

WebNFS also takes advantage of the scalability of NFS servers.

Network lock manager

The network lock manager is a facility that works in cooperation with the Network File System (NFS) to provide a System V style of advisory file and record locking over the network.

The network lock manager (**rpc.lockd**) and the network status monitor (**rpc.statd**) are network-service daemons. The **rpc.statd** daemon is a user level process while the **rpc.lockd** daemon is implemented as a set of kernel threads (similar to the NFS server). Both daemons are essential to the ability of the kernel to provide fundamental network services.

Note:

1. Mandatory or enforced locks are not supported over NFS.
2. Network Lock Manager is specific to NFS Version 2 and Version 3.

Network lock manager architecture

The network lock manager contains both server and client functions.

The client functions are responsible for processing requests from the applications and sending requests to the network lock manager at the server. The server functions are responsible for accepting lock requests from clients and generating the appropriate locking calls at the server. The server will then respond to the locking request of the client.

In contrast to NFS, which is stateless, the network lock manager has an implicit state. In other words, the network lock manager must remember whether the client currently has a lock. The network status monitor, **rpc.statd**, implements a simple protocol that allows the network lock manager to monitor the status of other machines on the network. By having accurate status information, the network lock manager can maintain a consistent state within the stateless NFS environment.

Network file locking process

When an application wants to obtain a lock on a local file, it sends its request to the kernel using the **lockf**, **fcntl**, or **flock** subroutines.

The kernel then processes the lock request. However, if an application on an NFS client makes a lock request for a remote file, the Network Lock Manager client generates a Remote Procedure Call (RPC) to the server to handle the request.

When the client receives an initial remote lock request, it registers interest in the server with the client's **rpc.statd** daemon. The same is true for the network lock manager at the server. On the initial request from a client, it registers interest in the client with the local network status monitor.

Crash recovery process

The **rpc.statd** daemon on each machine notifies the **rpc.statd** daemon on every other machine of its activities. When the **rpc.statd** daemon receives notice that another machine crashed or recovered, it notifies its **rpc.lockd** daemon.

If a server crashes, clients with locked files must be able to recover their locks. If a client crashes, its servers must hold the client locks while it recovers. Additionally, to preserve the overall transparency of NFS, the crash recovery must occur without requiring the intervention of the applications themselves.

The crash recovery procedure is simple. If the failure of a client is detected, the server releases the failed client locks on the assumption that the client application will request locks again as needed. If the crash and recovery of a server is detected, the client lock manager retransmits all lock requests previously

granted by the server. This retransmitted information is used by the server to reconstruct its locking state during a grace period. (The grace period, 45 seconds by default, is a time period within which a server allows clients to reclaim their locks.)

The **rpc.statd** daemon uses the host names kept in `/var/statmon/sm` and `/var/statmon/sm.bak` to keep track of which hosts must be informed when the machine needs to recover operations.

Starting the network lock manager

By default, the `/etc/rc.nfs` script starts the **rpc.lockd** and **rpc.statd** daemons along with the other NFS daemons.

If NFS is already running, you can verify that the **rpc.lockd** and **rpc.statd** daemons are running by following the instructions in "Getting the current status of the NFS daemons" on page 489. The status of these two daemons should be *active*. If the **rpc.lockd** and **rpc.statd** daemons are not active, and therefore not running, do the following:

1. Using your favorite text editor, open the `/etc/rc.nfs` file.
2. Search for the following lines:

```
if [ -x /usr/sbin/rpc.statd ]; then
       startsrc -s rpc.statd
fi
if [ -x /usr/sbin/rpc.lockd ]; then
       startsrc -s rpc.lockd
fi
```

3. If there is a pound sign (#) at the beginning of any of these lines, delete the character, then save and exit the file. Then start the **rpc.statd** and **rpc.lockd** daemons by following the instructions in "Starting the NFS daemons" on page 489.

   Note: Sequence is important. Always start the **statd** daemon first.
4. If NFS is running and the entries in the `/etc/rc.nfs` file are correct, stop and restart the **rpc.statd** and **rpc.lockd** daemons by following the instructions in "Stopping the NFS daemons" on page 489 and "Starting the NFS daemons" on page 489.

   Note: Sequence is important. Always start the **statd** daemon first.

If the **rpc.statd** and **rpc.lockd** daemons are still not running, see "Troubleshooting the network lock manager."

Troubleshooting the network lock manager

Some network lock manager problems you encounter can be solved by using the following tips.

If you receive a message on a client similar to:

```
clnttcp_create: RPC: Remote System error - Connection refused
rpc.statd:cannot talk to statd at {server}
```

then the machine thinks there is another machine which needs to be informed that it might have to take recovery measures. When a machine restarts, or when the **rpc.lockd** and the **rpc.statd** daemons are stopped and restarted, machine names are moved from `/var/statmon/sm` to `/var/statmon/sm.bak` and the **rpc.statd** daemon tries to inform each machine corresponding to each entry in `/var/statmon/sm.bak` that recovery procedures are needed.

If the **rpc.statd** daemon can reach the machine, then its entry in `/var/statmon/sm.bak` is removed. If the **rpc.statd** daemon cannot reach the machine, it will keep trying at regular intervals. Each time the machine fails to respond, the timeout generates the above message. In the interest of locking integrity, the daemon will continue to try; however, this can have an adverse effect on locking performance. The handling is different, depending on whether the target machine is just unresponsive or semi-permanently taken out of production. To eliminate the message:

1. Verify that the **statd** and **lockd** daemons on the server are running by following the instructions in "Getting the current status of the NFS daemons" on page 489. (The status of these two daemons should be *active*.)
2. If these daemons are not running, start the **rpc.statd** and **rpc.lockd** daemons on the server by following the instructions in "Starting the NFS daemons" on page 489.

   Note: Sequence is important. Always start the **statd** daemon first.

   After you have restarted the daemons, remember that there is a grace period. During this time, the **lockd** daemons allow reclaim requests to come from other clients that previously held locks with the server, so you might not get a new lock immediately after starting the daemons.

Alternatively, eliminate the message by:

1. Stop the **rpc.statd** and **rpc.lockd** daemons on the client by following the instructions in "Stopping the NFS daemons" on page 489.
2. On the client, remove the target machine entry from `/var/statmon/sm.bak` file by entering:

   ```
   rm /var/statmon/sm.bak/TargetMachineName
   ```

   This action keeps the target machine from being aware that it might need to participate in locking recovery. It should only be used when it can be determined that the machine does not have any applications running that are participating in network locking with the affected machine.
3. Start the **rpc.statd** and **rpc.lockd** daemons on the client by following the instructions in "Starting the NFS daemons" on page 489.

If you are unable to obtain a lock from a client, do the following:

1. Use the **ping** command to verify that the client and server can reach and recognize each other. If the machines are both running and the network is intact, check the host names listed in the `/var/statmon/hosts` file for each machine. Host names must exactly match between server and client for machine recognition. If a name server is being used for host name resolution, make sure the host information is exactly the same as that in the `/var/statmon/hosts` file.
2. Verify that the **rpc.lockd** and **rpc.statd** daemons are running on both the client and the server by following the instructions in "Getting the current status of the NFS daemons" on page 489. The status of these two daemons should be *active*.
3. If they are not active, start the **rpc.statd** and **rpc.lockd** daemons by following the instructions in "Starting the NFS daemons" on page 489.
4. If they are active, you might need to reset them on both clients and servers. To do this, stop all the applications that are requesting locks.
5. Next, stop the **rpc.statd** and **rpc.lockd** daemons on both the client and the server by following the instructions in "Stopping the NFS daemons" on page 489.
6. Now, restart the **rpc.statd** and **rpc.lockd** daemons, first on the server and then on the client, by following the instructions in "Starting the NFS daemons" on page 489.

   Note: Sequence is important. Always start the **statd** daemon first.

If the procedure does not alleviate the locking problem, run the **lockd** daemon in debug mode, by doing the following:

1. Stop the **rpc.statd** and **rpc.lockd** daemons on both the client and the server by following the instructions in "Stopping the NFS daemons" on page 489.
2. Start the **rpc.statd** daemon on the client and server by following the instructions in "Starting the NFS daemons" on page 489.
3. Start the **rpc.lockd** daemon on the client and server by typing:

   ```
   /usr/sbin/rpc.lockd -d1
   ```

When invoked with the **-d1** flag, the **lockd** daemon provides diagnostic messages to syslog. At first, there will be a number of messages dealing with the grace period; wait for them to time out. After the grace period has timed out on both the server and any clients, run the application that is having lock problems and verify that a lock request is transmitted from client to server and server to client.

You can restrict the number range of IP ports used by the NFS client for communication with the NFS server by setting the **NFS_PORT_RANGE** variable in the `/var/statmon/environment` file.

NFS port ranges

The **NFS_PORT_RANGE** environment variable can be used to limit the source port of network calls the client makes to the server.

If used, this environment variable must be added to the `/etc/environment` file. The format of the environment variable is as follows:

```
NFS_PORT_RANGE=udp[4000-5000]:tcp[7000-8000]
```

In this example, UDP packets sent by the client have a source port in the range 4000 - 5000, and TCP connections have a source port in the range 7000 - 8000. To avoid port reuse problems, the port numbers that are specified in this range must not be used as fixed port numbers for any of the Network File System (NFS) daemons in the `/etc/services` file.

NFS security

Information about NFS security can be found in several places.

The Network File System security topic in *Security* explains the details about DES security. For information about Kerberos security, see "Setting up a network for RPCSEC-GSS" on page 504.

NFS troubleshooting

As with other network services, problems can occur on machines that use the Network File System (NFS). Troubleshooting for these problems involves understanding the strategies for tracking NFS problems, recognizing NFS-related error messages, and selecting the appropriate solutions.

When tracking down an NFS problem, isolate each of the three main points of failure to determine which is not working: the server, the client, or the network itself.

Note: See "Troubleshooting the network lock manager" on page 518 for file lock problems.

Hard-mounted and soft-mounted file problems

When the network or server has problems, programs that access hard-mounted remote files fail differently from those that access soft-mounted remote files.

If a server fails to respond to a hard-mount request, NFS prints the message:

```
NFS server hostname not responding, still trying
```

Hard-mounted remote file systems cause programs to hang until the server responds because the client retries the mount request until it succeeds. Use the **-bg** flag with the **mount** command when performing a hard mount so if the server does not respond, the client will retry the mount in the background.

If a server fails to respond to a soft-mount request, NFS prints the message:

```
Connection timed out
```

Soft-mounted remote file systems return an error after trying unsuccessfully for a while. Unfortunately, many programs do not check return conditions on file system operations, so you do not see this error message when accessing soft-mounted files. However, this NFS error message prints on the console.

Identifying NFS problems

If you are encountering NFS problems, follow these steps.

If a client is having NFS trouble, do the following:

1. Verify that the network connections are good.
2. Verify that the **inetd**, **portmap**, and **biod** daemons are running on the client, by following the instructions in "Getting the current status of the NFS daemons" on page 489.
3. Verify that a valid mount point exists for the file system being mounted. For more information, see "Configuring an NFS client" on page 501.
4. Verify that the server is up and running by running the following command at the shell prompt of the client:

   ```
   /usr/bin/rpcinfo  -p  server_name
   ```

   If the server is up, a list of programs, versions, protocols, and port numbers is printed, similar to the following:

   ```
   program    vers  proto      port
   100000        2      tcp       111    portmapper
   100000        2      udp       111    portmapper
   100005        1      udp      1025    mountd
   100001        1      udp      1030    rstatd
   100001        2      udp      1030    rstatd
   100001        3      udp      1030    rstatd
   100002        1      udp      1036    rusersd
   100002        2      udp      1036    rusersd
   100008        1      udp      1040    walld
   100012        1      udp      1043    sprayd
   100005        1      tcp       694    mountd
   100003        2      udp      2049    nfs
   100024        1      udp       713    status
   100024        1      tcp       715    status
   100021        1      tcp       716    nlockmgr
   100021        1      udp       718    nlockmgr
   100021        3      tcp       721    nlockmgr
   100021        3      udp       723    nlockmgr
   100020        1      udp       726    llockmgr
   100020        1      tcp       728    llockmgr
   100021        2      tcp       731    nlockmgr
   ```

   If a similar response is not returned, log in to the server at the server console and check the status of the **inetd** daemon by following the instructions in "Getting the current status of the NFS daemons" on page 489.
5. Verify that the **mountd**, **portmap** and **nfsd** daemons are running on the NFS server by entering the following commands at the client shell prompt:

   ```
   /usr/bin/rpcinfo  -u  server_name  mount
   /usr/bin/rpcinfo  -u  server_name  portmap
   /usr/bin/rpcinfo  -u  server_name  nfs
   ```

   If the daemons are running at the server, the following responses are returned:

   ```
   program  100005  version  1  ready  and  waiting
   program  100000  version  2  ready  and  waiting
   program  100003  version  2  ready  and  waiting
   ```

   The program numbers correspond to the commands, respectively, as shown in the previous example. If a similar response is not returned, log in to the server at the server console and check the status of the daemons by following the instructions in "Getting the current status of the NFS daemons" on page 489.
6. Verify that the `/etc/exports` file on the server lists the name of the file system that the client wants to mount and that the file system is exported. Do this by entering the command:

```
showmount -e server_name
```

This command lists all the file systems currently exported by the *server_name*.

7. For NFS version 4, verify that the NFSv4 domain is properly set.
8. For NFS version 4, verify that the **nfsrgyd** daemon is running.
9. If you are using enhanced security, see "RPCSEC-GSS problem determination" on page 527.

Asynchronous write errors

When an application program writes data to a file in an NFS-mounted file system, the write operation is scheduled for asynchronous processing by the **biod** daemon.

If an error occurs at the NFS server at the same time that the data is actually written to disk, the error is returned to the NFS client and the **biod** daemon saves the error internally in NFS data structures. The stored error is subsequently returned to the application program the next time it calls either the **fsync** or **close** functions. As a consequence of such errors, the application is not notified of the write error until the program closes the file. A typical example of this event is when a file system on the server is full, causing writes attempted by a client to fail.

nfs_server error message

When your transmit buffer is too small, an error message is returned.

Insufficient transmit buffers on your network can cause the following error message:

```
nfs_server:  bad  sendreply
```

To increase transmit buffers, use the System Management Interface Tool (SMIT) fast path, `smit commodev`. Then select your adapter type, and increase the number of transmit buffers.

mount error messages

A remote mounting process can fail in several ways. The error messages associated with mounting failures are described here.

mount: ... already mounted
The file system that you are trying to mount is already mounted.

mount: ... not found in /etc/filesystems
The specified file system or directory name cannot be matched.

If you issue the **mount** command with either a directory or file system name but not both, the command looks in the `/etc/filesystems` file for an entry whose file system or directory field matches the argument. If the **mount** command finds an entry such as the following:

```
/dancer.src:
          dev=/usr/src
          nodename  =  d61server
          type  =  nfs
          mount  =  false
```

then it performs the mount as if you had entered the following at the command line:

```
/usr/sbin/mount -n dancer -o rw,hard /usr/src /dancer.src
```

... not in hosts database
On a network without Network Information Service, this message indicates that the host specified in the **mount** command is not in the `/etc/hosts` file. On a network running NIS, the message indicates that NIS could not find the host name in the `/etc/hosts` database or that the NIS **ypbind** daemon on your machine has died. If the `/etc/resolv.conf` file exists so that the name server is being used for host name resolution, there might be a problem in the **named** database. See "Host name resolution on an NFS server" on page 526.

Check the spelling and the syntax in your **mount** command. If the command is correct, your network does not run NIS, and you only get this message for this host name, check the entry in the `/etc/hosts` file.

If your network is running NIS, make sure that the **ypbind** daemon is running by entering the following at the command line:

```
ps -ef
```

You should see the **ypbind** daemon in the list. Try using the **rlogin** command to log in remotely to another machine, or use the **rcp** command to remote-copy something to another machine. If this also fails, your **ypbind** daemon is probably stopped or hung.

If you only get this message for this host name, check the `/etc/hosts` entry on the NIS server.

mount: ... server not responding: port mapper failure - RPC timed out
Either the server you are trying to mount from is down or its port mapper is stopped or hung. Try restarting the server to activate the **inetd**, **portmap**, and **ypbind** daemons.

If you cannot log in to the server remotely with the **rlogin** command but the server is up, check the network connection by trying to log in remotely to some other machine. Also check the server network connection.

mount: ... server not responding: program not registered
This means that the **mount** command got through to the port mapper, but the **rpc.mountd** NFS mount daemon was not registered.

mount: access denied ...
Your machine name is not in the export list for the file system you are trying to mount from the server.

You can get a list of the server exported file systems by running the following command at the command line:

```
showmount -e hostname
```

If the file system you want is not in the list, or your machine name or netgroup name is not in the user list for the file system, log in to the server and check the `/etc/exports` file for the correct file system entry. A file system name that appears in the `/etc/exports` file, but not in the output from the **showmount** command, indicates a failure in the **mountd** daemon. Either the daemon could not parse that line in the file, it could not find the directory, or the directory name was not a locally mounted directory. If the `/etc/exports` file looks correct and your network runs NIS, check the **ypbind** daemon on the server. It may be stopped or hung.

mount: ...: Permission denied
This message is a generic indication that some part of authentication failed on the server. It could be that, in the previous example, you are not in the export list, the server could not recognize your machine **ypbind** daemon, or that the server does not accept the identity you provided.

Check the `/etc/exports` file on the server and, if applicable, the **ypbind** daemon. In this case, you can just change your host name with the **hostname** command and retry the **mount** command.

mount: ...: Not a directory
Either the remote path or the local path is not a directory. Check the spelling in your command and try to run on both directories.

mount: ...: You are not allowed
You must have root authority or be a member of the system group to run the **mount** command on your machine because it affects the file system for all users on that machine. NFS mounts and unmounts are only allowed for root users and members of the system group.

Related information:

Network Information Services (NIS)

Causes of slow access times for NFS

If access to remote files seems unusually slow, ensure that access time is not being inhibited by a runaway daemon, a bad **tty** line, or a similar problem.

Network connections:

Use the **nfsstat** command to gather information about your network connections.

The **nfsstat** command determines whether you are dropping packets. Use the **nfsstat -c** and **nfsstat -s** commands to determine if the client or server is retransmitting large blocks. Retransmissions are always a possibility due to lost packets or busy servers. A retransmission rate of five percent or more is considered high.

The probability of retransmissions can be reduced by changing communication adapter transmit queue parameters. The SMIT menu can be used to change these parameters. For more information, refer to Available system management interfaces in *Operating system and device management*.

The following values are recommended for NFS servers.

Note:

1. Apply these values to NFS clients if retransmissions persist.
2. All nodes on a network must use the same MTU size.

Table 91. Communication Adapter Maximum Transmission Unit (MTU) and Transmit Queue Sizes

Adapter	MTU	Transmit queue
Token Ring		
4 Mb	1500 3900	50 40 (Increase if the **nfsstat** command times out.)
16 Mb	1500 8500	40 (Increase if the **nfsstat** command times out.) 40 (Increase if the **nfsstat** command times out.)
Ethernet	1500	40 (Increase if the **nfsstat** command times out.)

The larger MTU sizes for each token-ring speed reduce processor use and significantly improve read/write operations.

Setting MTU sizes:

To set MTU size, use the SMIT fast path, `smit chif`.

Select the appropriate adapter and enter an MTU value in the Maximum IP Packet Size field.

The **ifconfig** command can be used to set MTU size (and *must* be used to set MTU size at 8500). The format for the **ifconfig** command is:

`ifconfig tr`*n* *NodeName* `up mtu` *MTUSize*

where `tr`*n* is your adapter name, for example, `tr0`.

Another method of setting MTU sizes combines the **ifconfig** command with SMIT.

1. Add the **ifconfig** command for token rings, as illustrated in the previous example, to the `/etc/rc.bsdnet` file.
2. Enter the `smit setbootup_option` fast path. Toggle the **Use BSD Style** field to **yes**.

Transmit queue sizes:

Communication adapter transmit queue sizes are set with SMIT.

Enter the `smit chgtok` fast path, select the appropriate adapter, and enter a queue size in the Transmit field.

Hung programs:

If programs hang during file-related work, the NFS server could have stopped.

In this case, the following error message may be displayed:

```
NFS server hostname not responding, still trying
```

The NFS server (*hostname*) is down. This indicates a problem with the NFS server, the network connection, or the NIS server.

Check the servers from which you have mounted file systems if your machine hangs completely. If one or more of them is down, do not be concerned. When the server comes back up, your programs continue automatically. No files are destroyed.

If a soft-mounted server dies, other work is not affected. Programs that time out while trying to access soft-mounted remote files fail with the `errno` message, but you are still able to access your other file systems.

If all servers are running, determine whether others who are using the same servers are having trouble. More than one machine having service problems indicates a problem with the **nfsd** daemons on the server. In this case, log in to the server and run the **ps** command to see if the **nfsd** daemon is running and accumulating CPU time. If not, you might be able to stop and then restart the **nfsd** daemon. If this does not work, you have to restart the server.

Check your network connection and the connection of the server if other systems seem to be up and running.

Permissions and authentication schemes:

Sometimes, after mounts have been successfully established, there are problems in reading, writing, or creating remote files or directories. Such difficulties are usually due to permissions or authentication problems.

Permission and authentication problems can vary in cause depending on whether NIS is being used and secure mounts are specified.

The simplest case occurs when nonsecure mounts are specified and NIS is not used. In this case, user IDs (UIDs) and group IDs (GIDs) are mapped solely through the server `/etc/passwd` file and client `/etc/group` file. In this scheme, for a user named `B` to be identified both on the client and on the server as `B`, the user `B` must have the same UID number in the `/etc/passwd` file. The following is an example of how this might cause problems:

```
User B is uid 200 on client foo.
User B is uid 250 on server bar.
User G is uid 200 on server bar.
```

The /home/bar directory is mounted from server bar onto client foo. If user B is editing files on the /home/bar remote file system on client foo, confusion results when he saves files.

The server bar thinks the files belong to user G, because G is UID 200 on bar. If B logs on directly to bar by using the **rlogin** command, he may not be able to access the files he just created while working on the remotely mounted file system. G, however, is able to do so because the machines arbitrate permissions by UID, not by name.

The only permanent solution to this is to reassign consistent UIDs on the two machines. For example, give B UID 200 on server bar or 250 on client foo. The files owned by B would then need to have the **chown** command run against them to make them match the new ID on the appropriate machine.

Because of the problems with maintaining consistent UID and GID mappings on all machines in a network, NIS is often used to perform the appropriate mappings so that this type of problem is avoided.

Host name resolution on an NFS server:

When an NFS server services a mount request, it looks up the name of the client making the request. The server takes the client Internet Protocol (IP) address and looks up the corresponding host name that matches that address.

After the host name has been found, the server looks at the exports list for the requested directory and checks the existence of the client name in the access list for the directory. If an entry exists for the client and the entry matches exactly what was returned for the name resolution, then that part of the mount authentication passes.

If the server is not able to perform the IP address-to-host-name resolution, the server denies the mount request. The server must be able to find some match for the client IP address making the mount request. If the directory is exported with the access being to all clients, the server still must be able to do the reverse name lookup to allow the mount request.

The server also must be able to look up the correct name for the client. For example, if there exists an entry in the /etc/exports file like the following:

```
/tmp      -access=silly:funny
```

the following corresponding entries exist in the /etc/hosts file:

```
150.102.23.21        silly.domain.name.com
150.102.23.52        funny.domain.name.com
```

Notice that the names do not correspond exactly. When the server looks up the IP address-to-host-name matches for the hosts silly and funny, the string names do not match exactly with the entries in the access list of the export. This type of name resolution problem usually occurs when using the **named** daemon for name resolution. Most **named** daemon databases have aliases for the full domain names of hosts so that users do not have to enter full names when referring to hosts. Even though these host-name-to-IP address entries exist for the aliases, the reverse lookup might not exist. The database for reverse name lookup (IP address to host name) usually has entries containing the IP address and the full domain name (not the alias) of that host. Sometimes the export entries are created with the shorter alias name, causing problems when clients try to mount.

Limitations on the number of groups in the NFS structure:

On systems that use NFS Version 2 or 3, users cannot be a member of more than 16 groups without complications.

Groups are defined by the **groups** command. If a user is a member of 17 or more groups, and the user tries to access files owned by the 17th (or greater) group, the system does not allow the file to be read or copied. To permit the user access to the files, rearrange the group order.

The information above describes default behavior. Please see the **maxgroups** parameter of the **mount** command for more details.

NFS servers with earlier versions of NFS:

An NFS Version 3 client cannot mount to an NFS Version 4 server.

When mounting a file system from a pre-Version 3 NFS server onto a version 3 NFS client, a problem occurs when the user on the client executing the mount is a member of more than eight groups. Some servers are not able to deal correctly with this situation and deny the request for the mount. The solution is to change the group membership of the user to a number less than eight and then retry the mount. The following error message is characteristic of this group problem:

```
RPC:    Authentication  error;  why=Invalid  client  credential
```

RPCSEC-GSS problem determination:

Consider the following solutions when you are having trouble with RPCSEC-GSS.

- Use the **klist** command on the client to make sure that you have valid, current credentials.
- Make sure that the clocks on the client, server, and KDC are in sync. It is recommended that NTP or equivalent setup be used to ensure a consistent time across the entire Kerberos realm.
- Make sure that the server has a valid `keytab` file and host principal. If the following command fails, the server will not work:

  ```
  kinit -kt 'tail -n 1 /etc/nfs/hostkey' 'head -n 1 /etc/nfs/hostkey'
  ```
- Ensure the **gssd** daemon is running and responsive on the client and the server with the following command:

  ```
  rpcinfo -u localhost 400234
  ```

  If the **gssd** daemon is not responding, RPCSEC-GSS will fail; stopping and restarting the **gssd** daemon may correct this problem.
- If you are getting write errors with integrity or privacy, make sure that you are using the kernel module. Integrity and privacy are not supported without the kernel module. (The kernel module is the Kerberos kernel module, `/usr/lib/drivers/nfs.ext`. It is installed with the `modcrypt.base` file set from the expansion pack.)
- If specific users are experiencing denials when accessing data they should have access to, verify that the involved principals in the KDC are properly synchronized with the user's AIX account name.
- Activate the system log. Most RPCSEC-GSS errors will be logged. The errors have two parts: the first is the GSS error code (see RFC 2744 for details), and the second is a Kerberos error code.

  Note: Activating the system log might affect system performance; therefore, the log should be deactivated after the problem determination is complete.
  Some common error codes and their solutions are as follows:

  KRB5_CC_NOTFOUND
  : Valid Kerberos credentials could not be found. The **kinit** command may fix this problem.

  KRB5_KDC_UNREACH
  : The KDC is unreachable. Make sure that the KDC is up and that there are no network problems between the client or server and the KDC.

  KRB5_KT_NOTFOUND
  : The `keytab` entry for your server principal was not found. Use the **nfshostkey -l** command to

make sure that you are using the correct principal (it should be nfs/*<fully qualified domain name>*) and keytab file. Use the **klist -ke** to check the server keytab file for the appropriate entry.

KRB5KRB_AP_ERR_TKT_NYV
Most likely indicates a clock problem

KRB5KRB_AP_WRONG_PRINC and KRB5KDC_ERR_S_PRINCIPAL_UNKNOWN
Both of these errors indicate that the principal that the client is using for the client does not match the server's host principal.

KRB5KRB_AP_WRONG_PRINC
Indicates that the client succeeded in resolving the server's host name to an existing principal of the form nfs/*<fully qualified domain name>*, but the server's host principal does not match this principal.

KRB5KDC_ERR_S_PRINCIPAL_UNKNOWN
Indicates that the client was unable to resolve the server's host name to an existing principal. Use the **nfshostkey -l** command to check the server to make sure that it has the correct principal. If it does, the client's host mapping table will probably need to be updated; see the **nfshostmap** command in *Commands Reference, Volume 4* for details.

EIM problem determination:

When troubleshooting EIM, consider the following tips.

Consider the following when you are having problems with EIM:

- If the **nfsrgyd** or **chnfsim** commands are unable to connect to the EIM LDAP server, ensure the **ibmslapd** process is running on the EIM LDAP server by typing the following command:
  ```
  ps -ef | grep ibmslapd
  ```
  If the **ibmslapd** process is not running, type the following command to activate it:
  ```
  /usr/sbin/ibmslapd
  ```
- If the **nfsrgyd** or **chnfsim** commands are able to connect to the EIM LDAP server but are unable to perform any identity mapping operations, ensure the **ibmslapd** process is not running in configuration-only mode. This can happen if the **ldapdb2** database is not running when the **ibmslapd** server is started. Follow these steps:
  1. Log in to the EIM LDAP server as the root user.
  2. View the /var/ldap/ibmslapd.log file. Check when the last time the **ibmslapd** process started. Also check whether the server was started in configuration-only mode because it couldn't connect to the **ldapdb2** database.

  If the server was not able to connect to the **ldapdb2** database, the database needs to be started. Follow these steps to start the **ldapdb2** database:
  1. Log into the EIM LDAP server as the root user.
  2. Type the following command to check if the **ibmslapd** process is active:
     ```
     ps -ef | grep ibmslapd
     ```
     If it is active, disable it by running the following command:
     ```
     kill ibmslapd pid
     ```
     where *pid* is the process ID that was returned from the **ps -ef** command.
  3. After the **ibmslapd** process is disabled, start the **ldapdb2** database:
     a. Log in to the EIM LDAP server as the **ldapdb2** user.
     b. Type db2start.

4. After the **ldapdb2** database is started, activate the **ibmslapd** process:
   a. Log in to the EIM LDAP server as the root user.
   b. Type `ibmslapd`.

Problems that occur if the NFS kernel extension is not loaded:

Some NFS commands do not run correctly if the NFS kernel extension is not loaded. Some commands with this dependency are: **nfsstat**, **exportfs**, **mountd**, **nfsd**, and **biod**.

When NFS is installed on the system, the kernel extension is placed in the `/usr/lib/drivers/nfs.ext` file. This file is then loaded as the NFS kernel extension when the system is configured. The script that does this kernel extension loads the `/etc/rc.net` file. There are other things done in this script, one of which is to load the NFS kernel extension. It is important to note that **Transmission Control Protocol/Internet Protocol (TCP/IP)** kernel extension and the **nfs_kdes_null.ext** file should be loaded before the NFS kernel extension is loaded.

Note: The **gfsinstall** command is used to load the NFS kernel extension into the kernel when the system initially starts. This command can be run more than once per system startup and it will not cause a problem. The system is currently shipped with the **gfsinstall** command used in both the `/etc/rc.net` and `/etc/rc.nfs` files. There is no need to remove either of these calls.

Problems that occur if kerberos support is not installed:

If kerberos support is not installed, the **gssd** daemon will not start.

Ensure that the `krb5.client.rte` and `modcrypt.base` filesets are installed. If either is not installed, the **gssd** daemon will not run.

Items to check if the registry daemon is not running:

The **nfsrgyd** daemon will not run if the NFS version 4 domain has not been configured.

For information on configuring the NFS version 4 domain, see "/etc/nfs/local_domain file" on page 486.

NFS files

The NFS files and their descriptions can be referenced here.

Item	Description
`bootparams`	Lists clients that diskless clients can use for booting.
`exports`	Lists the directories that can be exported to NFS clients.
`filesystems`	Lists all the file systems that are mounted at system restart.
`hostkey`	Specifies the Kerberos host principal and the location of the `keytab` file.
`local_domain`	Contains the local NFS domain of the system.
`networks`	Contains information about networks on the Internet network.
`pcnfsd.conf`	Provides configuration options for the **rpc.pcnfsd** daemon.
`princmap`	Maps host names to Kerberos principals when the principal is not the fully qualified domain name of the server.
`realm.map`	Used by the NFS registry daemon to map incoming Kerberos principals.
`rpc`	Contains database information for Remote Procedure Call (RPC) programs.
`security_default`	Contains the NFS security defaults.
`xtab`	Lists directories that are currently exported.

NFS commands

The NFS commands and their descriptions can be referenced here.

Item	Description
chnfs	Starts a specified number of **biod** and **nfsd** daemons.
chnfsdom	Changes the local NFS domain.
chnfsim	Changes NFS foreign identity mappings.
chnfssec	Changes the default security flavor used by the NFS client
chnfsrtd	Changes the local NFS realm-to-domain mappings.
mknfs	Configures the system to run NFS and starts NFS daemons.
nfso	Configures NFS network options.
automount	Mounts an NFS file system automatically.
chnfsexp	Changes the attributes of an NFS-exported directory.
chnfsmnt	Changes the attributes of an NFS-mounted directory.
exportfs	Exports and unexports directories to NFS clients.
lsnfsexp	Displays the characteristics of directories that are exported with NFS.
lsnfsmnt	Displays the characteristics of mounted NFS systems.
mknfsexp	Exports a directory using NFS.
mknfsmnt	Mounts a directory using NFS.
nfshostkey	Configure the host key for an NFS server.
nfs4cl	Displays information about filesystems a client is accessing using NFS version 4.
nfs4smctl	Administers revocation of NFS version 4 State
rmnfs	Stops the NFS daemons.
rmnfsexp	Removes NFS-exported directories from a server's list of exports.
rmnfsmnt	Removes NFS-mounted file systems from a client's list of mounts.

NFS daemons

The NFS daemons and their descriptions can be referenced here.

Locking daemons

Item	Description
lockd	Processes lock requests through the RPC package.
statd	Provides crash-and-recovery functions for the locking services on NFS.

Network service daemons and utilities

Item	Description
biod	Sends the client read and write requests to the server.
mountd	Answers requests from clients for file system mounts.
nfsrgyd	Performs translation between security principals, NFS version 4 identity strings, and their corresponding numeric system IDs. In addition, mapping of identity information from foreign NFS version 4 domains is provided.
nfsd	Starts the daemons that handle a client requests for file system operations.
nfsstat	Displays information about the ability to receive calls for a particular machine.
on	Executes commands on remote machines.
pcnfsd	Handles service requests from PC-NFS clients.
portmap	Maps RPC program numbers to Internet port numbers.
rexd	Accepts request to run programs from remote machines.
rpcgen	Generates C code to implement an RPC protocol.
rpcinfo	Reports the status of RPC servers.
rstatd	Returns performance statistics obtained from the kernel.
rup	Shows the status of a remote host on the local network.
rusers	Reports a list of users logged on to the remote machines.
rusersd	Responds to queries from the **rusers** command.
rwall	Sends messages to all users on the network.
rwalld	Handles requests from the **rwall** command.
showmount	Displays a list of all clients that have mounted remote file systems.
spray	Sends a specified number of packets to a host.
sprayd	Receives packets sent by the **spray** command.

Secure networking daemons and utilities

Item	Description
chkey	Changes the user encryption key.
gssd	Provides NFS with access to security services provided by Network Authentication Services.
keyenvoy	Provides an intermediary between user processes and the key server.
keylogin	Decrypts and stores the user secret key.
keyserv	Stores public and private keys.
mkkeyserv	Starts the **keyserv** daemon and uncomments the appropriate entries in the `/etc/rc.nfs` file.
newkey	Creates a new key in the `publickey` file.
rmkeyserv	Stops the **keyserv** daemon and comments the entry for the **keyserv** daemon in the `/etc/rc.nfs` file.
ypupdated	Updates information in Network Information Service (NIS) maps.

For additional information on NFS security, see Network File System security in *Security*.

Sun diskless client support

Item	Description
bootparamd	Provides information necessary for booting to diskless clients.

NFS subroutines

The NFS subroutines are described here.

Item	Description
cbc_crypt, **des_setparity**, or **ecb_crypt**	Implements Data Encryption Standard (DES) routines.

Server Message Block file system

Server Message Block Filesystem (SMBFS) allows access to shares on SMB servers as local filesystems on AIX.

In this filesystem, the user can create, delete, read, write, and modify the access times of files and directories. The owner or access mode of files and directories cannot be changed.

SMBFS can be used to access files on an SMB server. The SMB server is a server running Samba or a Windows XP, Windows NT, or Windows 2000 server or workstation. Each of these server types allows a directory to be exported as a share. This share can then be mounted on an AIX system using SMBFS.

SMBFS installation

To install SMBFS on an AIX system, install the `bos.cifs_fs` package.

When the `bos.cifs_fs` package in installed, the device `nsmb0` is created. This device allows the **mount** command to establish a connection between the SMB server and the client using SMBv1.

SMBFS mounting

The Server Message Block Filesystem (SMBFS) can be mounted in one of two ways.

It can be performed through the AIX **mount** command. For example:

```
mount -v cifs -n pezman/user1/pass1 -o uid=201,fmode=750 /home /mnt
```

For more information on the **mount** command and for explanations of the flags used, see the **mount** command in *Commands Reference, Volume 3*.

You can specify mounting options using the
`-o`

flag. Command line options should be separated only by a comma, not a comma and a space. The options for the filesystem are:

Item	Description
`fmode`	Sets file or directory to octal mode. Default value is 755.
`uid`	Assigns a UID to files during mount. The default is **root**.
`gid`	Assigns a GID to files during mount. The default is **system**.
`wrkgrp`	Workgroup to which the SMB server belongs.
`op`	Set value to 1 if using opportunistic locking. Set value to 0 if opportunistic locking is not used.
`opfs`	Name of cache filesystem to use for storing lock cache files.
`opsz`	Size of individual cache files used for opportunistic locking.
`opfssz`	Size of cache filesystem used in opportunistic locking.

You can also mount the filesystem by using the SMIT utility, `smit cifs_fs`, which runs the **mount** command after gathering all necessary information.

In order to mount an SMBFS filesystem, it is necessary to provide a user name and password to authenticate to the server. This user name and password are used to perform all necessary file operations on the server. The **Password** field in the smit panel is not marked as required. If the password field is not filled in, the `cifscred` file is searched for matching credentials for the user or server that is provided. If there is a match, the stored password from the `cifscred` file is used; otherwise, the user is prompted for a password through the standard AIX password prompt. This way, the user can provide a password without making it viewable.

Note: The password used for mounting SMBFS can be up to 14 character in length and the password can contain special characters.

Whenever a filesystem command, such as read, is invoked on a file inside the SMBFS mount point, a request is sent to the server to read the file. The user name and password are sent as part of this request so that the server can determine whether the user has permission on the server to perform a read operation on that file. Therefore, ultimate authority lies with the server as to whether an operation on a file is permissible.

However, the `fmode` option of the **mount** command provides a way for the root user on the client system to control access to the files on the server before the server is queried. If the `fmode` option is not provided by the user, the default is `755`. The following table illustrates how the `fmode` option works using a write request:

Table 92. Five cases in which users were either allowed or denied access based on permissions given.

Case number	User authenticated to server	User on client side wanting write access	Mount owner, group, and mode	Owner, group, and mode on server	Access allowed
Case 1	user1	user2	user1, staff rwxr-xr-x	user1, staff rwxrwxr-x	no
Case 2	user1	root	user1, staff rwxr-xr-x	user2, staff rwxr-xr-x	no

Table 92. Five cases in which users were either allowed or denied access based on permissions given. (continued)

Case number	User authenticated to server	User on client side wanting write access	Mount owner, group, and mode	Owner, group, and mode on server	Access allowed
Case 3	user1	user1	user1, staff rwxr-xr-x	user2, staff rwxrwxr-x	yes
Case 4	user1	user1	user, staff rwxr-xr-x	root, system rwx------	no
Case 5	user1	user1	user1, staff rwxr-xr-x	root, system rwxrwxrwx	yes

In Case 1, access is denied because owner, group, and mode at mount on client did not allow write access to `user2`.

In Case 2, access is denied because, even though `root` has access to everything on the client side, the server-authenticated user, `user1`, does not have access to the file on the server.

In Case 3, access is granted because `user1` was the owner at mount, and `user1`, being a member of group `staff` on the server, had access to the file on the server.

In Case 4, access is denied because, even though `user1` was the owner at mount, the file is owned by `root` on the server, with no access by group or other.

In Case 5, access is granted because `user1` was the owner at mount, and `user1` had access to the file on the server through other permissions.

Notes:

1. The AIX SMBFS client supports only SMBv1.
2. On the mounted file system, a copy operation of one file to another is successful for a file of size 4 GB + 4096 bytes or less. For files beyond this size, a warning message is printed and 4 GB + 4096 bytes of the original file are copied to the destination.
3. On the mounted file system, the following characters cannot be used in a file name : Backslash key {\}, Forwardslash key {/}, Colon {:}, asterisk {*}, Questionmark {?}, Less Than key {< }, Greater Than Key {>}, Vertical Bar Key { | }.

Stored passwords

SMBFS can store `server/user/password` credentials in the `/etc/cifs_fs/cifscred` file to allow automatic retrieval of passwords when mounting SMBFS.

Credentials can be added, changed, and removed from this file with the **mkcifscred**, **chcifscred**, and **rmcifscred** commands (located in the `/usr/sbin` file). Passwords added to this file are encrypted. When mounting is attempted without providing a password, the `cifscred` file is searched for matching credentials. If there is a match, the stored password from the `cifscred` file is used; otherwise, the user is prompted for a password through the standard AIX password prompt.

Support for stored passwords comes with the following limitations:

- For stored password retrieval to work properly, the server naming convention must be consistent. For example, if the credentials are added with an IP address rather than a host name or a fully qualified domain name (FQDN), passwords will only be retrieved when mounting by IP address.

- Plain text password authentication is not supported with the stored password retrieval method. If the server requires plain text passwords, authentication fails.

/etc/filesystems support

SMBFS supports `/etc/filesystems` to allow automated mounting at system startup.

Support for `/etc/filesystems` also provides access to stored server, user, password, and options data when mounting. Use the **mkcifsmnt**, **chcifsmnt**, **rmcifsmnt**, and **lscifsmnt** commands (located in `/usr/sbin`) to add, change, remove, and list, respectively, cifs stanzas in `/etc/filesystems`. Credentials must be stored in the `cifscred` file.

Troubleshooting SMBFS

Follow these steps when encountering problems with SMBFS.

If the **mount** command or `smit cifs_fs` fastpath returns an error, consider the following:

1. Make sure the user name and password are correct. The user name and password need to allow access to the share on the server.
2. Make sure that the server name is correct. If the server name is correct, use the fully qualified host name in case the server is not part of the same subnet as the client. You can also try using the server IP address.
3. Make sure that the `lsdev -L|grep nsmb` command returns a device name. If an `nsmb` device is not available, then the AIX client will not be able to establish a connection to the SMB server.
4. Make sure that the share name is correct. If the share does not exist on the server or cannot be accessed with the user name and password given, the SMB server will reject the connection request.
5. Use event ID 525 to collect system trace data for SMBFS.
6. Ensure that the server is configured to accept NTLM, LM, or plain text passwords. These are the only types of password encryption supported by SMBFS.
7. If you want authentication against a domain, the domain name must be specified with the **wrkgrp** option. Without this option, authentication is handled locally by the server.

Asynchronous communications

AIX provides the following categories of asynchronous device drivers, also called tty device drivers:

- Drivers for the serial ports on the system planar
- Drivers for the serial ports connected to the system through an adapter
- Pseudo-tty drivers

Drivers in the first category are the PCI adapters. They include 2-port, 8-port and 128-port adapters.

In the second category, the 8-port and 128-port PCI adapters are called intelligent adapters, because they use an Intel 8086 processor to offload much of the character processing from the host CPU. These adapters are driven by a 20 ms poller instead of hardware interrupts and provide performance characteristics well suited for the majority of serial devices and applications. As more devices are added to the system, the system workload goes up very little, with the result that these adapters can support a very large number of serial devices, many more than would be possible using hardware interrupts. Also, because these adapters use a patented software performance enhancement, they can send and receive large amounts of data faster and more efficiently than the native system ports, as long as the data is being moved in large blocks. For more information, see the wantio description in the `/usr/include/sys/pse/README.pse` file.

Note: Integrated POWER5 *system ports* are similar to serial ports except that system ports are available only for specifically supported functions. Refer to "Functional differences between system ports and serial ports" on page 542 for more information.

However, some devices and applications expect or require very low latency in single character processing, so you may experience timing problems when connected to these intelligent adapters. Character latency, or character echo, may be defined as the time it takes to receive a single character on a serial port, deliver that character to an application, then echo the character back out the same serial port.

Because they use the highest priority interrupt on the system (INTCLASS0), interrupt-driven ports provide latency values in the 0.10 to 0.20 ms range on an idle system. The 8-port PCI adapters provide latency values averaging around 10 to 12 ms, with individual times varying by plus or minus 10 ms due to the 20 ms poller. The 128-port PCI adapters have the same 20 ms poller, which communicates over a polled communications link to the Remote Access Nodes (RANs). RANs allow a polling driver to control the serial ports. Latency values on these ports average around 30 ms, but can exceed 60 ms.

Latency values on the 8-port PCI and 128-port PCI adapters can be tuned for special applications using the "event delay" (EDELAY) parameter. For maximum responsiveness when receiving a single character, reduce the value of the EDELAY parameter. This minimizes the time required to get a single character from the serial port to the application, but can result in reduced throughput and overall system performance when multiple characters are received in a burst.

The 2-port PCI EIA-32 adapter is an asynchronous serial communications adapter, based on the Exar 17D152 Universal PC Dual UART. The 2-port adapter supports two DB-9 connectors and provides connectivity to asynchronous EIA-32 devices, such as modems and tty terminals.

On the IBM eServer™ p5 platform there are no native system ports available to AIX. Although the virtual terminal interface is enhanced to support physical serial ports located on the FSP through the hypervisor, this interface only supports a specific set of serial devices and is not a suitable replacement for a general-purpose physical serial port. The 2-port adapter behaves somewhat like a native system port. The adapter device driver is interrupt-driven and supports programmable transit and receives FIFO trigger levels. It is a PCI adapter; therefore, the device driver supports EEH, hot-plug, and VPD queries. The 2-port adapter does not support native system port features for when the virtual terminal is used, such as during boot, install, and KDB support.

Pseudo-tty drivers are used when accessing a system over a network using the **rlogin** or **telnet** commands or when accessing a system using a windowing system on a graphics monitor. The pseudo-tty driver provides a means of running legacy character-based applications, such as the **vi** text editor, over communications media other than serial. The important thing to note about pseudo-tty drivers is that they are not symmetrical. The slave end provides a POSIX-standard-compliant interface to earlier applications. The master end is controlled by an entity such as the **rlogin** or **telnet** daemon or X-windows, which must provide an emulation of a serial terminal device to the pseudo-tty driver. AIX can efficiently support very large numbers of pseudo-tty devices.

Non-POSIX line speeds

The interface to serial devices specified by POSIX and subsequent UNIX standards such as X/OPEN relies on the **termios** data structure, defined in `/usr/include/termios.h`. Unfortunately, that data structure cannot be used to specify line speeds in excess of 38,400 bits per second. Most serial hardware currently in use can support speeds of up to 230,000 bps. To use these higher line speeds on AIX, the desired speed should be specified when configuring the port using SMIT. If the serial port hardware (UART) can support your specified line speed, the port can be configured.

Get attribute ioctls using the **termio** or **termios** structure will report the line speed as 50 bps. The non-POSIX line speed will be used by the port until changed, so applications using set attribute ioctls

with the **termio** and **termios** structure should not modify the CBAUD flags unless they really do intend to change the line speed. If the serial port hardware (UART) cannot support the requested line speed, the port fails to configure and an error is returned.

Note: The 8- and 128-port PCI adapters support non-POSIX line speeds of 115,200 and 230,000 bps only. The 128-port PCI adapter has an additional constraint of 2.5 Mbps aggregate bandwidth (with the 8-wire cable), which would be completely consumed by 11 devices transferring at a sustained 230,000 bps each. This constraint is on the line connecting the adapter to the RANs, so a single adapter can be completely consumed by 22 such devices.

Asynchronous adapters

Asynchronous communications products offer the advantages of low cost, multiuser, medium- to high-performance terminal and device communications.

AIX allows many users to access system resources and applications. Each user must be connected through a terminal session. The connection can be local or remote through a serial port.

Each system unit has at least one standard serial port available (some systems have three serial ports). These ports can support asynchronous communication and device attachment.

Asynchronous ports allow attachment of asynchronous peripheral devices that meet EIA 232, EIA 422, or RS-423 standards such as:

- Asynchronous modems
- Bar code scanners
- Graphic and character printers
- Keyboard and display terminals
- Personal computers
- Plotters and printers
- Point-of-sale terminals
- Sensors and control devices
- Text scanners
- Time clocks

Asynchronous communications options

Expanded asynchronous capability can be added to the system unit with adapters, by using Peripheral Component Interconnect (PCI) buses.

Several factors might influence the type of asynchronous connectivity you choose. The following table summarizes these products.

Table 93. Asynchronous communication products

Asynchronous attachment	Based on POWER® processor based	Itanium based	Bus type	Feature code or machine type (model)	Maximum data rate per port (kbps)	Salient features
Standard serial port	X	X	System planar	n/a	Selectable based on baud rate generator clock speed of universal asynchronous receiver and transmitter (UART).	Standard feature
232 RAN	X	X		8130	57.6	Remote capability

Table 93. Asynchronous communication products (continued)

Asynchronous attachment	Based on POWER® processor based	Itanium based	Bus type	Feature code or machine type (model)	Maximum data rate per port (kbps)	Salient features
Enhanced 232 RAN	X	X		8137	230	Remote capability
16-Port RAN EIA 422	X	X		8138	230	Remote capability
128-Port Controller	X			8128	230	Efficiency, higher device counts
128-Port Controller	X			2933	230	Efficiency, higher device counts
128-Port Controller	X	X	PCI	2944	230	Efficiency, higher device counts

Note: Rack Mount RAN FC is 8136.

Note: Maximum Data Rate per Port is limited by line bandwidth (1.2 Mbps for standard RAN, or 2.4 Mbps for Enhanced RAN).

The first feature in this table represents the planar-attached serial ports that are standard with every system unit. The next features are the adapters. The 128-port asynchronous subsystem includes the remote asynchronous nodes (RANs) that attach to it.

Planar-attached asynchronous ports

Most system unit models have two integrated (standard) EIA 232 asynchronous serial ports. EIA 232 asynchronous serial devices can be attached directly to the standard serial ports using standard serial cables with 9-pin D-shell connectors.

Some multiprocessor systems have a third serial port used for communication to the remote service center.

Note: Itanium-based systems have one or two integrated serial ports. Initial workstation models have one port, while the initial server class models have two ports.

Adapter-attached asynchronous ports

Each of the adapters requires a bus slot and can only be used in systems that support the required bus type.

The 128-port, ISA 8-port adapters, and PCI 8-port adapters are intelligent adapters that provide significant offload of the main system processor.

EIA 232 is the most common communication standard, but the EIA 422A (used when a longer cable distance is needed) is also supported. The EIA 422A implementation does not include device status detection capability or RS 232 modem control signals.

Note: The Itanium-based platform supports only the 8- and 128-port PCI adapters.

Node-attached asynchronous ports

The 128-port adapter, available for the Micro Channel, ISA, or PCI bus, allows attachments of one to eight remote asynchronous nodes (RANs).

Each RAN has 16 asynchronous ports for connection to devices and are separately powered units. Up to four RANs can be daisy-chain-connected from each of two connections on the 128-port adapter card.

RANs can support 16 EIA 232 devices or 16 EIA 422 devices. The 128-port controller is an intelligent adapter that increases the number of asynchronous sessions possible at a given CPU usage level.

The following are additional characteristics of the 128-port feature:

- RANs may be located up to 300 meters from the system processor using 8-wire shielded cabling while maintaining full performance ratings.
- Distance may be extended to 1200 meters by reducing the data rate between the RANs and the system processor.
- RANs may be remotely located from the system processor using a synchronous EIA 232 and EIA 422 modems. Each four-RAN daisy chain is allowed only one modem pair at some point in the chain.
- System performance is enhanced by offloading tty character processing from the system processor.

Product selection considerations

The proper asynchronous product often depends on a particular situation.

The following questions will help you choose which product you need to install.

Expandability

How many asynchronous ports are needed?

How may ports will be needed in the future?

Topology

Will devices be in other buildings or remote locations?

Where will system/network administration be done?

Is there an HACMP™ cluster?

What type of cabling is required or already there?

Performance

Is your application CPU-intensive?

What types of devices will be attached?

What is the relative asynchronous bandwidth demand for the aggregate sum of the devices?

Table 94. Relative Device Bandwidth Demand

Low Demand	Moderate Demand	High Demand
ASCII terminals, Point-of-sale terminals, Asynchronous modems	Printers, Low-speed FAX/modems, Bar code scanners	Serial X-terminals, High-speed FAX/modems, High-speed printers, File transfer applications

Device Interface Requirement

What asynchronous interface is required, for example, EIA 232, EIA 422A, EIA 423?

Do the devices or applications require the full EIA 232 interface?

Security

Is system assurance kernel (SAK) required? (planar-attached ports only)

The following table shows the detailed product characteristics.

Table 95. Asynchronous attachment product characteristics

Characteristic	Native Serial Ports	2-port PCI	8-port PCI	128-port PCI with RAN
Number of asynchronous ports per adapter	n/a	2	8	128
Maximum number of adapters	n/a	no limit	20	20
Maximum number of asynchronous ports	2 or 3	2	160	2560
Number of asynchronous ports per RAN	n/a	n/a	n/a	16
Maximum number of RANs	n/a	n/a	n/a	160
Maximum speed (KBits/sec)	Selectable based on baud rate generator clock speed of UART.	230	230	230
Attachment method	planar	direct	direct	node
Asynchronous electrical interfaces supported	EIA 232	EIA 232	EIA 232 EIA 422A	EIA 232 EIA 422
Standard connector	DB9	DB9	DB25M	RJ-45 (10-pin or 8-pin)
DB25 cable options	n/a	n/a	n/a	RJ-45-DB25
Rack mount option	n/a	n/a	n/a	yes
Power supply	n/a	n/a	n/a	external
Signals supported (EIA 232)	TxD RxD RTS CTS DTR DSR DCD RI	TxD RxD RTS CTS DTR DSR DCD RI	TxD RxD RTS CTS DTR DSR DCD RI	TxD RxD RTS CTS DTR DSR DCD RI

Asynchronous adapter applications

Each product offering is characterized by a representative scenario for its strengths. The adapters in this topic are listed along with their specifications so that you can select for each specific scenario.

Item	Description
2-Port PCI Bus EIA 232	• PCI slot available. • Up to two ports per adapter. • Requires all EIA 232 ports. • Asynchronous speeds to 230 Kbps.
8-Port PCI Bus EIA 232/EIA 422	• PCI slot available. • Fewer than eight ports required with little or no expansion. • Requires all EIA 232, all EIA 422, or a mix of EIA 232 and EIA 422 ports. • Offload character interrupt and terminal I/O processing from the main CPU. • Asynchronous speeds to 230 Kbps. • Maximum performance for high speed (33.6 Kbps) modems with data compression.

Item	Description
128-Port Adapter (PCI)	• A Micro Channel, ISA, or PCI bus slot available. (For further information about Micro Channel or ISA, see “128-Port adapter (Micro Channel, ISA)” on page 624.) • Sixteen ports now with expansion of up to 128 ports without additional slots. • Most distant terminal located about 90 meters (300 feet) from the system at maximum data rate of 230 Kbps. • Terminals planned: nearby or on premises, distant on premises, and remote. • Need high asynchronous throughput with low processor demand. • Need terminal attached printer capability. • Need to connect to remote premises through fiber-optic or synchronous modems.

Scenarios for asynchronous solutions

The customer scenarios that follow were solved with an 8-port PCI and a 128-port asynchronous controller.

Item	Description
Real estate office	• Simplicity and cost are high priority. • Operating system and server. • Six to ten devices tied into the server accessing the database. • One slot is available for asynchronous communication. • Devices are less than 61 meters (200 feet) from the server. **Solution:** 8-port PCI.
Retail point-of-sale	• Cost per seat is high priority. • Operating system and server. • 20 or more ASCII terminals: for example, cash registers. • One slot is available for asynchronous communication. • Future expansion for additional terminals is planned. **Solution:** 128-port asynchronous controller with two RANs. Future expansion with additional RANs.

Topology considerations

The asynchronous family of adapters offers a wide variety of choices where distance topology is concerned.

The maximum cable lengths from the planar- and direct-attach adapters is generally the distance between the port and the asynchronous device, operating at the maximum specified data rate. The 128-port adapter is measured from the adapter card to the daisy-chained RAN attached to it. With the 128-port, unlimited distances can effectively be achieved by using the EIA 422 synchronous modems to attach the RANs to the adapter.

Proper cabling is extremely important and is unique to each environment.

Serial communication

Asynchronous communication standards, hardware, terminology, and concepts are described here.

Serial ports are used to physically connect asynchronous devices to a computer. They are located on the back of the system unit, either integrated or using a multiport adapter, such as the 2-port, 8-port, 16-port, and 128-port asynchronous adapters.

Note: The POWER5 integrated system ports are not general-purpose, full-function serial ports. See "Functional differences between system ports and serial ports" on page 542 for more information.

To understand the functionality of a serial port, it is necessary to first examine parallel communications. A standard parallel port uses eight pins, or wires, to simultaneously transmit the data bits, making up a single character. The following illustration shows the parallel transmission of the letter a.

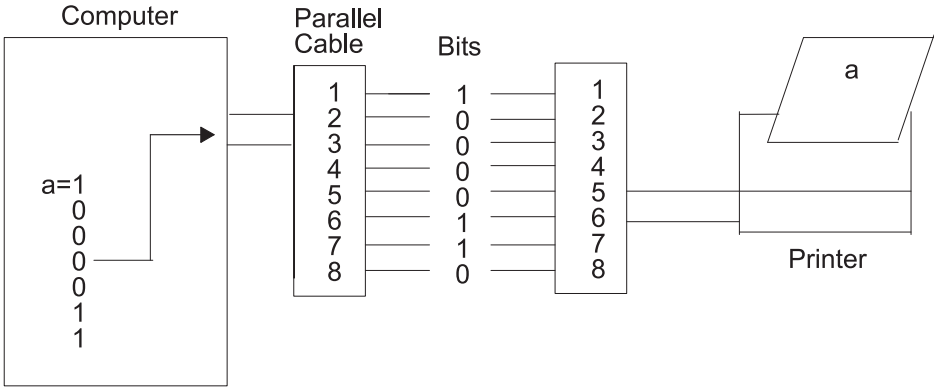


Figure 29. Parallel communications port

Serial ports require only a single pin, or wire, to send the same data character to the device. To accomplish this, the data is converted from a parallel form (sent by the computer), to a sequential form, where bits are organized one after the other in a series. The data is then transmitted to the device with the least significant bit (or zero-bit) sent first. After the data is received by the remote device, the data is converted back into parallel form. The following illustration shows the serial transmission of the letter a.

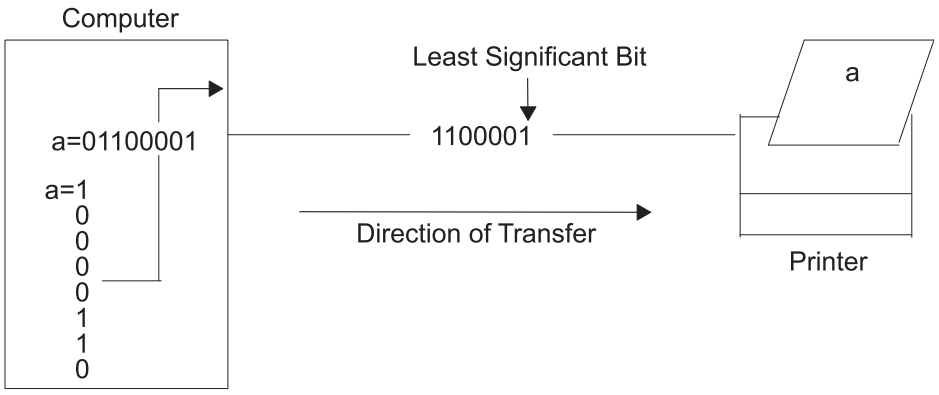


Figure 30. Serial communications port

Serial transmissions of a single character are simple and straight forward; however, complications arise when a large number of characters are transmitted in series as shown in the following illustration. The

receiving system does not know where one character ends and the other begins. To solve this problem, both ends of the communication link must be synchronized or timed.

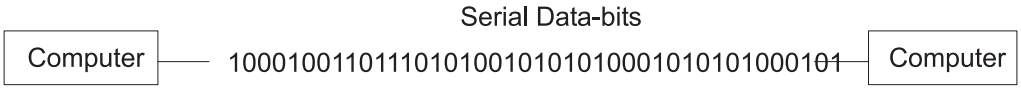


Figure 31. Serial transmission

Functional differences between system ports and serial ports

Integrated POWER5 *system ports* are similar to serial ports except that system ports are available only for specifically supported functions.

System ports are disabled when a Hardware Management Console (HMC) port is connected to an HMC. Either the HMC ports can be used or the system ports can be used, but not both.

Even when no HMC is attached, integrated system ports are limited to serial-connected TTY console function. They work properly only with approved call-home modems, async terminals, and certain UPSs. Attachment of any other serial devices (including system-to-system connections for HACMP) requires a serial port adapter in a PCI slot.

Synchronization

Synchronization is the process of timing the serial transmission to properly identify the data being sent.

The two most common modes are synchronous and asynchronous.

Synchronous transmission:

The term *synchronous* is used to describe a continuous and consistent timed transfer of data blocks.

These types of connections are used when large amounts of data must be transferred very quickly from one location to the other. The speed of the synchronous connection is attained by transferring data in large blocks instead of individual characters.

The data blocks are grouped and spaced in regular intervals and are preceded by special characters called `syn` or synchronous idle characters. See the following illustration.

Figure 32. Synchronous transmission

After the `syn` characters are received by the remote device, they are decoded and used to synchronize the connection. After the connection is correctly synchronized, data transmission may begin.

An analogy of this type of connection would be the transmission of a large text document. Before the document is transferred across the synchronous line, it is first broken into blocks of sentences or paragraphs. The blocks are then sent over the communication link to the remote site. With other transmission modes, the text is organized into long strings of letters (or characters) that make up the words within the sentences and paragraphs. These characters are sent over the communication link one at a time and reassembled at the remote location.

The timing needed for synchronous connections is obtained from the devices located on the communication link. All devices on the synchronous link must be set to the same clocking.

The following is a list of characteristics specific to synchronous communication:

- There are no gaps between characters being transmitted.
- Timing is supplied by modems or other devices at each end of the connection.
- Special syn characters precede the data being transmitted.
- The syn characters are used between blocks of data for timing purposes.

Asynchronous transmission:

The term *asynchronous* is used to describe the process where transmitted data is encoded with start and stop bits, specifying the beginning and end of each character.

An example of asynchronous transmission is shown in the following figure.

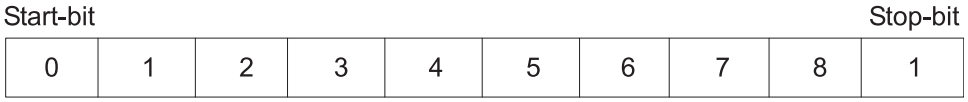


Figure 33. Asynchronous transmission

These additional bits provide the timing or synchronization for the connection by indicating when a complete character has been sent or received; thus, timing for each character begins with the start bit and ends with the stop bit.

When gaps appear between character transmissions, the asynchronous line is said to be in a mark state. A mark is a binary 1 (or negative voltage) that is sent during periods of inactivity on the line as shown in the following figure.

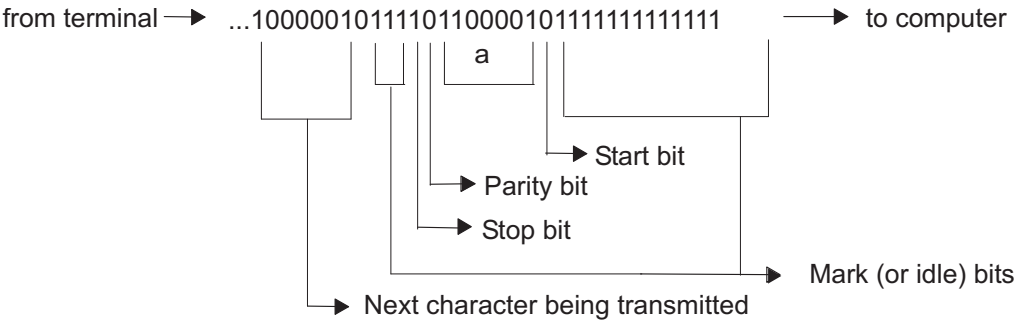


Figure 34. Mark (idle) bits in the data stream

When the mark state is interrupted by a positive voltage (a binary 0), the receiving system knows that data characters are going to follow. It is for this reason that the start bit, which precedes the data character, is always a space bit (binary 0) and that the stop bit, which signals the end of a character, is always a mark bit (binary 1).

The following is a list of characteristics specific to asynchronous communication:

- Each character is preceded by a start bit and followed by one or more stop bits.
- Gaps or spaces between characters may exist.

Serial communication parameters

Parameters used during serial communication include bits-per-character, bits-per-second (bps), baud rate, parity, and start, stop, and mark bits.

Bits-per-character:

The number of bits-per-character (bpc) indicates the number of bits used to represent a single data character during serial communication.

This number does not reflect the total amount of parity, stop, or start bits included with the character. Two possible settings for bpc are 7 and 8.

When using the seven bits-per-character setting, it is possible to only send the first 128 characters (0-127) of the Standard ASCII character set. Each of these characters is represented by seven data bits. The eight bits-per-character setting must be used to send the ASCII Extended character set (128-255). Each of these characters may only be represented using eight data bits.

Bits-per-second (bps):

This provides a description of the bits-per-second statistic.

Bits-per-second (bps) is the number of data bits (binary 1's and 0's) that are transmitted per second over the communication line.

Baud rate:

The baud rate is the number of times per second a serial communication signal changes states; a state being either a voltage level, a frequency, or a frequency phase angle.

If the signal changes once for each data bit, then one bps is equal to one baud. For example, a 300 baud modem changes its states 300 times a second.

Parity bits:

The parity bit, unlike the start and stop bits, is an optional parameter, used in serial communications to determine if the data character being transmitted is correctly received by the remote device.

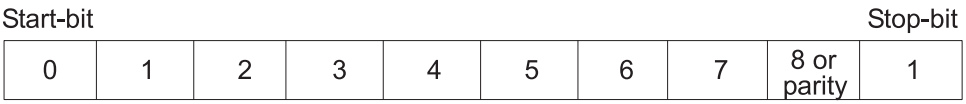


Figure 35. Parity

The parity bit can have one of the following five specifications:

Item	Description
none	Specifies that the local system must not create a parity bit for data characters being transmitted. It also indicates that the local system does not check for a parity bit in data received from a remote host.
even	Specifies that the total number of binary 1s, in a single character, adds up to an even number. If they do not, the parity bit must be a 1 to ensure that the total number of binary 1s is even. For example, if the letter a (binary 1100001) is transmitted under even parity, the sending system adds the number of binary 1s, which in this case is three, and makes the parity bit a 1 to maintain an even number of binary 1s. If the letter A (binary 1000001) is transmitted under the same circumstances, the parity bit would be a 0, thus keeping the total number of binary 1s an even number.
odd	Operates under the same guidelines as even parity except that the total number of binary 1s must be an odd number.
space	Specifies that the parity bit will always be a binary zero. Another term used for space parity is bit filling, which is derived from its use as a filler for seven-bit data being transmitted to a device which can only accept eight bit data. Such devices see the space parity bit as an additional data bit for the transmitted character.

Item	Description
mark	Operates under the same guidelines as space parity except that the parity bit is always a binary 1. The mark parity bit acts only as a filler.

Start, stop, and mark bits:

The start and stop bits are used in asynchronous communication as a means of timing or synchronizing the data characters being transmitted.

Without the use of these bits, the sending and receiving systems will not know where one character ends and another begins.

Another bit used to separate data characters during transmission is the mark (or idle) RS bit. This bit, a binary 1, is transmitted when the communication line is idle and no characters are being sent or received.

When a start bit (binary 0) is received by the system, it is understood that a fixed number character bit (determined by the **bits per character** parameter), and even a parity bit (determined by the **parity** parameter), follows the start bit. Then a stop bit (binary 1) is received by the system. In the following example, the **parity** bit is present, and the **bits per character** is 7.

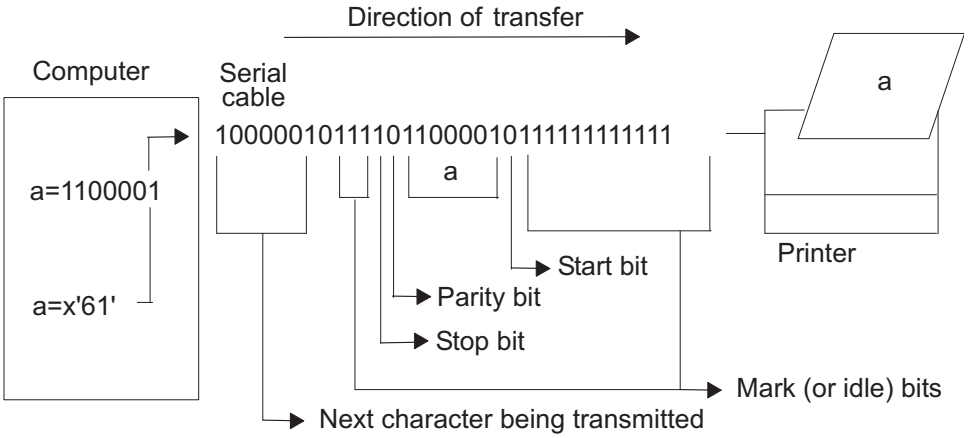


Figure 36. Start, stop, and mark bits

The EIA 232D standard

The EIA 232D standard was developed in 1969 to specify the connections between a computer and a modem.

The term itself is an acronym which can be read as follows:

Electronics Industry Association (EIA) accepted standard, ID number 232 revision D

EIA 232D specifies the characteristics of the physical and electrical connections between two devices. Names and abbreviations are assigned to each pin or wire necessary for serial communications, for example:

Table 96. EIA 232D connections

Signal	Equip. Type	Symbol	Pin
Transmit Data	DCE	TxD	2
Receive Data	DTE	RxD	3
Request to Send	DCE	RTS	4
Clear to Send	DTE	CTS	5
Data Set Ready	DTE	DSR	6
Signal Ground		SG	7
Carrier Detect	DTE	CD	8
Data Terminal Ready	DCE	DTR	20
Ring Indicator	DTE	RI	22

In EIA 232D, devices using pin 2 (TxD) for output (for example, computers and terminals) are given the name data terminal equipment (DTE). Devices using pin 2 (TxD) for input (for example, modems) are given the name data communication equipment (DCE).

EIA 232D also specifies the connectors. A DTE device normally has male connectors while DCE devices have female connectors. This standard is not always adhered to by manufacturers; therefore users should always review the device documentation before cable connection.

Asynchronous communication methods

This describes the two forms of asynchronous communication, one-way and two-way (which includes half-duplex and full-duplex).

Simplex, or one-way communication, is the simplest connection form between two devices. This mode of communication allows data to be transmitted in only one direction and requires only two lines be connected, for example, TxD (or RxD) and SG.

There are two forms of two-way communications: half-duplex and full-duplex. A connection in half-duplex mode allows data to be transmitted in two directions but not simultaneously. An analogy of half-duplex would be the use of a CB-radio where two-way communication is possible but only one person can speak at a time.

In full-duplex, or duplex mode, data communication can take place in two directions simultaneously. An analogy for full-duplex is a telephone conversation when both persons are talking at the same time.

Flow control

Some type of data flow control is needed by the serial device to limit the amount of data transmitted by the system.

Serial devices, such as printers and modems, do not process data as quickly or efficiently as the computers they are connected to.

The term *flow control* is used to describe the method in which a serial device controls the amount of data being transmitted to itself.

RTS/CTS hardware flow:

Request to send/clear to send (RTS/CTS) is sometimes called pacing or hardware handshaking instead of flow control.

The term hardware handshaking comes from the use of cabling and voltages as a method of data transmission control. Unlike XON/XOFF, which sends control characters in the data stream, RTS/CTS uses positive and negative voltages along dedicated pins or wires in the device cabling.

A positive voltage means data transmission is allowed while a negative voltage signifies that data transmission should be suspended.

DTR/DSR hardware flow:

Data terminal ready (DTR), another form of hardware flow control, is normally generated by the devices, such as printers to indicate that they are ready to communicate with the system. This signal is used in conjunction with data set ready (DSR) generated by the system to control data flow.

A positive voltage means data transmission is allowed while a negative voltage signifies that data transmission should be suspended.

XON/XOFF software flow:

Transmitter on/transmitter off (XON/XOFF) flow controls involves the sending of data transmission control characters along the data stream (TxD and RxD). For this reason it is referred to as software flow control.

When data is sent to a modem, it is placed in a buffer. Just before that buffer reaches its maximum capacity, the modem will send an XOFF character to the system and the system will stop transmitting the data. When the modem's buffer is almost empty and ready for more data, it will send an XON character back to the system causing more data to be sent.

Configuring a port for RTS/CTS hardware handshaking:

Modems attached to the server operating at a speed of 9600 or above are recommended to use RTS/CTS hardware handshaking instead of XON/XOFF flow control.

This will avoid buffer overrun in a system with limited resources. RTS is not a default value on any tty port and must be set by the system administrator accordingly.

`Prerequisites`

You must use at least a five-wire cable to support RTS/CTS.

To enable RTS/CTS for a port, use the following steps:

1. Use the `smit tty` fast path.
2. Select **Change / Show Characteristics of a TTY**.
3. Select the tty on which RTS/CTS is to be enabled.
4. Set the FLOW CONTROL to be used field to **rts**.
5. Select **Do**.
6. Exit SMIT.

TTY terminal device

A tty terminal device is a character device that performs input and output on a character-by-character basis.

The communication between terminal devices and the programs that read and write to them is controlled by the tty interface. Examples of tty devices are:

- Modems
- ASCII terminals

- System console (LFT)
- **aixterm** under AIXwindows

The tty devices can be added, deleted, listed, and changed like any other device on your system by using the SMIT tool, or device-specific commands.

TERM values for different displays and terminals

Information about terminal capabilities is stored in the `terminfo` database.

The value of the **TERM** environment variable identifies the specific terminal description in the `terminfo` database. This provides all information that a program needs for communicating effectively with the current tty device.

Table 97. TERM values for various terminals

Display/Terminal	Value
3161 ASCII Terminal	ibm3161
3163 ASCII Terminal	ibm3161
DEC VT100 (terminal)	vt100
DECVT220	vt220
3151 ASCII Display Station with Cartridge or 3161 ASCII Display Station with Cartridge	ibm3161-C
3162 ASCII Display Station	ibm3161
3162 ASCII Display Station with Cartridge	ibm3162
6091 Display	lft
AIXwindows	aixterm

For information about the entries in the `terminfo` database, see the `terminfo` file format in *Files Reference*. To convert `termcap` entries into `terminfo` entries, see the **captoinfo** command in *Commands Reference, Volume 1*. (The `termcap` file contains the terminal descriptions for older Berkeley systems.)

TTY characteristics

The *line discipline* provides the hardware-independent user interface for communicating between the computer and an asynchronous device.

For example, a user is able to erase a single line or to interrupt a currently running process by typing a particular sequence of characters. You can define the meaning of these character sequences as well as set other terminal characteristics, such as the communication speed, by using the **chdev** command, the System Management Interface Tool (SMIT), or the **stty** command.

Attribute requirements on the attached TTY device

A correct communication between the host and an attached tty device must have these requirements.

- A properly wired communications cable
- Matching communications values (line speed, character size, parity, stop bit, and interface) between the host and the attached tty device.

TTY device management

Device management tasks and their associated SMIT fast paths and commands can be referenced here.

Table 98. Managing TTY devices tasks

Task	SMIT fast path	Command or file
List Defined TTY Devices	`smit lsdtty`	**lsdev -C -c tty -H**
Add a TTY	`smit mktty`	**mkdev -t tty**[1,2]
Move a TTY to Another Port[3]	`smit movtty`	**chdev -l** *Name* **-p** *ParentName* **-w** *ConnectionLocation*[2,4]
Change/Show Characteristics of a TTY	`smit chtty`	**lsattr -l***Name* **-E** (to show); **chdev -l** *Name* (to change)[4,5]
Remove a TTY[3]	`smit rmtty`	**rmdev -l** *Name*
Configure a Defined TTY (Make Available for Use)	`smit mktty`	**mkdev -l** *Name*

Note:

1. Other flags may be used to further specify the new tty device. For example, to define and configure an RS-232 tty device connected to port `0` on the 8-port asynchronous adapter `sa3` with the `speed` attribute set to `19200` and other attributes set to values retrieved from the `foo` file:

   ```
   mkdev -t tty -s rs232 -p sa3 -w 0 -a speed=19200 -f foo
   ```

2. The **mkdev** and **chdev** commands support options that are not possible with SMIT.
3. Disable the tty before doing this task. See the **pdisable** command in *Commands Reference, Volume 4*.
4. Use flags to change specific characteristics about a tty from the command line.
5. You can select a Posix baud rate from the List function, or you can type in non-Posix baud rate directly into the field. If the selected baud rate cannot be supported by the modem's hardware, the system displays an error message.

If adding or changing a tty from the command line, consult the following list to find out the *Attribute* name to specify in the **-a** *Attribute*=*Value* flag for the characteristic you want to set. For example, specify `-a speed=Value` to set the baud rate of a tty device.

Table 99. TTY attributes

Characteristic	Attribute Name
Enable LOGIN	login
BAUD rate speed	speed
PARITY	parity
BITS per character	bpc
Number of STOP BITS	stops
TIME before advancing to next port setting	timeout
XON-XOFF handshaking	xon
TERMINAL type	term
FLOW CONTROL to be used	flow_disp
OPEN DISCIPLINE to be used	open_disp
STTY attributes for RUN time	runmodes
STTY attributes for LOGIN	logmodes
RUN shell activity manager	shell
LOGGER name	logger
STATUS of device at BOOT time	autoconfig
TRANSMIT buffer count	tbc
RECEIVE trigger level	rtrig
STREAMS modules to be pushed at open time	modules
INPUT map file	imap

Table 99. TTY attributes (continued)

Characteristic	Attribute Name
OUTPUT map file	omap
CODESET map file	csmap
INTERRUPT character	intr
QUIT character	quit
ERASE character	erase
KILL character	kill
END OF FILE character	eof
END OF LINE character	eol
2nd END OF LINE character	eol2
DELAY SUSPEND PROCESS character	dsusp
SUSPEND PROCESS character	susp
LITERAL NEXT character	lnext
START character	start
STOP character	stop
WORD ERASE character	werase
REPRINT LINE character	reprint
DISCARD character	discard

TTY troubleshooting

There are several common TTY troubleshooting scenarios.

Common TTY troubleshooting scenarios include `Respawning Too Rapidly` errors, hung TTY ports, and common error logging files, commands, and error report messages.

Respawning Too Rapidly errors:

The system records the number of **getty** processes created for a particular tty in a short time period. If the number of **getty** processes created in this time frame exceeds five, then the `Respawning Too Rapidly` error is displayed on the console and the port is disabled by the system.

The tty stays disabled for about 19 minutes or until the system administrator enables the port again. At the end of the 19 minutes, the system automatically enables the port, resulting in the creation of a new **getty** process.

Possible causes include the following:

- Incorrect modem configuration
- A port is defined and enabled but no cable or device is attached to it
- Bad cabling or loose connection
- Noise on communication line
- Corruption of, or tampering with, `/etc/environment` or `/etc/inittab` files
- tty configuration is corrupted
- Hardware is defective

Of the following procedures for recovery, use the one that applies to your situation.

- Incorrect modem configuration:

  Ensure that the modem carrier detect is *not* forced high.

  Note: The following applies to Hayes-compatible modems

1. Connect to the modem and examine the active profile.
2. Set the modem carrier detect to **&C1** rather than **&C0** (forced high). Use the following AT modem commands to set and change the carrier attribute:

   ```
   AT&C1
   AT&W
   ```

   Note:

   a. See "Sending AT commands with the cu command" on page 562
   b. See your modem documentation for further information.

- Disable the tty, remove the tty definition, or attach a device to the port:
  - To disable the tty definition use the **chdev** command as follows:

    ```
    chdev -l ttyName -a Login=disable
    ```

    After running this command, the tty does *not* become enabled after a system restart.
  - To remove the tty definition:
    1. Disable the tty port, use the **pdisable** command, enter:

       ```
       pdisable ttyName
       ```

    2. Remove the tty definition from the system. See "TTY device management" on page 548 for further information.
- Check for bad cables or loose connections:
  1. Check cabling. Tighten loose connections and replace damaged or inappropriate connectors.
  2. Verify that the suspected cabling is IBM serial cable P/N 6323741 or that the cable meets the same standard. Replace damaged or inappropriate cables.
- Eliminate noise on communication line:
  1. Verify that cabling is correct length and impedance.
  2. Ensure that toroid rings are in place where needed on longer cables.
  3. Check routing of cables; they should not be close to fluorescent lights or motors.
- Check for corruption of, or tampering with, the `/etc/environment` or the `/etc/inittab` files:
  1. If possible, compare these files against known good copies.
  2. Copy the files as a backup and make changes as needed.
  3. In the `/etc/environment` file, remove any lines that are *not*:
     - blank lines
     - comment lines
     - *variable=value*
  4. In the `/etc/inittab` file, examine the tty devices lines. If the tty is set to off, it is likely that the tty port is not being used. If it is not being used, remove the tty definition or attach a device to the port.
- Remove corrupted tty configuration:
  1. Remove the tty definition. See "TTY device management" on page 548 for further information.
  2. If you want a hard copy record of the tty definition before removing it, press the Image key (F8 or Esc+8). This will capture the current screen image and copy it to the `smit.log` file in your `$HOME` directory.
  3. Read the tty definition. See the instructions for Adding a TTY under "TTY device management" on page 548.
- Locate defective hardware:
  1. Run diagnostics using the **diag** command.
  2. If any hardware problems are detected, follow local problem solving procedures.

Error log information and TTY log identifiers:

The following commands and logging files relate to TTYs.

Command: **errclear**

This command deletes entries from the error log. The entire log can be erased with `errclear 0` or entries with specified error ID numbers, classes, or types can be removed.

Command: **errpt**

This command generates an error report from entries in the system error log. The most used format for this command is `errpt -a | pg`, which generates a detailed report starting with the most current errors.

File: `/var/adm/ras/errlog`

This file stores instances of errors and failures encountered by system. The `errlog` file tends to become quite lengthy. If not cleared on a regular basis, it can occupy quite a bit of space on your hard disk. Use the **errclear** command mentioned previously to clean out this file.

File: `/usr/include/sys/errids.h`

The `errids.h` header file correlates error IDs with error labels.

The following common error report messages relate to TTY:

Table 100. TTY error messages

Message	Description	Comments
Core Dump	Software program abnormally terminated	This error is logged when a software program abnormally ends and causes a core dump. Users might not be exiting applications correctly, the system might have been shut down while users were working in application, or the user's terminal might have locked up and the application stopped.
Errlog On	Errdaemon turned on	This error is logged by the **error** daemon when the error logging is started. The system automatically turns off error logging during shutdown.
Lion Box Died	Lost communication with 64-port concentrator	This error is logged by the 64-port concentrator driver if communications with the concentrator are lost. If you receive this error, check the date and time stamp to see if user might have caused this message to occur. A series of these errors can indicate a problem with the 64-port adapter or its associated hardware.
Lion Buffero	Buffer overrun: 64-port concentrator	This error occurs when the hardware buffer in a 64-port concentrator is overrun. If device and cabling allow, try adding request to send (RTS) handshaking to the port and device. Also try lowering the baud rate.

Table 100. TTY error messages (continued)

Message	Description	Comments
Lion Chunknumc	Bad chunk count: 64-port controller	This error occurs when the value for the number of characters in a chunk does not match the actual values in the buffer. This error may indicate a problem with the hardware; try running diagnostics on devices.
Lion Hrdwre	Cannot access memory on 64-port controller	This error is logged by the 64-port concentrator driver if it is unable to access memory on the 64-port controller.
Lion Mem ADAP	Cannot allocate memory: ADAP structure	This error is logged by the 64-port concentrator driver if the **malloc** routine for the adap structure fails.
Lion Mem List	Cannot allocate memory: TTYP_T List	This error is logged by the 64-port concentrator driver if the **malloc** routine for the *ttyp_t* list structure fails
Lion Pin ADAP	Cannot pin memory: ADAP structure	This error is logged by the 64-port concentrator driver if the **pin** routine for the adap structure fails.
SRC	Software program error	This error is logged by the System Resource Controller (SRC) daemon in the event of some abnormal condition. Abnormal conditions are divided in three areas: failing subsystems, communication failures, and other failures.
Lion Unkchunk	Unknown error code from the 64-port concentrator	Error Code: Number of characters in the chunk received.
TTY Badinput	Bad cable or connection	The port is generating input faster than the system can consume it, and some of that input is being discarded. Usually, the bad input is caused by one or more RS-232 signals changing their status rapidly and repeatedly in a short period of time, causing your system to spend a lot of time in the interrupt handler. The signal errors are usually caused by a loose or broken connector; a bad, ungrounded, or unshielded cable; or by a "noisy" communications link.

Table 100. TTY error messages (continued)

Message	Description	Comments
TTY Overrun	Receiver overrun on input	Most TTY ports have a 16-character input FIFO, and the default setting specifies that an interrupt is posted after 14 characters have been received. This error is reported when the driver interrupt handler cleared the input FIFO and data has been lost. Potential solutions depend on the hardware you are using: • 8-port and 128-port adapters Verify that flow control is configured correctly. If it is, run diagnostics, and replace the hardware as appropriate. • Native ports If the problem happens on an idle system, move the workload to a different port. If that corrects the problem, upgrade the system firmware. • General solutions – Reduce the "RECEIVE trigger level" parameter for this port from 3 to either 2 or 1. – Reduce the line speed on this port. – Examine other devices and processes to try to reduce the time that the system is spending with interrupts disabled.
TTY TTYHOG	TTYHOG overrun	This error is usually caused by a mismatch in the flow control method being used between the transmitter and receiver. The TTY driver has made several attempts to ask the transmitter to pause, but the input has not stopped, causing the data to be discarded. Check the flow control methods configured on each end to make sure that the same method is being used on each.
TTY Parerr	Parity/Framing error on input	This error indicates parity errors on incoming data to asynchronous ports on a character-by-character basis. This is usually caused by a mismatch in line control parameters (parity, line speed, character size, or number of stop bits) between the transmitter and receiver. Line control parameters have to be set the same on both sides in order to communicate.
TTY Prog PTR	Driver internal error	This error is logged by the tty driver if *t_hptr* pointer is null.

Clearing a hung TTY port:

In this example of clearing a hung port, assume that the hung tty port is `tty0`.

You must have root authority to be able to complete this procedure.

1. Determine whether the tty is currently handling any processes by typing the following:
   ```
   ps -lt tty0
   ```
   This should return results similar to the following:

```
     F S UID   PID  PPID   C PRI NI ADDR    SZ    WCHAN    TTY  TIME CMD
240001 S 202 22566  3608   0  60 20 781a   444 70201e44   tty0  0:00 ksh
```

The Process ID (PID) here is 22566. To kill this process, type the following:

```
kill 22566
```

Ensure that the process was successfully cleared by typing the command `ps -lt tty0`. If the process still exists, add the `-9` flag to the kill command as shown in the example below.

Note: Do not use the -9 option to kill an slattach process. Killing an slattach process with the `-9` flag might cause a slip lock to remain in the `/etc/locks` file. Delete this lock file to clean up after slattach.

```
kill -9 22566
```

2. Determine if any process is attempting to use the tty by typing the following:

   ```
   ps -ef | grep tty0
   ```

   Note: If the **ps -ef | grep tty** command returns something similar to the following:

   ```
   root 19050      1    0    Mar 06       -  0:00 /usr/sbin/getty /dev/tty
   ```

   where the "-" is displayed between the date (`Mar 06`) and the time (`0:00`), this tty does not have the correct cable. This status indicates that the system login process (getty) is attempting to open this tty, and the open process is hanging because the RS-232 signal Data Carrier Detect (DCD) is not asserted. You can fix this by using the correct null modem adapter in the cabling. When getty can open the tty port, the "-" is replaced by the tty number. For more information on cables, see “Attaching the modem with appropriate cables” on page 561.

   Note: The following command can be used to disable the login process on `tty0`.

   ```
   pdisable tty0
   ```

   If the process has been successfully cleared but the tty is still unresponsive, continue to the next step.

3. Type the following command:

   ```
   fuser -k /dev/tty0
   ```

   This will clear any process that can be found running on the port and display the PID. If the tty is still unusable, continue to the next step.

4. Use the **strreset** command to flush outgoing data from the port that is hung due to data that cannot be delivered because the connection to the remote end has been lost.

   Note: If the **strreset** command fixes the hung port, the port has a cable or configuration problem because the loss of the connection to the remote end should have caused buffered data to be flushed automatically.

   You need to first determine the major and minor device numbers for the tty by typing the following:

   ```
   ls -al /dev/tty0
   ```

   Your results should look similar to the following:

   ```
   crw-rw-rw-    1 root     system    18,   0 Nov  7  06:19 /dev/tty0
   ```

   This indicates that `tty0` has a major device number of 18 and a minor device number of 0. Specify these numbers when using the **strreset** command as follows:

   ```
   /usr/sbin/strreset -M 18 -m 0
   ```

   If the tty is still unusable, continue to the next step.

5. Detach and reattach the cable from the hung tty port. AIX uses the Data Carrier Detect (DCD) signal to determine the presence of a device attached to the port. By dropping DCD, detaching and reattaching the cable will in many cases clear hung processes.

   To determine the location of the port on which the tty is configured, type the following command:

```
lsdev -Cl tty0
```

The results should look similar to the following:

```
tty0    Available   00-00-S1-00  Asynchronous Terminal
```

The third column in the above output indicates the location code of the tty. In this example, `S1` indicates the serial port is configured for native serial port 1. For more information on interpreting location codes, see Device location Codes in *Operating system and device management*.

If the tty is still unusable, continue to the next step.

6. Flush the port using **stty-cxma**. Type the following:

   ```
   /usr/lbin/tty/stty-cxma flush tty0
   ```

   This command is intended for the ttys configured on ports of the 8-port and 128-adapters. In some cases, however, it can be used successfully to flush other tty ports.

   If the tty is still unusable, continue to the next step.

7. On the keyboard of the hung terminal, hold down the `Ctrl` key and press `Q`. This will resume any suspended output by sending an **Xon** character.

   If the tty is still unusable, continue to the next step.

8. A program will sometimes open a tty port, modify some attributes, and close the port without resetting the attributes to their original states. To correct this, bring the tty down to a DEFINED state and then make it available by typing the following:

   ```
   rmdev -l tty0
   ```

   This command leaves the information concerning the tty in the database but makes the tty unavailable on the system.

   The following command reactivates the tty:

   ```
   mkdev -l tty0
   ```

   If the tty is still unusable, consider moving the device to another port and configuring a tty at that location until the system can be rebooted. If rebooting does not clear the port, you most likely have a hardware problem. Check the error report for port hardware problems by entering the following:

   ```
   errpt -a | pg
   ```

Some of the preceding commands will not work, and they will give a method error indicating that the device is busy. This is because of the process running on the tty. If none of the steps detailed above free the hung tty, as a last resort, reboot the AIX system and flush the kernel so that the process will go away.

Modems

Modems provide serial communications across ordinary telephone lines. Modem concepts include standards, general modem setup, and specific configuration tips for popular modems.

A *modem* is a device that allows you to connect one computer to another across ordinary telephone lines. The current telephone system is incapable of carrying the voltage changes required for a direct digital connection. A modem overcomes this limitation by modulating digital information into audio tones for transmission across the phone line, and by demodulating those tones back into digital information on reception. Modems are commonly used with Basic Network Utilities (BNU) or other implementations of the UNIX-to-UNIX Copy Program (UUCP). A high-speed (14,400 bps or greater) modem can be used with Serial Line Interface Protocol (SLIP) to provide Transmission Control Protocol/Internet Protocol (TCP/IP) connectivity as well.

Often, the term *baud* is used to refer to modem speed instead of bps. Baud is actually a measurement of the modulation rate. In older modems, only 1 bit was encoded in each signal change, so modem baud rate was equal to modem speed. Modems that operate at higher speeds, however, still generally operate at 2,400 (or even 1,200) baud, and encode two or more bits per signal change. A modem's bps rate is

calculated by multiplying the number of data bits per signal with the baud (for example, 2,400 baud x 6 bits per signal change = 14,400 bits per second). Most modern modems can communicate at a variety of speeds (for example, 28,800, 14,400, 9,600, 7,800, 4,800, and 2,400 bps).

Telecommunications standards

The older speeds of 300, 1,200, and 2,400 bps were well defined. However, as modem manufacturers began to devise methods for gaining higher speeds, each modem manufacturer started to use a proprietary method incompatible with modems from other manufacturers. Today, the ITU-TSS (formerly the United Nations Consultative Committee for International Telephony and Telegraphy, abbreviated CCITT) defines standards for most high-speed communications.

Even high-speed modems are much slower than other methods of computer communication. A high-speed modem can operate at 28,800 bps, but an Ethernet connection operates at 10,000,000 bps. To boost data throughput, high-speed modems typically offer one or more data compression algorithms. These algorithms can boost the throughput of a high-speed modem to speeds of 57,600 bps (if the data rate is 14,400 bps) or 115,200 bps (if the data rate is 28,800 bps). Note that these compression algorithms are sensitive to the data being transmitted. If the data has already been compressed (for example, with the **compress** command), the data compression methods of high-speed modems offer little or no benefit, and might even reduce data throughput. When using a modem with data compression technology, the speed of the data terminal equipment/data circuit-terminating equipment (DTE/DCE) connection between the computer and the modem is equal or greater than the nominal data rate of the connection between modems. For example, with a V.32*bis* modem with V.42*bis* data compression, the data rate of the modem (the speed at which the modem communicates across telephone lines) is 14,400 bps. When the V.42*bis* compression is active, actual data throughput can reach 57,600 bps. To accommodate the greater throughput offered by data compression, the speed of the link between the computer and the modem should be set to 57,600 bps.

The ITU-TSS defines standards for high-speed communications, including data compression algorithms. ITU-TSS standards are usually named V.*nn*, where *nn* is a number. Another, slightly less common standard is the Microcom Networking Protocol (MNP). Available in versions (called classes) 1-9, MNP is a high-performance, high-speed protocol that was available relatively early, and became something of a de facto standard before the advent of the ITU-TSS standards.

Full and half duplex transmissions:

When studying telecommunications standards, it is important to understand the differences between half duplex and full duplex transmissions.

In a *half duplex* (HDX) transmission, a data packet is sent by one system and received by the other. Another data packet cannot be sent until the receiving system sends an acknowledgment back to the sender.

In a *full duplex* (FDX) transmission, both the sending and receiving systems communicate with each other simultaneously; in other words, both modems can send and receive data at the same time. This means a modem can be receiving a data packet while acknowledging the receipt of another.

ITU-TSS communications standards:

Some common communications standards defined by the ITU-TSS are described here.

Note that this only a partial list. For a complete list, refer to the Internet website for the International Telecommunication Union.

Item	Description
V.29	ITU-TSS standard for half-duplex 9600 bps communications.
V.32	ITU-TSS standard for full-duplex 9600 bps communications.
V.32*bis*	ITU-TSS standard for 14,400 communications. V.32*bis* is a revision to the V.32 standard.
V.34	ITU-TSS standard for 33,600 bps communications. Note that this standard achieves 33,600 bps data rates using multiple bit encoding, instead of the data compression scheme used by MNP Class 9. This standard was previously referred to as V.*fast*.
V.42	ITU-TSS error correcting procedures for DCEs using asynchronous to synchronous conversion.
V.42*bis*	Revised ITU-TSS data compression standard.

Microcom Networking Protocol:

Another de facto standard is the **Microcom Networking Protocol (MNP)** which was originally developed by Microcom, Inc.

Available in versions (called classes) 1-9, **MNP** is a high-performance, high-speed protocol that was available before the advent of the ITU-TSS standards. With **MNP**, errors in the transmitted data packets are detected by the remote modem causing it to request a retransmission of the data packet in error. The capability to recognize and quickly correct data errors makes **MNP** one of today's most common protocols.

The following table specifies **MNP** communication standards.

Item	Description
MNP Class 1	An asynchronous, half-duplex, byte-oriented method of transferring data realizing about 70% efficiency. This standard is uncommon in modern modems.
MNP Class 2	A full-duplex counterpart to **MNP Class 1** which is also uncommon in modern modems.
MNP Class 3	A synchronous, bit-oriented full-duplex method of transferring data realizing about 108% efficiency. Efficiency greater than 100% is realized because the start/stop bits required for an asynchronous connection are eliminated. The DTE/DCE between the modem and the system are still asynchronous.
MNP Class 4	An enhancement to **MNP Class 3** including a mechanism for varying the packet size (adaptive packet assembly) and a means of eliminating redundant administrative overhead (data phase optimization). An **MNP Class 4** modem offers approximately 120% efficiency.
MNP Class 5	Includes data compression along with Class 4 features. An **MNP Class 5** modem offers 200% efficiency.
MNP Class 6	Allows incorporation of multiple, incompatible modulation techniques into one modem (universal link negotiation). This allows **MNP Class 6** modems to begin communication at a slower speed and negotiate a transition to a higher speed. **Class 6** also includes a statistical duplexing scheme which dynamically allocates utilization of half-duplex modulation to simulate full-duplex service. All features of **MNP Class 5** are supported.
MNP Class 7	Incorporates enhanced data compression. Combined with Class 4, efficiencies of 300% can be realized.
MNP Class 8	Not Applicable.
MNP Class 9	Combine enhanced data compression with V.32 technology to allow data rates up to 28,800 bps.

Modem considerations

Modem interface requirements for the general user can vary.

The configuration of a modem attached to this operating system is different than that of a personal computer (PC) or workstation.

Supported modems:

Any modem that is EIA 232 compliant and capable of returning results in response to a command can be attached to this operating system.

Data Carrier Detect handling:

The server uses the Data Carrier Detect (DCD) signal to monitor the true state of a modem.

If the DCD signal on the modem's port is "high," the server believes the modem to be in use. It is therefore important to know which circumstances cause this signal to be forced into a "high" state. The DCD signal can be raised high for the following reasons:

- The use of **clocal** in the stty attributes for runtime field on the SMIT **TTY Configuration** panel.
- Having the Ignore Carrier Detect field set to **enable** on the SMIT **TTY Configuration** panel for ttys connected to a 128-port adapter.
- The modem forces DCD high with either AT commands or switches.
- The tty port is already in use by an application.

Note: When modems make a connection with another modem, the modem raises the CD. Most modem defaults settings set this signal "high" at all times even when the modem is idle. CD should not be forced "high."

Data Terminating Equipment or Data Circuit-Terminating Equipment speeds:

Data Terminating Equipment (DTE) and Data Communication Equipment (DCE) are used to describe two different hardware groups.

The term DTE is used primarily for those devices that display user information. It also includes any devices that store or generate data for the user. The system units, terminals, and printers all fall into the DTE category.

DCE includes any device which can be used to gain access to a system over telecommunication lines. The most common forms of DCEs are modems and multiplexers.

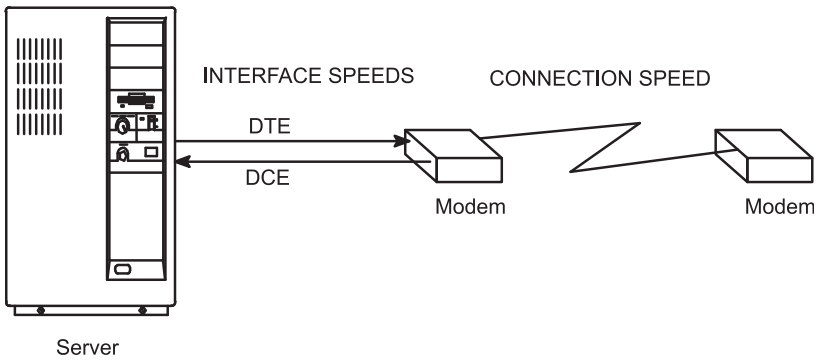


Figure 37. Modem speed considerations

With serial communication on this operating system involving modems, as pictured in the above illustration, there are three major considerations:

- DTE interface speed (server to modem). This is the speed the server communicates to the modem.
- DCE interface speed (modem to server) sometimes called the "serial port interface speed." This is the speed at which the modem communicates to the server.
- Connection speed (modem to modem). This is the speed at which a modem communicates (or talks) to another modem.

Most modern, high-speed modems allow the DCE interface speed to be different than the connection speed. This allows the DTE speed to be locked at a single baud rate while allowing the connection speed to fluctuate, up or down as needed, for proper communication between modems.

Modern high-speed modems hold the data to be transmitted to the server in a buffer and send it when the system can accept it. They can also hold data to be transmitted to the other modem in a buffer and send it as the remote is able to accept it. This kind of data transmission requires the modem and the server to engage in *flow control*.

Modem control signals:

Modems are often used to initiate and receive calls. It is therefore important to program the modem to negotiate a connection at the highest possible speed and to reset itself to a known state after a connection is stopped.

The server will toggle the Data Terminal Ready (DTR) signal from on to off to instruct the modem to terminate the connection. Most modems can be configured to reset themselves when this on-to-off DTR transition occurs.

Note: The tty can be configured to not drop DTR by disabling the **hupcl** flag in the stty run-time attributes.

For the connection between the server and the modem to be fully functional, the cabling must have the following qualifications:

- It must meet specifications.
- It should be properly shielded.
- The following signals should be provided: RxD, TxD, RTS, CTS, SG, DCD, and DTR.

  Note: The 16-port asynchronous adapter does not provide support for the RTS and CTS signals. It is therefore impossible to use RTS/CTS hardware flow control with this adapter.

If binary data is to be transferred using a modem on this adapter, a file transfer protocol that detects incorrect data and resends the missing data (for example, Xmodem, zmodem, Kermit, and UUCP) should be used.

The following describes the signals used by the server:

Signal	Description
FG	Frame Ground. Pin 1 of the EIA 232D specification that provides for a cable shield. Properly used, the signal is attached at pin 1 on one side of the cable only and is connected to a metal sheath around the cable.
TxD	Transmit Data. Pin 2 of the EIA 232D specification. Data is transmitted on this signal. Controlled by the server.
RxD	Receive Data. Pin 3 of the EIA 232D specification. Data is received on this signal, controlled by the modem, which is sent by the modem.
RTS	Request To Send. Pin 4 of the EIA 232D specification. Used when RTS/CTS flow control is enabled. This signal is brought high when the system is ready to send data and dropped when the system wants the modem to stop sending data.
CTS	Clear To Send. Pin 5 of the EIA 232D specification. Used when RTS/CTS flow control is enabled. This signal will be brought high when the modem is ready to send or receive data. It will be dropped when the modem wishes the server to stop sending data. Controlled by the modem.
DSR	Data Set Ready. Pin 6 of the EIA 232D specification. Signals the server that the modem is in a state where it is ready for use. Controlled by the modem.
SG	Signal Ground. Pin 7 of the EIA 232D specification. This signal provides a reference voltage for the other signals.
DCD	Data Carrier Detect. Pin 8 of the EIA 232D specification. This provides a signal to the server that the modem is connected with another modem. When this signal is brought high, programs running on the server will be able to open the port. Controlled by the modem.
DTR	Data Terminal Ready. Pin 20 of the EIA 232D specification. This provides a signal to the modem that the server is on and ready to accept a connection. This signal is dropped when the server wishes the modem to drop connection to another modem. It is brought high when the port is being opened. Controlled by the server.
RI	Ring Indicate. Pin 22 of the EIA 232D specification. This provides a signal to the server that the modem is receiving a call. It is seldom used and is not needed for common operations. Controlled by the modem.

Modem cabling

These tables display a summary of the cable information needed to properly attach a modem to any of the serial controllers.

Adapter/Controller	IBM Part Number(s)
Native Serial (S1 or S2)	00G0943*, 6326741
2-port Controller	00G0943*, 6326741
8-port Controller	6323741
128-port Controller	43G0935, 6323741

IBM Part Number	Description	Length in Feet
00G0943*	Serial Port Jumper (pigtail)	.33
6323741	Asynchronous	10
43G0935	RJ-45 to DB25 Converter Cable	2

*This part number is not required for some machine types.

TTY device setup on the operating system

Use the System Management Interface Tool (SMIT) to define a tty port for the device attachment.

Most fields are for the general device type. The only field that can affect the modem is the Enable LOGIN field with the following values:

Item	Description
DISABLE	No getty process is run on the port. Use this setting for dial-out only modem ports.
ENABLE	A getty process is run on the port. Use this setting for dial-in modems only.
SHARE	A getty process is run on the port, but the getty process allows programs to dial in and out of this port without manually changing to disable or enable. Use this setting for bidirectional port usage.
DELAY	A getty is run on the port in bidirectional mode, but no herald is sent until the getty process receives a keystroke from the user.

Fields specific to the 128-port asynchronous adapter:

Item	Description
Force Carrier or Ignore Carrier Detect	disable*
Perform Cooked Processing in Adapter	disable

Note: This setting indicated by an asterisk (*) is set to disabled if the 10 pin RJ-45 connector is used. This setting should be enabled if the 8 pin RJ-45 connector is used.

Attaching the modem with appropriate cables

The first step in setting up a modem is to attach the modem with the appropriate cables.

Part numbers and their descriptions are listed below.

6323741

Async Cable, EIA-232; used to attach all asynchronous devices; sometimes used with other cable assemblies.

59F3740

10 to 25-pin D-shell connector used to attach asynchronous cable 6323741 to native serial ports S1 and S2 as shown in the following figure.

Figure 38. 10 to 25-pin connector

This illustration shows a 10 to 25-pin connector.

Following are some examples of cable connections:

1. To attach a modem to native serial port S1, use the following cables:

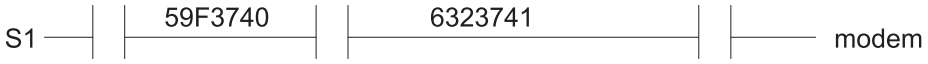


Figure 39. Modem to Native Serial Port Cable Assembly

This illustration shows a 59F3740 cable on the serial port end and a 6323741 on the modem end.

2. To attach a modem to an 8-port async adapter (EIA-232) interface cable assembly, use the following cables:

Figure 40. 8-port interface to modem cable assembly

This illustration shows an 8-port interface connected to a modem with a 6323741 cable.

Adding a TTY for the modem

Use this information when adding a tty for a modem.

First, ensure that the system is turned on and that the modem is turned off. Use the SMIT fast path `smit mktty`.

Modem configuration

Use only one of the two methods presented here for configuring the modem.

If you have Basic Networking Utilities (BNU) installed, see "Sending AT commands with the cu command." If you do not have BNU installed, see "Sending AT commands using a C program" on page 563. For information on installing BNU, see "Basic Networking Utilities" on page 406.

Sending AT commands with the cu command:

If you have the Basic Network Utilities (BNU) installed, use the **cu** command to configure a modem as follows.

The commands and settings discussed in this section configure a Hayes-compatible modem with the basic parameters needed for operation on the server's serial ports.

1. Add the following line to your **/usr/lib/uucp/Devices** file. Do not add the line if it is already in the file. (Replace # with the number for your port.)

   ```
   Direct tty# - Any direct
   ```

2. Verify that the tty is disabled by typing the following:

   ```
   pdisable tty#
   ```

3. Type the following command:

   ```
   cu -ml tty#
   ```

You should see a message that says `Connected`.

4. Verify that you have the attention of the modem by typing the following:
   ```
   AT
   ```
   The modem should respond with `OK`. If it didn't, refer to “Modem troubleshooting” on page 565.
   For additional **AT** commands and their descriptions, see “AT commands” on page 567.
5. Depending on which getty option you selected, enter one of the following commands. Substitute the tty device for *n*.
   - `penable tty`*n*
   - `pshare tty`*n*
   - `pdelay tty`*n*
   - `pdisplay tty`*n*

The modem is now configured with the basic commands needed to perform most of the operating system's serial communications needs. If you have problems, invoke the **cu -dl** command to start a diagnostic trace on the connection.

Sending AT commands using a C program:

If you could not configure the modem using the **cu** command, or if you do not have BNU installed, try running the following C program.

Create a file called `motalk.c` containing the following code. Save the file. Compile and run it according to the instructions in the program comments.

```
/*************************************************************/
/*  MoTalk - A "C" program for modem setup.                  */
/*           This program is meant as an aid only and is     */
/*           not supported by IBM.                          */
/*                  compile:  cc -o motalk motalk.c          */
/*                  Usage:  motalk /dev/tty? [speed]         */
/*************************************************************/
#include <errno.h>
#include <stdio.h>
#include <signal.h>
#include <fcntl.h>
#include <termio.h>
FILE *fdr, *fdw;
int fd;
struct termio term_save, stdin_save;
void Exit(int sig)
{
   if (fdr) fclose(fdr);
   if (fdw) fclose(fdw);
   ioctl(fd, TCSETA, &term_save);
   close(fd);
   ioctl(fileno(stdin), TCSETA, &stdin_save);
   exit(sig);
}
main(int argc, char *argv[])
{
   char *b, buffer[80];
   int baud=0, num;
   struct termio term, tstdin;
   if (argc < 2 || !strcmp(argv[1], "-?"))
   {
      fprintf(stderr, "Usage: motalk /dev/tty? [speed]\n");
      exit(1);
   }
   if ((fd = open(argv[1], O_RDWR | O_NDELAY)) < 0)
   {
```

```
   perror(argv[1]);
   exit(errno);
}
if (argc > 2)
{
   switch(atoi(argv[2]))
   {
      case   300: baud = B300;
                  break;
      case  1200: baud = B1200;
                  break;
      case  2400: baud = B2400;
                  break;
      case  4800: baud = B4800;
                  break;
      case  9600: baud = B9600;
                  break;
      case 19200: baud = B19200;
                  break;
      case 38400: baud = B38400;
                  break;
      default:    baud = 0;
                  fprintf(stderr, "%s: %s is an unsupported baud\n", argv[0],argv[2]);
                  exit(1);
      }
   }
/* Save stdin and tty state and trap some signals */
ioctl(fd, TCGETA, &term_save);
ioctl(fileno(stdin), TCGETA, &stdin_save);
signal(SIGHUP, Exit);
signal(SIGINT, Exit);
signal(SIGQUIT, Exit);
signal(SIGTERM, Exit);
/*  Set stdin to raw mode, no echo */
ioctl(fileno(stdin), TCGETA, &tstdin);
tstdin.c_iflag = 0;
tstdin.c_lflag &= ~(ICANON | ECHO);
tstdin.c_cc[VMIN] = 0;
tstdin.c_cc[VTIME] = 0;
ioctl(fileno(stdin), TCSETA, &tstdin);
/*  Set tty state */
ioctl(fd, TCGETA, &term);
term.c_cflag |= CLOCAL|HUPCL;
if (baud > 0)
{
   term.c_cflag &= ~CBAUD;
   term.c_cflag |= baud;
}
term.c_lflag &= ~(ICANON | ECHO); /* to force raw mode */
term.c_iflag &= ~ICRNL; /* to avoid non-needed blank lines */
term.c_cc[VMIN] = 0;
term.c_cc[VTIME] = 10;
ioctl(fd, TCSETA, &term);
fcntl(fd, F_SETFL, fcntl(fd, F_GETFL, 0) & ~O_NDELAY);
/*  Open tty for read and write */
if ((fdr = fopen(argv[1], "r")) == NULL )
{
   perror(argv[1]);
   exit(errno);
}
if ((fdw = fopen(argv[1], "w")) == NULL )
{
   perror(argv[1]);
   exit(errno);
}
/*  Talk to the modem */
puts("Ready... ^C to exit");
```

```
   while (1)
   {
      if ((num = read(fileno(stdin), buffer, 80)) > 0)
         write(fileno(fdw), buffer, num);
      if ((num = read(fileno(fdr), buffer, 80)) > 0)
         write(fileno(stdout), buffer, num);
      Exit (0);
   }
}
```

Using Hayes and Hayes-compatible modems

Use this procedure for Hayes and Hayes-compatible modems.

1. Change the tty settings, if necessary, using the SMIT fast path, `smit chtty`. For example, you might want to change the Enable LOGIN field to **Share** or **Enable**.
2. Add the following line to `/usr/lib/uucp/Systems` file:

   ```
   hayes Nvr HAYESPROG 2400
   ```

3. Add this to `/usr/lib/uucp/Devices` file:

   ```
   # For programming the  hayes modem only:
   HAYESPROG tty0 - 2400 HayesProgrm2400
   #regular ACU entry:
   ACU tty0 - Any hayes
   ```

4. Add this to `/usr/lib/uucp/Dialers` file:

   ```
   # This Entry is used to PROGRAM the modem ONLY:
   # the next 3 lines should be made into one:
   HayesProgrm2400     =,-,     "" \d\dAT\r\c OK AT&F\r\c OK ATM1\r\c OK
   AT&D3\r\c OK AT&K3&C1\r\c OK ATL0E0Q2\r\c OK ATS0=1\r\c OK AT&W\r\c
   OK
   hayes     =,-,     "" \dAT\r\c OK ATDT\T\d\r\c CONNECT
   ```

5. To program the modem, enter the command `cu -d hayes`. This command uses the **cu** command to program the modem. Because no connection is made to another system, the command will fail. The modem is programmed if `sendthem AT&W` and then `OK got it` are displayed in the output.

   If you are not doing binary file transfers or using BNU, leave out the **&K3** command, and set XON as the flow control to be used. However, it is more efficient to use hardware flow control (as opposed to XON-XOFF handshaking). To do that, use the settings and the `Dialers` entries from the next step.
6. After the modem is programmed, you can configure the system device driver to use hardware flow control. Using SMIT (`smit chtty` fast path), change the flow control to RTS. Check your modem manuals to find out whether your modem supports hardware flow control.

Modem troubleshooting

When encountering problems when you use a modem with your computer, keep in mind the following points.

- Some modems are case-sensitive. Use uppercase letters for the **AT** commands.
- In normal operation, it is preferable for the modem to reset when the DTR is dropped (&D3 setting). When the modem is being set up for the first time, however, it is advisable not to have the modem reset if the DTR is dropped (&D2 setting). If the modem resets itself, all programmed settings that are not saved in the modem's memory will be lost.

  Not having the modem reset also protects changes when &C1 is set. Changing the Carrier Detect status might cause the carrier detect line to toggle some modems, which causes the **cu** command to drop the line. You might want to set up the modem to &D3 after the final setup is made.
- Although the commands given in this topic collection are standard for most Hayes-compatible modems, there is no guarantee that they are standard for your modem. Compare the commands with your modem documentation before proceeding.

A convenient way to debug any modem problem is to remove the modem and attach an ASCII terminal (with an interposer or null modem) to the same port and cabling as the modem. Set up the terminal with the same line speed, bits per character, and parity as the modem. If the port is enabled for login, a login

herald must be displayed on the screen. If the herald is displayed on the terminal screen, then the problem is quickly isolated to the modem configuration.

The following tips help you in isolating problems that are associated with modem connections:

Table 101. Modem problems and solutions

Problem	Resolution
Respawning too rapidly	The **getty** program is respawned by **init**.
Messages on console or errpt	If **init** sees that it has to respawn any program more that five times in 225 seconds, it will display the message on the console and not respawn it for a period of time. The solution is to find out why **getty** is dying. There may be several causes: • Incorrect modem settings, usually as a result of having CD strapped high on the modem or cabling and also having either "echo" or "command response" turned on. (CD can also be assumed high by adding clocal in the runmodes and/or logmodes in the port configuration or also forced on the 128 port.) • Toggling of the CD signal. The **getty** process will die every time CD is toggled from an on to off state. (This action could be caused by a number of reasons. Be sure to verify that the cable is properly shielded. Logging in and out several times in rapid succession can cause this.)
No login prompt displayed after connection to modem	Make sure **getty** is running on the port. If it is, verify that the carrier detect connection to modem signal is being raised after the remote side has connected to the modem. If CD is being properly asserted, then verify the modem is connected to the right port. If you still do not see login, then attach a terminal with interposer to the cable in place of the modem and verify that a login prompt does appear. If you still don't see a prompt, try to echo characters to the terminal screen to verify the cable and hardware are functioning properly.
When a remote modem connects, it immediately disconnects	Verify that the modem is talking to the server at the same speed at which the server is listening to the modem. Try different baud rates for the tty, or program the modem to lock DTE speed to match the speed of the tty port. Verify that the modem or port is not keeping the carrier detect signal high or that the port is already being used by another process.
Getting garbage characters instead of a login prompt	This is due to a difference in protocols. Be sure to verify that the modem and the tty port agree on the same parity, baud rate, flow control, and character size.
Sometimes, after a successful session, no one is able to login	It might be that the modem does not reset after disconnect. See the modem manual to see how the modem can be set to reset after an on to off DTR transition.
Receiver buffer overruns in errpt	The UART chip buffer is being overrun. Lower the value of the receive trigger in SMIT for the tty. This solution is only valid for the native, 8- or 16-port asynchronous adapters. Verify that the modem and tty port are using the same flow control.
ttyhog errors in errpt	Modem and tty either do not agree on flow control, or no flow control is taking place.

Software Services Modem Questionnaire:

Before calling for assistance with modem problems, collect some basic information to expedite your service.

The information you have available includes the following:

- Level of the operating system. How long have you been at this level of the operating system?
- Has the modem ever worked before?
- What type of modem are you using? What type of modem is on the other end of the telephone connection?
- To what adapter type is the modem attached?
- To which port number is the modem attached?
- To which tty number is the modem attached?
- What type of cabling are you using?
- What is the login setting (share, delay, enable)?
- Can the modem connect to other modems?
- Can other modems connect to your modem?
- What are the following values in SMIT, modem, or port?

  - XON/XOFF?
  - RTS/CTS?
  - BPS rate?
- Include the following in your problem description:
  - Does the port lock intermittently?
  - Can you dial out? Can others dial in?
  - Any other specific and descriptive error conditions.
- Are there errors on the console? What are they?
- Are there errors in the error report? (**errpt** or **errpt -a**)
- What command are you using to dial out?
- What software is involved on the system?

AT commands:

The Hayes Smartmodem command set include the AT command set used by many popular modems.

This information comes from the Hayes Smartmodem 2400 *Quick Reference Card*, published by Hayes Microcomputer Products, Inc. Consult the modem documentation for a list of relevant AT commands.

Item	Description
AT	Command prefix - precedes command line.
<CR>	Carriage return (newline) character - terminated the command line.
A	Go off-hook, remain in command mode.
A/	Repeat previous command line. This command is not preceded with **AT** or followed by **<CR>**/.
B0	Select CCITT V.22 standard for 1200 bps communications.
B1	Select Bell 212A standard for 1200 bps communications.
D	Enter originate mode, dial the number that follows, and attempt to go online. D is usually followed by T for tone, P for pulse may also be used.
DS=*n*	Dial the number stored in location *n*
E0	Disable character echo in the command state.
E1	Enable character echo in the command state.
H0	Go on-hook (hang up the phone).
H1	Operate switch-hook and auxiliary relay.
I0	Return product identification code.
I1	Perform checksum on firmware ROM; return checksum.
I2	Perform checksum on firmware ROM; returns `OK` or `ERROR` as the result.
L0	Speaker off.
L1	Low speaker volume.
L2	Medium speaker volume.
L3	High speaker volume.
M0	Speaker off.
M1	Speaker on until carrier detected.
M2	Speaker always on.
M3	Speaker on until carrier detected, except during dialing.
O0	Enter online state.
O1	Enter online state and initiate equalizer retrain.
Q0	Modem returns result codes.
Q1	Modem does not return result codes.
Sr	Set pointer to register r.
Sr=*n*	Set register r to value *n*.
V0	Display result codes in numeric form.
V1	Display result codes in verbose form (as words).
X0	Enable features represented by result codes 0-4.
X1	Enable features represented by result codes 0-5, 10.
X2	Enable features represented by result codes 0-6, 10.
X3	Enable features represented by result codes 0-5, 7, 10.

Item	Description
X4	Enable features represented by result codes 0-7, 10.
Y0	Disable long space disconnect.
Y1	Enable long space disconnect.
Z	Reset modem
&C0	Assume data carrier always present.
&C1	Track presence of data carrier.
&D0	Ignore DTR signal.
&D1	Assume command state when an on-to-off transition of DTR occurs.
&D2	Hang up and assume command state when an on-to-off transition of DTR occurs.
&D3	Reset when an on-to-off transition of DTR occurs.
&F	Recall the factory settings as the active configuration.
&G0	No guard tone.
&G1	500 Hz guard tone.
&G2	1800 Hz guard tone.
&J0	RJ-11/RJ41/RJ45S telco jack.
&J1	RJ-11/RJ-13 telco jack.
&P0	Pulse dial with make/break ratio 39/61.
&P1	Pulse dial with make/break ratio 33/67.
&Q0	Operate in asynchronous mode.
&Qn	Operate in synchronous mode *n*
&R0	Track CTS according to RTS.
&R1	Ignore RTS; always assume presence of CTS.
&S0	Assume presence of DSR signal.
&S1	Track presence of DSR signal.
&T0	Terminate test in progress.
&T1	Initiate local analog loopback.
&T3	Initiate digital loopback.
&T4	Grant request from remote modem for remote data link (RDL).
&T5	Deny request from remote modem for RDL.
&T6	Initiate remote digital loopback.
&T7	Initiate remote digital loopback with self-test.
&T8	Initiate local analog loopback with self-test.
&V	View active configuration, user profiles, and stored numbers.
&Wn	Save storable parameters of active configuration as user profile *n*.
&X0	Modem provides transmit clock signal.
&X1	Data terminal provides transmit clock signal.
&X2	Receive carrier provides transmit clock signal.
&Yn	Recall user profile *n*.
&Zn=*x*	Store phone number *x* in location *n*.

S-register summary:

The S-registers, their ranges, and their descriptions are outlined in the following table.

Table 102. S-register descriptions

Register	Range	Description
S0	0-255	Select number of rings before answer.
S1	0-255	Ring count (incremented with each ring).
S2	0-127	Define escape sequence character (ASCII).
S3	0-127	Define carriage return character (ASCII).
S4	0-127	Define line feed character (ASCII).
S5	0-32, 127	Define backspace character (ASCII).
S6	2-255	Select wait-time in seconds before blind dialing.

Table 102. S-register descriptions (continued)

Register	Range	Description
S7	1-55	Select wait-time in seconds for carrier/dial tone.
S8	0-255	Select duration in seconds of comma.
S9	1-255	Carrier detect response time in .1 second increments (10 = 1 second).
S10	1-255	Delay between carrier loss and hang-up in .1 second increments.
S11	50-255	Duration/spacing of tones in milliseconds.
S12	50-255	Escape sequence guard time in .02 second intervals.
S13	—	Reserved.
S14	—	Reserved.
S15	—	Reserved.
S16	—	Reserved - functions for this register are controlled by the **&T** commands).
S17	—	Reserved.
S18	0-255	Test timer duration in seconds.
S19	—	Reserved.
S20	—	Reserved.
S21	—	Reserved.
S22	—	Reserved.
S23	—	Reserved.
S24	—	Reserved.
S25	0-255	Select DTR change detect time in .01 second intervals.
S26	0-255	RTS to CTS delay in .01 second intervals.
S27	—	Reserved.

Result codes for asynchronous adapters:

The result codes returned by asynchronous adapters, including their numbers, words, and descriptions, are identified in the following table.

Table 103. Asynchronous adapter result codes

Number	Word	Description
0	OK	Command executed.
1	CONNECT	Connection established at 0-300 bps.
2	RING	Ring signal detected.
3	NO CARRIER	Carrier signal lost or not detected.
4	ERROR	Invalid command, checksum, error in command line, or command line too long.
5	CONNECT 1200	Connection established at 1200 bps.
6	NO DIALTONE	No dial tone detected.
7	BUSY	Busy signal detected.
8	NO ANSWER	No response when dialing a system.
9	CONNECT 2400	Connection established at 2400 bps.

Dial modifiers:

Dial modifiers and their descriptions can be referenced in the following table.

Item	Description
0-9 # * A-D	Digits and characters for dialing.
P	Pulse dial.
T	Tone dial.
,	Delay processing of next character.
!	Hookflash.
@	Wait for silence.
W	Wait for dial tone.
;	Return to command state after dialing.
R	Reverse mode.
S=*n*	Dial number stored at location *n*.

Modem assistance:

When encountering problems with your modem, assistance is available in the following places.

- Your local area representative can assist in the modem configuration.
- There are many different support options available to customers in the Support Services offered, including on-site assistance or over-the-phone support. Contact your nearest service representative for assistance.
- Perhaps an often overlooked source of help is the modem manufacturer themselves. Most manufacturers have some type of online assistance for their products.

/usr/lib/uucp/Dialers.samples file entries:

These example file entries are supplied without any warranty and will work as is for the models mentioned, but might not meet your specific needs.

Some modifications will be required to meet your individual needs. Consult your modem manual for a more detailed explanation of the settings.

To use the settings to program the modem, you need an entry in the `/usr/lib/uucp/Systems` file such as:

```
hayes Nvr HayesPRGM Any
```

The `/usr/lib/uucp/Devices` file should have an entry such as:

```
HayesPRGM tty0 - 2400 HayesProgrm2400
```

With the above two entries made, use the following **cu** command to program the modem:

```
cu -d hayes
```

```
# COMPONENT_NAME: cmduucp
#
#
# (C) COPYRIGHT International Business Machines Corp. 1994
# Licensed Materials - Property of IBM
# US Government Users Restricted Rights - Use, duplication or
# disclosure restricted by GSA ADP Schedule Contract with IBM
# Corp.
#############################################################
# Motorola UDS Modem
#
# Use udsmodemPROGRAM to program the modem.
# Port needs to have rts/cts set.
# Use uds or hayes dialer.
#
```

```
# The "udsmodemPROGRAM" line should be a single, continuous line
#
############################################################
udsmodemPROGRAM =,-, "" \dAT&FQ2\r\c OK
ATE0Y0&C1&D2&S1%B5%E0*LC\r\c OKAT&K3&W\r\c OK

uds =,-, "" \dAT\r\c OK\r ATDT\T\d\r\c CONNECT

################################################################
#
# IBM 7855 Model 10
# Use IBMProgrm to program the modem.
# This sets rts/cts flow control, turns
# off xon/xoff, and sets the DTE speed at 19,200 bps.
# The modem will connect at the appropriate speed and
# flow control with the server.
# Port needs to have rts/cts set.
#
# The "IBMProgrm" line should be a single, continuous line
#
############################################################
IBMProgrm =,-, "" \dATQ0\r\c OK AT&F\r\c OK ATM1\r\c OK
AT&D3\r\c OK AT&C1\R2\Q2\M14\r\c OK AT&B8N1L0E0\A0\r\c OK
ATS0=1\r\c OK ATQ1&W0&Y0\r\c ""

#####################################################
# The following are used for Dialing out on a 7855
# regular ACU device. We have to turn on result
# codes (Q0) because they are turned off when we
# programmed it. (Keeps all upper case login from
# happening on dial in attempts.)
# We have to have an extra "\" before "\N" because
# the BNU programs strips it if it's before an "N".
#####################################################
ibm =,-, "" \dATQ0\r\c OK ATDT\T\d\r\c CONNECT

# IBM 7855 ECL (No Compression)
ibmecl =,-, "" \dAT\\N3%C0Q0\r\c OK ATDT\T\d\r\c CONNECT

# IBM 7855 ECLC (Compression)
ibmeclc =,-, "" \dAT\\N3%C1Q0\r\c OK ATDT\T\d\r\c CONNECT

# IBM 7855 ECLC Compression with 256 byte block size
ibmeclc256 =,-, "" \dAT\\N3%C1Q0\A3\r\c OK ATDT\T\d\r\c CONNECT

# IBM 7855 No Compression 1200bps
ibm_ne12 =,-, "" \dATQ0\\N0&A2%C0\r\c OK ATDT\T\d\r\c CONNECT

# IBM 7855 No Compression 2400bps
ibm_ne24 =,-, "" \dATQ0\\N0&A3%C0\r\c OK ATDT\T\d\r\c CONNECT

# IBM 7855 No Compression 9600bps
ibm_ne96 =,-, "" \dATQ0\\N0&A6%C0\r\c OK ATDT\T\d\r\c CONNECT

# IBM 7855 No Compression 19200bps
ibm_ne192 =,-, "" \dATQ0\\N0%C0\r\c OK ATDT\T\d\r\c CONNECT

# IBM 7855 No Compression 12000bps
ibm_ne120 =,-, "" \dATQ0\\N3%C0&AL8\r\c OK ATDT\T\d\r\c CONNECT

# IBM 7855 No Compression 1200bps (Dial Quietly)
ibmq12 =,-, "" \dATQ0\r\c OK AT&A2M0DT\T\d\r\c CONNECT

# IBM 7855 No Compression 2400bps (Dial Quietly)
ibmq24 =,-, "" \dATQ0\r\c OK AT&A3M0DT\T\d\r\c CONNECT

# IBM 7855 No Compression 9600bps (Dial Quietly)
```

```
ibmq96 =,-, "" \dATQ0\r\c OK AT&A6M0DT\T\d\r\c CONNECT

# IBM 7855 No Compression 19200bps (Dial Quietly)
ibmq192 =,-, "" \dATQ0\r\c OK ATM0DT\T\d\r\c CONNECT

################################################################
#
# Intel 9600EX Modem
# Use IntelProgram to program the modem.
# This sets rts/cts flow control, and turns
# off xon/xoff.
# Port needs to have rts/cts set. (Use hayes dialer)
#
# The "IntelProgram" line should be a single, continuous line
#
################################################################
#IntelProgram =,-, "" \d\dAT\r\c OK AT&F\r\c OK AT&S1M1\r\c OK
AT&D3\r\c OKAT&C1\r\c OK ATL0E0Y0&Y0\X1\r\c OK ATS0=1\r\c OK
AT&W\r\c OK

################################################################
# Practical Peripherals 1440FXMT Modem
# Use PracPerProgram144 to program the modem.
# This sets rts/cts flow control, and turns
# off xon/xoff. (Use hayes dialer)
# DTE speed will be locked at connect speed when
# the modem is programmed. (Suggestion: 38400 baud)
#
# The "PracPerProgram144" line should be a single, continuous
# line
################################################################
PracPerProgram144 =,-, "" \d\dAT\r\c OK AT&F\r\c OK ATM1\r\c OK
AT&D3\r\c OKAT&C1&K3\r\c OK ATQ2E1&Q9\r\c OK ATS0=1S9=20\r\c OK
AT&W\r\c OK

################################################################
# Practical Peripherals 9600 bps Modem
# Use PracPerProgram9600 to program the modem.
# This sets rts/cts flow control, and turns
# off xon/xoff. (Use hayes dialer)
#
# The "PracPerProgram144" line should be a single, continuous
# line
################################################################
PracPerProgram9600 =,-, "" \d\dAT\r\c OK AT&F\r\c OK ATM1\r\c OK
AT&D3\r\c OKAT&C1&K3\r\c OK ATL0E0\r\c OK ATS0=1S9=20\r\c OK
AT&W\r\c OK

################################################################
# Practical Peripherals 2400 bps Modem
# Use PracPerProgram to program the modem
#
# The "PracPerProgram2400" line should be a single, continuous
# line
################################################################
PracPerProgram2400 =,-, "" \d\dAT\r\c OK AT&F\r\c OK ATM1\r\c OK
AT&D3\r\c OKAT&C1\r\c OK ATL0E0\r\c OK ATS0=1S9=20\r\c OK AT&W\r\c OK

################################################################
# Hayes 2400 bps Modem
# Use HayesProgrm2400 to program the modem.
# (Use hayes dialer to dial)
#
# The "HayesProgrm2400" line should be a single, continuous line
#
################################################################
HayesProgrm2400 =,-, "" \d\dAT\r\c OK AT&F\r\c OK ATM1\r\c OK
```

```
AT&D3\r\c OKAT&C1\r\c OK ATL0E0\r\c OK AT S0=1\r\c OK AT&W\r\c OK

###############################################################
# Telebit t2000 Trailblazer Plus
# Use TelebitProgrm to program the modem
# This sets rts/cts flow control, and turns
# off xon/xoff and sets the Default DTE speed at
# 19,200 bps.
# Port needs to have rts/cts set.
# This sets modem to send PEP tones last as they can
# can confuse some other modems.
#
# The "TelebitProgram" line should be a single, continuous line
#
###############################################################
TelebitProgram =,-, "" \dAT&F\r\c OK
ats2=255s7=60s11=50s41=2s45=255s51=254s52=2s54=3s58=2s64=1s66=1\r\c OK
ATs69=1s92=1s96=0s105=0s110=1s111=30s130=3s131=1F1M0Q6TV1W0X3Y0\r\c OK
ATE0&W\r\c OK
# Telebit T2000 dialers Entries:
# Forces a PEP connection:
tbfast =,-, "" \dATs50=255s7=60\r\c OK\r ATDT\T\r\c
CONNECT-\d\c-CONNECT

# 2400bps connection:

#tb2400 =,-, "" \dATs50=3\r\c OK\r ATDT\T\r\c CONNECT

# 2400 MNP:
tb24mnp =,-, "" \dAT\r\c OK ATS0=0S95=2S50=3S41=0\r\c OK
ATDT\T\r\c CONNECT

# 1200bps connection:#tb1200 =,-, "" \dATs50=2\r\c OK\r
ATDT\T\r\c CONNECT

# 1200 MNP:
tb12mnp =,-, "" \dAT\r\c OK ATS0=0S95=2S50=2S41=0\r\c OK
ATDT\T\r\c CONNECT

###############################################################
# Telebit WorldBlazer
# WORLDBLAZERProgram sets the DTE speed at 38400, but
# you could set it higher if the DTE connection can
# handle it. We answer with PEP tones last so as not
# to confuse other modems. This turns off xon/xoff
# and turns on RTS/CTS flow control. The port should
# be locked to 38400 with these settings, and needs
# to have RTS/CTS turned on.
#
# The "WORLDBLAZERProgram" line should be a single, continuous
# line
###############################################################
WORLDBLAZERProgram =,-, "" \dAT\r\c AT AT&F3M0\r\c AT
ATs51=253s92=1\r\c ATAT&W\r\c AT

#####################################################
# ACU Dialers for various BAUD rates for the
# WorldBlazer - each sets the modem to attempt to
# connect at a specific speed and lower. The
# WBlazer will accept whatever the remote modem can
# do. You will want to use PEP for other Telebits,
# so use WBlazer38400 or WBlazer19200 for those
#####################################################
# WBlazer =,-, "" \dAT\r\c OK ATDT\T\d\r\c CONNECT
WBlazer38400 =,-, "" \dATs50=255\r\c OK ATDT\T\d\r\c CONNECT
WBlazer19200 =,-, "" \dATs50=255\r\c OK ATDT\T\d\r\c CONNECT
# WBlazer14400 attempts to negotiate a V.42bis connection.
```

```
WBlazer14400 =,-, "" \dATs50=7\r\c OK ATDT\T\d\r\c CONNECT

# For a V.32 connection:
WBlazer9600 =,-, "" \dATs50=6\r\c OK ATDT\T\d\r\c CONNECT

# For a V.22 connection:
WBlazer2400 =,-, "" \dATs50=3\r\c OK ATDT\T\d\r\c CONNECT

# For a 1200 bps connection:
WBlazer1200 =,-, "" \dATs50=2\r\c OK ATDT\T\d\r\c CONNECT
```

128-Port modem cabling considerations:

This operating system does not require DSR in modem-control applications, and because almost all of today's modems have auto-answering capability, the Ring Indicator signal is generally unnecessary.

The 10-pin RJ-45 plugs is not the predominant cabling subsystem and may be difficult to obtain in the retail market. The TTY subsystem of this operating system provides an optional feature called ALTPIN, which swaps the logical functions of DSR (Data Set Ready) with DCD (Data Carrier Detect) for a port. When ALTPIN is enabled, DCD becomes available on pin 1 of an 8-pin RJ-45 connector (equivalent to pin 2 of a 10-pin connector).

If you wish to build an 8-wire modem cable for the 128-port RAN, use the 8-pin RJ-45 plug wired as in the following table:

Table 104. 128-Port modem cabling

Item	Description	Modem
SYSTEM END CONNECTOR 8-pin RJ-45	**DEVICE END**RI	22
1	DSR	6
2	RTS	4
3 (Chassis)	GND	SHELL
4	TxD	2
5	TxD	3
6 (Signal)	GND	7
7	CTS	5
8	DTR	20
	CD	8

Note: The physical location of DSR and CD may be swapped with the ALTPIN parameter when enabled using the stty-cmxa command.

The following table shows the asynchronous signal communication between the system unit and an attached modem. Here, data is being sent from the system unit to a remote system.

Table 105. Asynchronous signal communication

DEVICE	SIGNAL	ON/OFF	MEANING
Computer	DTR	+	Hey, modem, are you ready to connect to another system?
Modem	DSR	+	Yes, I am ready. Go ahead and dial.
Modem	DCD	+	I've got the other system on the phone.
Computer	RTS	+	OK, can I send data now?
Modem	CTS	+	Sure, go ahead.
Computer	TxD		Sending data out to modem.
Modem	RxD		I've received the data.
Modem	CTS	–	Don't send me any more data, I'm sending it out...
Modem	CTS	+	OK, I'm ready for more data, let me have it!
Transmit data steps may be repeated until... Computer	DTR	–	FINISHED! Go ahead and hang up.
Modem	DCD	–	OK.
Here is the signal communication between a RS/6000® and a modem about to receive an incoming call from another system. Computer	DTR	+	I'm ready and have "enabled" the port for dial-in.
Modem	DSR	+	I'm ready also but I'm just waiting around for a call.
Someone calls in! Modem	DCD	+	Somebody called in and I've got them on the line.
Modem	CTS	+	I've got data from another box, can I send you data now?
Computer	RTS	+	I'm ready to receive. Go ahead and send.
Modem	RxD		Here it comes!
Modem continues to send data until... Computer	RTS	–	WAIT! My buffer is full, don't send any more data.
Computer	RTS	+	I'm OK now. Send me more data.
Modem	DCD	–	Call has ended.
Computer	DTR	–	OK, please hang up.

stty-cxma terminal options

stty-cxma is a utility program that sets and displays the terminal options for the PCI 2-, 8-, and 128-port adapters and is located in `/usr/lbin/tty` directory.

The format is:

```
stty-cxma [-a] [option(s)] [ttyname]
```

With no options, **stty-cxma** displays all special driver settings, modem signals, and all standard parameters displayed by **stty(1)** for the tty device referenced by standard input. Command options are provided to change flow control settings, set transparent print options, force modem control lines, and display all tty settings. Any unrecognized options are passed to **stty(1)** for interpretation. The options are:

-a Displays all of the unique adapter option settings, as well as all of the standard tty settings reported by the **stty -a** command.

ttyname
Sets and displays options for the given tty device, instead of standard input. This form can be used with a tty path name prefixed by **/dev/** or with a simple tty name beginning with tty. This option may be used on a modem control line when no carrier is present.

The following options specify transient actions to be performed immediately:

break Sends a 250 ms break signal out on the tty line.

flush Indicated an immediate flush (discard) of tty input and output.

flushin
Flushes tty input only.

flushout
Flushes tty output only.

The following options specify actions that are reset when the device is closed. The device will use the default values the next time it is opened.

stopout
Stops output exactly as if an XOFF character was received.

startout
Restarts stopped output exactly as if an XON character was received.

stopin Activates flow control to stop input.

startin Releases the flow control to resume stopped input.

[-]dtr [drop]
Raises the DTR modem control line, unless DTR hardware flow control is selected.

[-]rts [drop]
Raises the RTS modem control line, unless RTS hardware flow control is selected.

The following options remain in effect until the system is rebooted or until the options are changed.

[-]fastcook
Performs cooked output processing on the intelligent card to reduce host CPU usage, and increase raw mode input performance.

[-]fastbaud
Alters the baud rate tables, so 50 baud becomes 57,600 baud, 75 baud becomes 76,800 baud, 110 baud becomes 115,200 baud, and 200 baud becomes 230,000 baud for supported devices.

[-]rtspace
Enables/disables RTS hardware input flow control, so RTS drops to pause remote transmission.

[-]ctspace
Enables/disables CTS hardware output flow control, so local transmission pauses when CTS drops.

[-]dsrpace
Enables/disables DSR hardware output flow control, so local transmission pauses when DSR drops.

[-]dcdpace
Enables/disables DCD hardware output flow control, so local transmission pauses when DCD drops.

[-]dtrpace
Enables/disables DTR hardware input flow control, so DTR drops to pause remote transmission.

[-]forcedcd
Disable [re-enable] carrier sense, so the tty may be opened and used even when carrier is not present.

[-]altpin
Maps the RJ-45 connector pinouts to the default 10-pin connector values or the 8-pin connector values. When this parameter is **enabled**, the location of DSR and DCD is switched so that DCD is available when using an 8-pin RJ-45 connector instead of the 10-pin RJ-45 connector. (Default=**disable**.)

Possible values:

enabled (specifies 8-pin connector values)

disable (specifies 10-pin connector values)

startc *c*
Sets the XON flow control character. The character may be given as a decimal, octal, or hexadecimal number. Octal numbers are recognized by the presence of a leading zero, and hexadecimal numbers are denoted by a leading 0x. For example, the standard XON character, CTRL-Q, can be entered as 17 (decimal), 021 (octal), or 0x11 (hexadecimal).

stopc*c* Sets the XOFF flow control character. The character may be given as a decimal, octal, or hexadecimal number (see **startc** for format of octal and hexadecimal numbers).

astartc*c*
Sets auxiliary XON flow control character. The character may be given as a decimal, octal, or hexadecimal number (see **startc** for format of octal and hexadecimal numbers).

astopc*c*
Sets auxiliary XOFF flow control character. The character may be given as a decimal, octal, or hexadecimal number (see **startc** for format of octal and hexadecimal numbers).

[-]aixon
Enables auxiliary flow control, so that two unique characters are used for XON and XOFF. If both XOFF characters are received, transmission will not resume until both XON characters are received.

[-]2200flow
Uses 2200 style flow control on the port. The 2200 terminals support an attached printer and use four flow control characters: terminal XON (0xF8), printer XON (0xF9), terminal XOFF (0xFA) and printer XOFF (0xFB).

[-]2200print
Determines how these flow control characters are interpreted. If 2200print is set, run independent flow control for terminal and transparent print devices. Otherwise, terminal and printer flow control are logically tied together. If either XOFF character is received, all output is paused until the matching XON character is received.

maxcps*n*
Sets the maximum characters per second (cps) rate that characters are output to the transparent print device. The rate chosen should be just below the average print speed. If the number is too low, printer speed will be reduced. If the number is too high, the printer uses flow control, and user entry time is reduced. The default is 100 cps.

maxchar*n*
Sets the maximum number of transparent print characters the driver places in the output queue. Reducing this number increases system overhead; increasing this number delays operator keystroke echo times when the transparent printer is in use. The default is 50 characters.

bufsize*n*
Sets the driver's estimate of the size of the transparent printer's input buffer. After a period of inactivity, the driver bursts this many characters to the transparent printer before reducing to the maxcps rate. The default is 100 characters.

onstr*s*
Sets the terminal escape sequence to turn transparent printing on. The strings can be composed of standard ASCII printing and nonprinting characters. Control (nonprinting) characters must be entered by their octal values, and must consist of three digits preceded by a backslash character. For example, the Escape character, 33 octal, should be entered as \033. If transparent printing is turned on by the string <Esc>[5i (ANSI standard), it would be entered as: \033[5i.

offstr*s*
Sets the terminal escape sequence to turn transparent printing off. Refer to **onstr***s* for the format of the strings.

term*t* Sets the transparent printer on/off strings to values found in the internal default table. Internal defaults are used for the following terminals: adm31, ansi, dg200, dg210, hz1500, mc5, microterm, multiterm, pcterm, tvi, vp-a2, vp-60, vt52, vt100, vt220, wyse30, wyse50, wyse60, or wyse75. If the terminal type is not found in the internal default table, ditty reads the terminfo entry for the terminal type and sets transparent print on/off strings to values given by the mc5/mc4 attributes found in the terminfo entries.

Asynchronous Point-to-Point Protocol subsystem

The **Asynchronous Point-to-Point Protocol (PPP)** subsystem provides an alternative to SLIP.

PPP provides a standard method for transporting multiprotocol datagrams over point-to-point media. **PPP** is comprised of three main layers:

1. A method for encapsulating multiprotocol datagrams. **PPP** supports the TCP/IP network layer protocols.
2. A **Link Control Protocol (LCP)** for establishing, configuring, and testing the data-link connection. **PPP** implements this through streams kernel extensions.
3. A family of **Network Control Protocols (NCPs)** for establishing and configuring different network layer protocols. **PPP** supports **Internet Protocol Control Protocol (IPCP/IPv6CP)** for negotiating a TCP/IP connection.

This implementation of **PPP** supports the following Request for Comments (RFCs):

- RFC 1661, *The Point-to-Point Protocol, LCP*
- RFC 1332, *The PPP Internet Protocol Control Protocol (IPCP)*
- RFC 1662, *PPP in HDLC-like Framing*
- RFC 1334, *PPP Authentication Protocols*
- RFC 1990, *PPP Multilink*
- RFC 2472, *IP Version 6 over PPP*

PPP differentiates between client and server. This operating system can act as both a client and a server. The distinction is made to simplify configuration. **PPP** servers tend to allocate a pool of IP/IPv6CP addresses among the connections that are being made. There is some correlation between the media devices. This implementation of **PPP** breaks this correlation. All server **PPP** connections are allocated on a first-available basis. This facilitates the separation of **PPP** from the media. The attachment process must request to be linked to the proper type of link.

PPP user-level processes

The **Asynchronous Point-to-Point Protocol** on this operating system utilizes three user-level processes.

1. A control daemon (**pppcontrold**) run by root under the System Resource Controller (**startsrc -s pppcontrold**). The control daemon's function encompasses loading and configuring all kernel extensions associated with the subsystem. It remains running as long as **PPP** function is required by the operating system.
2. An attachment process (**pppattachd**) that binds a TTY stream to an instance of the **Link Control Protocol**, **Network Control Protocol**, and a datagram protocol. An instance of **pppattachd** exists for each active **PPP** connection in the system. Any user of the attachment process must belong to the **uucp** group and contain `/usr/sbin` within their **PATH** environment variable.
3. A dialer process (**pppdial**) that establishes an outgoing connection. The dialer is intended to be executed by **pppattachd** as the connector program. Its purpose is to interact over the asynchronous device prior to **PPP** negotiation. This interaction is defined similarly to the UUCP chat dialog format. The dialer capability is provided to assist in establishing a connection with a remote system. The actual session establishment is out of the scope of **PPP**.

Asynchronous Point-to-Point Protocol configuration

You can use SMIT to configure the **Asynchronous Point-to-Point Protocol**.

The following table shows all tasks that you may need when configuring your system. You must have root privileges to perform the tasks in this table.

At a minimum, when you initially configure your system, you must choose the following tasks from the table:

- Add a Link Configuration
- Add a Server Interface (if you are setting up the machine as a PPP server)
- Add a Demand Interface (if you want the machine to support demand connections)
- Manipulate PAP or CHAP Users/Passwords (if you want the machine to support **PPP** authentication)
- Start **PPP** to effect your changes (or Stop then Start **PPP**, if **PPP** is currently running)

Table 106. Configuring the asynchronous PPP tasks

Task	SMIT fast path
Create Link Control Configuration	`smit ppplcp`
Add a Link Configuration	`smit addlcp`
Change/Show a Link Configuration	`smit chglcp`
Remove a Link Configuration[1]	`smit rmlcp`
Create PPP IP Interfaces	`smit pppip`
Add a Server Interface	`smit addpppserver`
Change/Show a Server Interface	`smit listserver`
Remove a Server Interface[1]	`smit rmlistserver`
Add a Demand Interface	`smit addpppdemand`
Change/Show a Demand Interface	`smit listdemand`
Remove a Demand Interface[1]	`smit rmlistdemand`
Manipulate PAP users/passwords	`smit ppppap`
Add a PAP User	`smit addpapuser`
Change/Show a PAP User	`smit listpapuser`
Remove a PAP User	`smit rmpapuser`
Manipulate CHAP users/passwords	`smit pppchap`
Add a CHAP User	`smit addchapuser`
Change/Show a CHAP User	`smit listchapuser`
Remove a CHAP User	`smit rmchapuser`
Start **PPP**[2]	`smit startppp`

Table 106. Configuring the asynchronous PPP tasks (continued)

Task	SMIT fast path
Stop **PPP**[3]	`smit stopppp`
PPP IPv6 Interfaces	`smit pppipv6`
Add a **PPP** IPv6 Server Interface	`smit addpppv6server`
Show or change a **PPP** IPv6 interface.	`smit listv6server`
Remove a **PPP** IPv6 interface.	`smit rmlistv6server`
Add a **PPP** IPv6 client interface.	`smit addpppv6client`
Show or change a **PPP** IPv6 client interface.	`smit listpppv6client`
Remove a **PPP** IPv6 client interface.	`smit rmlistpppv6client`
Add a **PPP** IPv6 demand interface	`smit addpppv6demand`
Show or change a **PPP** IPv6 demand interface.	`smit listpppv6demand`
Remove a **PPP** IPv6 demand interface.	`smit rmlistpppv6demand`
PPP IP and IPv6 Interfaces	`smit pppipv4_6`
Add a **PPP** IP/IPv6 Server Interface	`smit addpppv4_6server`
Show or change a **PPP** IP/IPv6 interface.	`smit listv4_6server`
Remove a **PPP** IP/IPv6 interface.	`smit rmlistv4_6server`
Add a **PPP** IP/IPv6 client interface.	`smit addpppv4_6client`
Show or change a **PPP** IP/IPv6 client interface.	`smit listpppv4_6client`
Remove a **PPP** IP/IPv6 client interface.	`smit rmlistpppv4_6client`
Add a **PPP** IP/IPv6 demand interface	`smit addpppv4_6demand`
Show or change a **PPP** IP/IPv6 demand interface.	`smit listpppv4_6demand`
Remove a **PPP** IP/IPv6 demand interface.	`smit rmlistpppv4_6demand`

Note:

1. Selecting this task destroys the existing information.
2. An alternative way to start **PPP** is to issue the **startsrc -s pppcontrold** command. However, the SMIT interface also allows you to set **PPP** to start at boot time.
3. An alternative way to stop **PPP** is to issue the **stopsrc -s pppcontrold** command. However, the SMIT interface also allows you to have **PPP** not start at boot time.

Enabling PPP SNMP

PPP can interact with the TCP/IP SNMP daemon to report PPP link layer configuration information as well as information about active **Link Control Protocol (LCP)** interfaces.

Providing that both the TCP/IP SNMP and the SNMP management software are configured correctly, **PPP** SNMP enables:

- retrieval of **PPP** Link Configuration information (such as Maximum Receive Unit size and Asynchronous Character Mapping)
- setting of **PPP** Link Configuration information
- retrieval of LCP interface information for active LCP links
- changing of the state of active LCP links can be changed to "down" by setting the appropriate **ifAdminStatus** Management Information Base (MIB) object

Not all objects defined by RFC1471 for the **PPP** MIB are supported. Only the **pppLink** table applies to the **PPP** subsystem, thus the **pppLqr** and **pppTests** portions are not supported. The **pppLink** portion is supported with the following exceptions:

- The **pppLinkConfigMagicNumber** object is read only. In **PPP**, magic number negotiation is always performed and cannot be disabled.

- The **pppLinkConfigFcsSize** object is read only. **PPP** only supports FCS sizes of 16 with this operating system.

By default, SNMP for **PPP** is disabled. To enable **PPP** SNMP, you can use the following procedure. You must have root privileges to perform this procedure.

Note: The following procedure assumes that **PPP** Link Configuration is already set. If not, perform the procedure described in "Asynchronous Point-to-Point Protocol configuration" on page 579 before enabling **PPP** SNMP.

1. Start the SMIT Interface and display the Change/Show a Link Configuration screen by entering:
   ```
   smit chglcp
   ```
2. Toggle the Enable **PPP** SNMP subagent field to yes.
3. Accept your changes and exit SMIT.

PPP SNMP is not enabled until **PPP** is restarted.

- If **PPP** is currently running,
  1. Stop **PPP** using the `smit stopppp` fast path (see the table in "Asynchronous Point-to-Point Protocol configuration" on page 579).
  2. Periodically check to see if the subsystem has completed shutdown by entering:
     ```
     lssrc -s pppcontrold
     ```

     The amount of time it takes to completely stop the subsystem is dependent on the number of links defined in the **PPP** configuration. The subsystem is completely shut down when the output of this command shows a status of `inoperative`.
  3. Start **PPP** using the `smit startppp` fast path (see the table in "Asynchronous Point-to-Point Protocol configuration" on page 579).
- If **PPP** is not currently running, start **PPP** using the `smit startppp` fast path (see the table in "Asynchronous Point-to-Point Protocol configuration" on page 579).

Serial Line Internet Protocol

Serial Line Internet Protocol (SLIP) is the protocol which TCP/IP uses when operating through a serial connection.

It is commonly used on dedicated serial links and dial-up connections that operate at speeds between 1200bps and 19.2Kbps or higher.

Note: To use baud rates higher than 38400, specify a baud rate of 50 in the `/etc/uucp/Devices` file for the desired tty, then change the SMIT configuration for that tty to reflect the actual baud rate desired.

For example. to run the **cu** command on tty0 with a baud rate of 115200, use the following procedure:

1. Ensure the hardware supports the baud rate.
2. Edit `/etc/uucp/Devices` to include the following line:
   ```
   Direct tty0 - 50 direct
   ```
3. Enter the `smit chtty` fast path.
4. Select tty0.
5. Change the baud rate to 115200.
6. Exit SMIT.

Configuring SLIP

There are two recommended steps to follow during **SLIP** configuration.

Using this two-step approach separates the hardware and machine-dependent configuration requirements from the SLIP software and command syntax problems.

1. Use ATE or the **cu** utility to accomplish a successful login at the remote system. This proves the usability and correctness of the physical link.

   It is important to verify the operability of any modems that are involved in a **SLIP** link, as they are the most frequent cause of problems during the setup phase.
2. After you establish an error-free login to the remote system, by using ATE or the **cu** command, you can begin the **SLIP** configuration.

SLIP modem considerations

When configuring modems for **SLIP**, it is important that these changes be made on both ends of the communication link.

Both the local and remote modems must be configured exactly the same.

1. The modem must acknowledge the presence of DTR.

   Referencing the local modem, if DTR is assumed or ignored, the modem can never perform a hang-up. It can only close the line or hang up when it recognizes the loss of carrier from the other end. This means that disconnects can occur only when instigated by the other end. The AT commands &D2 or &D3 are proper settings for most Hayes-compatible modems.
2. The modem must never force, assume, or ignore data carrier detect (DCD).

   DCD must follow or track the real condition. This means that carrier will exist after a bona fide connection to the other end (modem) across the switched telephone line. This also applies to a dedicated line. &C1 is the suggested setting for most Hayes-compatible modems.
3. The modem must never force, assume, or ignore a clear to send (CTS) signal.

   CTS must track or follow request To send (RTS). If CTS is forced true, the port open will fail whenever a **getty** is put upon the port or when RTS flow control protocol is added to the port.
4. Modems should be configured to turn off automatic repeat request (ARQ) codes if problems arise during **slattach** dial attempts.

   If, the modems repeatedly fail to make a connection during **slattach** dial-in attempts, the user should check the modem configurations and turn off the ARQ codes if they are currently on. In most Hayes-compatible modems, this is the &A0 setting.

   Disabling ARQ result codes does not affect error-controlled connections nor does it keep the modem from returning standard CONNECT messages (if result codes are enabled) as needed for the **slattach** dial string.
5. ECL (Error Checking on the Link) is critical.

   Either BOTH modems or NEITHER modem can use it. Normally, both modems must agree on its usage during the connect session. If ECL is chosen, the physical telephone line must be good enough to allow a recovery from a data error before the TCP/IP timers expire while awaiting an acknowledge packet for the last data sent across the **SLIP** link.
6. Data Compression across the link.

   It is acceptable to use data compression across the link as long as it is totally handled by the modems. **SLIP** does not perform any type of compression. If data compression is invoked, it is much better to have two modems of the exact same type; this ensures that each will perform the compression in the same manner and same time frame.

Manually programming modems using the cu command

Use the following procedure to manually program modems attached to the system unit.

- The UNIX-to-UNIX Copy Program (UUCP) must be installed on the system. Use the **lslpp -f | grep bos.net.UUCP** command to verify installation.
- A modem must be attached to the system and powered on.
- Root user authority is needed to change the appropriate files.

1. Add the following line to the /etc/uucp/Devices file if it does not already exist (replace # with the number for your port).

   ```
   Direct tty# - Any direct
   ```

   Note: Any line in the Devices file which begins with a # sign in the leftmost column is a comment.
2. Save and exit the file.
3. Type the following command on the command line:

   ```
   cu -ml tty#
   ```
4. A connected message should appear on the screen indicating that the modem is connected and ready to be programmed.
5. Type AT and press Enter. The modem will respond with OK. If there is no response from the modem or if characters typed do not appear on the screen, check the following:
   - Verify modem cabling connections.
   - Verify that modem is powered on.
   - Observe the modem front panel lights when you press Enter. If the Receive Data (RD) and Send Data (SD) lights flash, then the modem is communicating with the system and the problem may lie with the current modem settings. If the lights do not flash, then the problem is with the modem connection.
   - Type the following and see if the condition changes:

     ```
     ATE1 <enter>
     ATQ0 <enter>
     ```

     ATE1 turns the echo mode on which displays any typed characters to the screen. ATQ0 enables the displaying of result codes.
6. Program the modem using the settings shown in the previous section, "Modem Considerations." The following example demonstrates how to program and save basic settings for a Hayes-compatible modem. Enter:

   ```
   AT&F    <enter>
   AT&D2   <enter>
   ATS0=1  <enter>
   ATS9=12 <enter>
   AT&C1   <enter>
   AT&W    <enter>
   ~.      <enter>
   ```

   Where &F is used to reset the modem to factory defaults, &D2 sets DTR, S0 and S9 set register values, &C1 set carrier, and &W writes the settings to the modem. The tilde-period ends the connection.

Configuring automated modems

Users can customize their modems manually or use the **cu** utility with its associated files to create an automated modem configuration script.

- UUCP must be installed on the system. Use the **lslpp -f | grep bos.net.UUCP** command to verify installation.
- A modem must be attached to the system and powered on.
- The modem AT command string must already exist (for example, at&f&c1&d3). Users should not attempt automated modem configuration until the command string has first been tried manually using the **cu** command.
- Root user authority is needed to change the appropriate files.

The following example shows how to automatically configure a Telebit T3000 modem attached to tty0.

1. Edit the /etc/uucp/Systems file.
2. Add the following line at the end of the file. The entry should begin in the leftmost column of the file.

```
telebit Nvr TELEPROG 19200
```

3. Save and exit the file.
4. Edit the `/etc/uucp/Devices` file.
5. Add the following line at the end of the file. The entry should begin in the leftmost column of the file.

   ```
   TELEPROG tty0 - 19200 TelebitProgram
   ```

6. Save and exit the file.
7. Edit the `/etc/uucp/Dialers` file.
8. Add the following lines at the end of the file. The entries should begin in the leftmost column of the file.

   Note: The following four lines should be made into one long line:

   ```
   TelebitProgram  =,-,    "" \dAT&F\r\c OK
   ats0=1s2=255s7=60s11=50s41=2s45=255s51=252s63=1s58=2s64=1\r\c OK
   ATs69=2s105=0s111=30s255=0M0&C1Q2&D3&Q0&R3&S1&T5\r\c OK
   ATE0X12&W\r\c OK
   ```

9. Save and exit the file.
10. To begin the automated configuration, type the following command:

    ```
    cu -d telebit
    ```

    The command will fail because you are not connecting to a system. Watch the debug output of the command to see that `ATE0X12&W` is sent to the modem and that an OK is received. If so, then the modem has been successfully programmed.

Problems can arise because of incorrect values placed in the `Dialers` file or because of the modem's existing configuration. If this occurs, try programming the modem manually and enter the dialers strings (in step 8) one by one.

Configuring SLIP over a modem

To configure **Serial Line Interface Protocol (SLIP)** between two systems that communicate through a modem, you can use this procedure, which alternates between the System Management Interface Tool (SMIT) interface and the command line to complete the configuration.

For clarity, the following instructions use the names bronze and gold for the two hosts.

1. Physically connect the modems to bronze and gold.
2. To create a tty on bronze using SMIT, follow these steps:
   a. Type:

      ```
      smit maktty
      ```

   b. Select **rs232** as the type of tty you wish to create.
   c. Select an available serial port, for example `sa0` (system serial port 1).
   d. Select a port number for this tty from the list.
   e. Set the BAUD rate to the baud rate of your modem.
   f. Set Enable LOGIN to disable.
   g. Exit SMIT.
3. Create a tty on gold.

   Follow the same procedure as you did for bronze (in step 2), except set Enable LOGIN to **enable**.

   The rest of these instructions assume that the tty number on both bronze and gold is tty1.
4. Test the physical connection with ATE.
   a. On bronze, type:

      ```
      ate
      ```

b. At the Unconnected Main Menu, select the **Alter** subcommand. Set the Rate to the baud rate of your modem and the Device to tty1.
c. At the Unconnected Main Menu, select the **Connect** subcommand. When ATE prompts you for a phone number, enter the phone number of gold and press Enter.
d. At this point, you should receive a login prompt for gold. Log in.
e. Return to the connected screen, log out from gold, press Ctrl-v (to get to the ATE CONNECTED MAIN MENU), press the T key to terminate the connection, and press the Q key to exit ATE.

Note: If you do not receive a login prompt, return to step 1 and verify that your configuration is correct. Do not proceed until you can log in to gold.

5. Because the tty configuration for use with ATE is slightly different from the configuration for use with SLIP, you must make the following changes:
   a. On bronze, type:
      ```
      smit chgtty
      ```
   b. On gold, type:
      ```
      smit chgtty-pdisable tty1
      ```
   c. Select **tty1**, then select **Change/Show TTY Program**.
   d. Set Enable LOGIN to disable, then exit SMIT.
6. Add the following line to the `/usr/lib/uucp/Devices` file on both bronze and gold:
   ```
   Direct tty1 - 9600 direct
   ```
   or replace `9600` with whatever your modem speed is.
7. Create a **SLIP** network interface on bronze.
   a. Type:
      ```
      smit mkinet1sl
      ```
   b. For TTY PORT for SLIP Network Interface, select **tty1**.
   c. Specify an INTERNET ADDRESS, for example, 130.130.130.1.
   d. Specify the DESTINATION address (of gold), for example, 130.130.130.2.
   e. Specify the BAUD RATE of your modem.
   f. Specify the DIAL STRING, for example:
      - "" AT OK ATDT555-1234 CONNECT ""
      - The meaning of this command is: Use **tty1** at 9600 baud. Send AT to the modem. The modem should respond with OK. Dial the phone number `555-1234`. The modem should respond with `CONNECT`. The spaces before and after the "" characters are necessary.
   g. Exit SMIT.
8. Create a **SLIP** network interface on gold. Follow the same procedure as you did for bronze (in step 5), except exchange the INTERNET ADDRESS and the DESTINATION address.
9. Add the following two entries to the **/etc/hosts** file on both bronze and gold:
   ```
   130.130.130.1   bronze
   130.130.130.2   gold
   ```
   The name you assign must be unique. In other words, if the Token-Ring interface on bronze is already assigned the name `bronze`, assign the **SLIP** interface a name such as `bronze_slip`.

   Note: For a simplified interface to the **slattach** command, you might use the script `/usr/sbin/slipcall`.
10. Test the **SLIP** connection.
    a. On bronze, type:
       ```
       ping gold
       ```
    b. On gold, type:

```
ping bronze
```

If both tests succeed, the **SLIP** connection is ready for use. If not, return to step 5 and verify that the configuration on both bronze and gold is correct.

Configuring SLIP over a null modem cable

To configure **SLIP** between two systems that are attached using a null modem cable, you can use this procedure, which alternates between the System Management Interface Tool (SMIT) interface and the command line to complete the configuration.

For clarity, these instructions use the names bronze and gold for the two hosts.

1. Physically connect bronze and gold by the null modem cable. The following cables are required. (The cables are listed in the order they will be connected from bronze to gold.)
   a. Cable B (part number 00G0943). Serial Port Jumper Cable; two are provided with each system, except models 220, 340, and 350 do not require them.
   b. Cable D (part number 6323741, feature code 2936). Asynchronous Cable EIA-232/V.24.
   c. Cable E (part number 59F2861, feature code 2937). Printer/Terminal Interposer EIA-232 (null modem cable).
   d. Changer Adapter (both sides of the adapter are sockets).
2. Create a tty on bronze.
   a. Type:
      ```
      smit maktty
      ```
   b. Select **rs232** as the type of tty you wish to create.
   c. Select an available serial port, for example **sa0** (system serial port 1).
   d. Select a port number for this tty from the list.
   e. Set the BAUD rate to 19200. (Later, you will change this to 38400. But for now, use 19200.)
   f. Set Enable LOGIN to disable, and then exit SMIT.
3. Create a tty on gold. Follow the same steps as you did for bronze (in step 2), except set Enable LOGIN to **enable**.

   Note: The rest of these instructions assume that the tty number on both bronze and gold is tty1.
4. Test the physical connection with ATE.
   a. On bronze, type:
      ```
      ate
      ```
   b. At the Unconnected Main Menu, select the **Alter** subcommand. Set the Rate to 19200, and the Device to tty1.
   c. At the Unconnected Main Menu, select the **Connect** subcommand. When ATE prompts you for a phone number, press Enter. You should receive the message:
      ```
      ate: 0828-010 The Connect command has made a connection through port tty1
      ```
   d. Press Enter. You should receive a login prompt for gold. Log in to gold.
   e. Finally, return to the connected screen, log out from gold, press Ctrl-v (to get to the ATE CONNECTED MAIN MENU), press the T key to terminate (end) the connection, and press the Q key to exit ATE.

      Note: If you do not receive a login prompt, return to step 1 and verify that your configuration is correct. Do not proceed until you can log in to gold.
5. Because the tty configuration for use with ATE is slightly different from the configuration for use with **SLIP**, you must make the following changes:
   a. On bronze, type:
      ```
      smit chgtty
      ```

b. Select **tty1**. Set the BAUD rate to `38400`, and then exit SMIT.
c. On gold, type:
```
pdisable tty1
```
d. On gold, type:
```
smit chgtty
```
e. Select **tty1**. Set Enable LOGIN to disable, set the BAUD rate to `38400`, and then exit SMIT.

6. Add the following line to the `/usr/lib/uucp/Devices` file on both bronze and gold:
```
Direct tty1 - 38400 direct
```
7. Create a **SLIP** network interface on **bronze**.
   a. Type:
   ```
   smit mkinet1sl
   ```
   b. For TTY PORT for SLIP Network Interface, select **tty1**.
   c. Specify an INTERNET ADDRESS, for example 130.130.130.1.
   d. Specify the DESTINATION address (of gold), for example, `130.130.130.2`, and then select OK or Enter.
   e.
8. Create a **SLIP** network interface on gold. Follow the same procedure as you did for bronze (in step 5), except exchange the INTERNET ADDRESS and the DESTINATION address.
9. Add the following two entries to the `/etc/hosts` file on both bronze and gold:
```
130.130.130.1   bronze
130.130.130.2   gold
```
   The name you assign must be unique. In other words, if the Token-Ring interface on bronze is already assigned the name `bronze`, assign the **SLIP** interface a name such as `bronze_slip`.
10. Start **SLIP** on both bronze and gold. Type:
```
slattach tty1
```
11. Test the **SLIP** connection.
    a. On bronze, type:
    ```
    ping gold
    ```
    b. On gold, type:
    ```
    ping bronze
    ```

If both tests succeed, the **SLIP** connection is ready for use. If not, return to step 5 and verify that the configuration on both bronze and gold is correct.

Deactivating a SLIP connection

To deactivate a **SLIP** connection, use this procedure.

1. Type:
```
ps -ef | grep slatt
```
   Note the process numbers of processes associated with the **slattach** command.
2. For each process number, type:
```
kill process_number
```
   Do not use the **-9** flag of the **kill** command.

   If **slattach** is accidentally killed with a **-9** flag, a slip lock might remain in /etc/locks. Delete this lock file to clean up after **slattach**.

To temporarily deactivate a **SLIP** connection, do the following on both the local and remote systems:

1. Type:
```
ifconfig sl# down
```

2. List the currently running **slattach** processes using the command:
   ```
   ps -ef | grep slat
   ```

   The output might be similar to the following:
   ```
   root  1269  1    0   Jun 25  ... slattach
   ```
3. Kill the **slattach** process using its process ID. For example, to kill the **slattach** process shown in the preceding example, enter:
   ```
   kill 1269
   ```

   where 1269 is the **slattach** process ID. Do *not* remove the **slattach** process using the **-9** flag of the **kill** command.

   The **SLIP** connection is now disabled.

Activating a SLIP connection

Use these instructions to activate a **SLIP** connection that is temporarily disabled.

Run these commands on both the local and remote systems.

1. Type:
   ```
   ifconfig sl# up
   ```
2. Re-issue the **slattach** command used initially.

Removing a SLIP interface

Use these instructions to completely remove a **SLIP** interface.

After these instructions are executed both the sl# interface and its associated **slattach** process are removed. Any entries made to the `/etc/hosts` file will remain and must be removed manually.

1. To remove the **SLIP** interface and its associated **slattach** process, use the `smit rminet` fast path to access the **Available Network Interfaces** screen.
2. Select the appropriate entry from the **Available Network Interfaces** screen and select **Do**.

   Note: Any entries made to the `/etc/hosts` file will remain and must be removed manually.

SLIP troubleshooting

These commands are needed to debug **SLIP** problems.

Each command comes with an example of how the command is used for troubleshooting **SLIP** problems.

In addition, a list of common problems and error messages is provided for your reference.

netstat command:

The **netstat** command works in conjunction with the **ifconfig** command to provide a status condition of the TCP/IP network interface.

The command **netstat -in** for example uses the **-i** flag to present information on the network interfaces while the **-n** flag prints the IP addresses instead of the host names. Use this command to verify SLIP interfaces, addresses, and host names. The following section describes **netstat -in** output.

Program the modem using the settings shown in the section, “SLIP modem considerations” on page 582. The following example demonstrates how to program and save basic settings for a Hayes-compatible modem. Enter:

```
Name  Mtu   Network    Address            Ipkts     Ierrs Opkts Oerrs  Col
lo0   1536  <Link>                        2462       0    2462   0     0
lo0   1536  127        localhost.austi    2462       0    2462   0     0
```

```
tr0   1492  <Link>                        1914560    0   21000   0     0
tr0   1492  129.35.16  glad.austin.ibm    1914560    0   21000   0     0
sl0   552   1.1.1.0    1.1.1.1            48035      0   54963   0     0
sl1*  552   140.252.1  140.252.1.5        48035      0   54963   0     0
```

Notice the * next to the sl1 interface. This shows that the network interface is down or unavailable for use. The user can correct this by issuing the **ifconfig sl1 up** command if it is a valid **SLIP** interface.

netstat provides statistics concerning input and output packet counts as well as input and output errors that are helpful when troubleshooting **SLIP** connections.

For example, a user enters a **ping** to a remote host across a **SLIP** link and the **ping** command appears to hang. The user quickly run a **netstat -in** command from another command shell and notice that the Opkts are increasing but that there are no Ipkts from the remote host. This indicates that the remote system is not returning (or not receiving) the information. The user must run the same **netstat** command on the remote system to verify the receipt of the **ping** packets or rise in the error count.

The translation of hostnames versus Internet numbers is relative to name resolution and thus critical to proper operation of a **SLIP** line. To debug host name, aliases, and routing problems, use the **netstat -rn** command. The base name of the host or host name is the only name that should return from the **/etc/hosts** file. If the machine is being serviced by a name server (that is, `/etc/resolv.conf` exists), then the name-server will return the fully qualified-domain name in this command.

ifconfig command:

The **ifconfig** command is the network interface configuration tool that allows the network interface STRUCTURE to be dynamically created or deleted from the kernel memory.

This command accepts data from the command line, then builds a memory structure that conforms to the parameters. For debugging purposes, the **ifconfig** command is used to examine the status of a communications interface.

Note: Any changes made to the attributes of an interface using the **ifconfig** command will be lost when the system is rebooted.

For example, to examine the current status of the sl1 interface:

1. Enter the **netstat -i** command and examine the output selecting the appropriate sl# interface. For example, sl0, sl1, sl2, and so on.
2. Enter the **ifconfig sl#** command and examine the ifconfig output for the following key fields:

Item	Description
POINTTOPOINT flag	This flag should always be present on an operational SLIP link. If not, the link could be in a down or disconnected state. Try issuing the **ifconfig sl# up** and the **ifconfig sl#** commands again to see if its condition changes.
UP flag	Indicates that the network sl# interface is activated and should be operational.
RUNNING flag	Indicates that the **slattach** command was successful. In actuality, the link is accessed, a dial is completed, the other end has answered, and the remote end has returned CARRIER DETECT status. When the CD status occurs the flags are updated with the running bit.

pdisable and lsdev commands:

Any tty port that is used for **SLIP** connections must be in a disabled or unavailable state.

To verify that the port for tty1 is disabled, obtain root user authority and enter one of the following commands:

- `lsattr -El tty1  -a login`

This command displays the permanent state of the tty port as recorded in the system's Object Database Manager (ODM). If the output is anything other than `login disable`, use SMIT to change the enable LOGIN field to **disable**.

- `pdisable | grep tty1`

  This command, when used without parameters, displays all tty ports that are in a disabled state. In this example, **pdisable** is piped to the **grep** command to eliminate unnecessary output. If tty1 is not displayed after running this command, the port is not disabled.

ps command:

The **ps** command displays information about active processes to standard output.

Use this command to verify the existence (or nonexistence) of **slattach** processes that are used to assign a tty line to network interfaces.

If **netstat -in** shows that the interface is down, the user should run the **ps -ef | grep slat** command to see if an **slattach** process is currently running on the associated tty port. Note that for a directly connected **SLIP** interface, broken connections are retried automatically without manual intervention. For a **SLIP** interface connected by modem, broken connections must be manually redialed. If a user supplies a dial string in the **slattach** command line, the user must reenter the command and dial string to restore a broken connection.

ping command and modem lights:

The **ping** command and modem lights are used to debug **SLIP** communication problems.

A ping is an echo request packet, sent out of the machine, and an echo response packet is returned. This sequence of events is useful if the administrator can see the modem lights.

For example, a local system constructs the echo request packet and sends it to the remote system. The Send Data (SD) light on the local modem illuminates. This means that the local TCP/IP, **slattach**, and tty were able to group information and send it out of the modem to the remote system.

The remote modem receives the packet and the receive data light flashes but its SD light does not. This means that the remote system was not able to send (or return) the local system's ping request. As a result, the user on the local system may see the **ping** command hang, requiring a Ctrl-C to exit the condition.

The most common cause of this problem is the use of XON/XOFF flow control in one or both modems, however, the user should not overlook the possibility of routing or address conflicts on the systems.

Common SLIP problems and error messages:

Common **SLIP** problems and error messages, their possible causes, and suggested user actions can be referenced here.

Message: 0821-296 Cannot set line discipline for `/dev/tty#` to slip.ioctl(TXSETLD). A system call received a parameter that is not valid.

Possible Causes: This type of error normally occurs when starting the **slattach** process and is attributable to incorrect configuration of SLIP. The problem is most likely caused by a mismatch between the tty device number and the sl interface number. This also explains why the system reported that ifconfig had not been run before **slattach**.

This problem may also occur when **slattach** processes are dropped or killed incorrectly or when the user attempts to move a **SLIP** connection to another tty port and forgets to reconfigure the sl# interface to match the tty. Check for running **slattach** processes that may still be running (for example, `ps -ef | grep slat`).

Action: The tty device for SLIP is **/dev/tty24** and user has created an sl0 interface. This is incorrect. The user should create an sl24 interface which matches the tty number (tty24 and sl24). If the problem continues, the user should bring down the sl interface (see "Bringing Down an SLIP Interface") and reconfigure the connection using the following commands:

```
lsdev  -Cc if  -s SL
lsattr -El sl0
```

Message:

network is not currently available

route to remote host not available

Possible Cause: These errors occur most often when a user attempts to ping a host over the **SLIP** link and the link has been improperly established. The most likely problem is that one or both tty ports associated with the sl# interface are in an enabled state. It is also possible that there is an address or route conflict between the host systems.

Actions:

- Remove the sl# interface using the `smit rminet` fast path This must be done on both the local and remote SLIP hosts.
- Do the following for each SLIP host:
  1. Enter `pdisable | grep tty#`.
  2. If the tty device is NOT listed in the output of the previous command, the tty is not disabled. Disable the tty either through SMIT or the command line. With tty ports disabled, use SMIT to recreate the **SLIP** interfaces on both systems. If problem persists, verify network addresses and routes (if any). Use the **netstat -ir** command to quickly view address, routing, and interface information.

Problem: When the remote site dials in to the local host, the modem on the local host connects but does not complete the login process.

Possible Causes: If the two modems connect and begin to handshake or exchange connection information but then disconnect, the problem may be due to modem result codes. This problem can also be caused by an improper **slattach** dial string. If the two modems ring but never begin the handshake process, the problem may be that the modem is not set for auto-answer.

Actions:

1. Test the modem connection first with the **cu** command. The modem on the remote host should allow the user to login to the system. There should not be any garbage on the screen during the login attempt; if so, it may indicate a noisy phone line which may be part of the problem. During the login, multiple login heralds should *not* scroll across the screen. If they are present, this could again indicate a problem phone line or incorrect modem settings.
2. Check the modem configurations and try turning off the ARQ codes if they are currently on. In most Hayes-compatible modems this is the &A0 setting. Disabling ARQ result codes does not affect error-controlled connections nor does it keep the modem from returning standard CONNECT messages (if result codes are enabled) as needed for the **slattach** dial string.

Problem: The user is unable to **ping** across a modem **SLIP** connection. The **ping** command may hang or return error messages.

Possible Causes:

1. The modems and/or tty ports may be configured to use XON/XOFF flow control.
2. The **slattach** process may have been terminated on the remote host or the modem connection dropped.
3. The addresses assigned to the **SLIP** hosts may be incorrect.

Actions:

1. Examine both the local and remote modem configurations. They should be set to use RTS/CTS (hardware) flow control or no flow control at all. The user should attempt to ping from each system. Ping systemA to systemB.
2. Verify that the **slattach** process is still running on both local and remote systems. Use the command: `ps -ef |grep slat`. Verify that the sl# interface is in a running state. Use the command: `ifconfig sl#`.
3. Verify that there is not a conflict between the **SLIP** addresses and those associated with other network interface (if any). Use the command: **netstat -ir**. If the address or address class is in question, reconfigure **SLIP** using a simpler address scheme such as 1.1.1.1 for the local host and 1.1.1.2 for the remote host.

SLIP Questionnaire

Use this questionnaire to record data on **SLIP** configurations.

Information collected on these sheets can be faxed to a service representative when additional assistance with **SLIP** configuration is required.

1. Was this **SLIP** configuration working previously? (Y/N) ____
2. What are the machine types? (example: UNIX/PC, DOS/PC, etc.)

   Local System: _____ Remote System: _____

   If the host is not an IBM UNIX system, please name the type of software being used to establish the **SLIP** connection.

   _____
3. What versions of the IBM UNIX operating system are on each of the system units? Issue the `/bin/oslevel` command. If this command is not recognized use the following method:

   ```
   lslpp -h bos.rte
   ```

   look for the *active commit* line release level.

   Local System: _____ Remote System: _____
4. List all interfaces available on both systems (for example, sl0, sl1). To do this, use the command: `lsdev -Cc if`

   Local System: _____ Remote System: _____

   _____ _____

   _____ _____

   _____ _____

   The **SLIP** interface number should match the tty device number. For example, `/dev/tty53` should be used with sl53.
5. Is **SLIP** being configured through SMIT or with commands? SLIP configurations using commands are not permanent and are not present after a system reboot.

   _____
6. Is **SLIP** being configured over modems or a direct serial line?

   _____
7. If modems are being used, list the manufacturer and modem type for both the local and remote systems.

```
        TYPE      BAUD RATE  IBM CABLING If not IBM cable,
                             (Yes/No)    what type?
Local:  _________ _________    ___      _________________________
Remote: _________ _________    ___      _________________________
```

8. If modems are in use, what is the phone carrier type? (leased-line or normal switched)

   _____

9. What hardware is the **SLIP** line being used on?

   128-Port Adapter (with 16-port RAN(s)): ___

   2-Port Adapter: ___

   8-Port Adapter: ___

   Native, S1 or S2 serial ports: ___

10. Is it possible to ping from the local system to the remote system?

    (Y/N) _____ (on the local system enter: ping <remote address>)

11. Is it possible to ping from the remote system to the local system?

    (Y/N) _____ (on the remote system enter: ping <local address>)

12. Are the tty ports disabled on both the local and remote systems?

    (Y/N) _____

    Use the command: `pdisable | grep tty#`. Only disabled tty numbers are displayed as output of this command.

13. Are any error messages being displayed? If yes, please list them below:

    _____

    _____

    _____

    _____

    _____

    _____

    _____

    _____

    _____

    _____

Asynchronous Terminal Emulation

The Asynchronous Terminal Emulation (ATE) program enables terminals on the operating system to emulate a terminal, thus allowing users a means of connecting to most other systems that support asynchronous terminals.

ATE accomplishes this by making the remote system see a terminal either as a system's display or as a DEC VT100 terminal. The VT100 option allows the user to log in to systems that do not support their terminal, but do support VT100 terminals.

ATE uses both direct (cabled) and modem connections to communicate between the user's system and a remote system as shown in the following illustration.

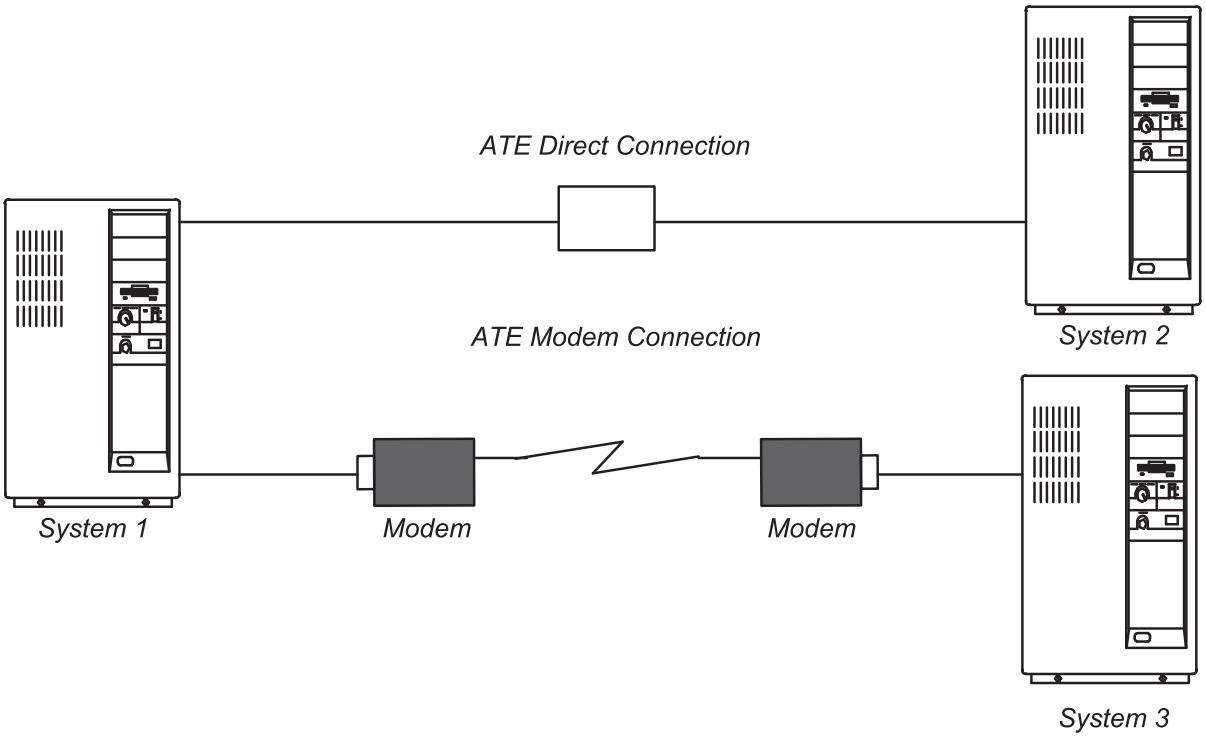


Figure 41. ATE connection types

Depending upon the connection type used, the user can configure ATE to connect either to a system in the next room or to a system across the country. For a direct connection, the user must know the port to use on their system. For a modem connection, users must know the port to use on his system and the telephone number of the remote system. Users must also have a login ID and password on the remote system.

ATE enables a user to run commands on the remote system, send and receive files, and use the **xmodem** protocol to check the data integrity in the files transferred between systems. The user can also capture and file incoming data from the remote system.

> **Note:** You must be a member of the UNIX-to-UNIX Copy Program (UUCP) group in order to use ATE. A user with root authority uses System Management Interface Tool (SMIT) to install individual users in groups.

Setting up ATE

Before running ATE, the system administrator must install the proper software (if needed) and configure the tty ports and connections.

- ATE is an optional program product. All files necessary for operation of ATE are contained in the **bos.net.ate** program product available on the install media. Use the following commands to verify that ATE is available on your system:

  ```
  lslpp -h | more   <return>
  /bos.net.ate      <return>
  ```

  If ATE is not available on your system, install the **bos.net.ate** image from the installation media (tape, diskette, or network server).
- If ATE is installed on the system, a list of files associated with this program can be displayed using the following commands:

  ```
  lslpp -f | more   <return>
  /bos.net.ate      <return>
  ```

- The user must have root user authority to set up the port for the communications device.

ATE uses both direct (cabled) connections and modem connections. Local RS-232C connections allow a maximum distance of 15 meters (50 feet) between machines, and RS-422A connections allow up to 1200 meters (4000 feet) between machines.

Before using ATE to call a remote system, verify that the remote system's tty device is ready to accept a call.

To prepare ATE to run on the system, perform the following steps:

1. Install an asynchronous adapter card in an appropriate slot in the system unit, unless the system has a built-in serial port.
2. Plug the RS-232C or RS-422A cable into the adapter card or the built-in serial port.
3. Add a tty device for the communications port using the `smit mkdev` fast path.
4. Select the terminal type to emulate with ATE and make the necessary adjustments for the environment. The most common changes are line speed, parity settings, number of bits per character, and whether the line is to be driven as a remote or local line. Use bpc 8 and no parity if National Language Support (NLS) is required.
5. Set up the port for the device. To set up a port to call out with ATE, use the **pdisable** command. For example, to set up port tty1, enter:

   ```
   pdisable tty1
   ```

   To set up a port so that others can call in, use the **penable** command. For example, to let other systems call in to the tty2 port, enter:

   ```
   penable tty2
   ```
6. Ensure the device has previously been defined to the remote system. After the device is defined, the ATE program must be customized to reflect the device settings on the remote system. Customize the default settings with the alter and modify subcommands or by editing the `ate.def` default file. To change the default settings for a telephone connection, use a dialing directory file entry.

ATE main menus

ATE displays menus according to the subcommands used.

Starting ATE with the **ate** command displays the Unconnected Main Menu, which lets you:

- Temporarily change characteristics of ATE (**modify**, **alter**)
- Connect to another system (**directory**, **connect**)
- Get help (**help**)
- Execute workstation operating system commands on the system (**perform**)
- Leave ATE (**quit**)

Depending on the subcommand issued from the Unconnected Main Menu, ATE displays various submenus:

Table 107. ATE submenus

When you use	ATE displays
modify subcommand	Modify Menu (for information, see the **ate** command in *Commands Reference, Volume 1*)
alter subcommand	Alter Menu (for information, see the **ate** command in *Commands Reference, Volume 1*)
connect or **directory** subcommand to connect to a remote system	Connected Main Menu
directory subcommand	dialing directory (a list of phone numbers)

From the Connected Main Menu, you can issue subcommands to:

- Send files to and receive files from the remote system (**send**, **receive**)
- Send a break signal to the remote system (**break**)
- End the connection to the remote system (**terminate**)

In addition, the **modify**, **alter**, **help**, **perform**, and **quit** subcommands perform the same functions as those provided from the Unconnected Main Menu.

You can control certain actions of ATE with control key sequences. These key sequences are known as the CAPTURE_KEY, the MAINMENU_KEY, and the PREVIOUS_KEY. The key sequences are discussed in "ATE control key sequences" on page 597. ATE is installed with default key combinations for these keys, but you can change the key combinations by modifying the ATE default file, `ate.def`.

ATE Unconnected Main Menu:

Use the **ate** command to display the ATE Unconnected Main Menu.

After a connection is established, use the ATE **connect** subcommand to display the Unconnected Main Menu.

You can issue the following subcommands from the ATE Unconnected Main Menu. To issue the subcommand, type the first letter of the subcommand at the command prompt on the menu. For example, type **d** to issue the **directory** subcommand.

Item	Description
alter	Temporarily changes data transmission characteristics, such as the speed of transmission.
connect	Makes a connection.
directory	Displays a dialing directory.
help	Displays help information.
modify	Temporarily modifies local settings, such as the capture file, for incoming data.
perform	Allows you to perform workstation operating system commands within ATE.
quit	Quits the ATE program.

Note: Of the control key sequences CAPTURE_KEY, MAINMENU_KEY, and PREVIOUS_KEY, only PREVIOUS_KEY can be used from the ATE Unconnected Main Menu.

ATE Connected Main Menu:

Use the **connect** subcommand from the ATE Unconnected Main Menu to display the Connected Main Menu.

Alternately, press the MAINMENU_KEY while connected to a remote system.

You can issue the following subcommands from the ATE Connected Main Menu. For the definitions of these subcommands, refer to the **ate** command in *Commands Reference, Volume 1*. To issue the subcommand, type the first letter of the subcommand at the command prompt on the menu. For example, type `a` to issue the **alter** subcommand.

Item	Description
alter	Temporarily changes data transmission characteristics, such as the speed of transmission.
break	Sends a break signal to the remote system.
help	Displays help information.
modify	Temporarily modifies local settings used by the emulator, such as the capture file for incoming data.
perform	Allows you to perform workstation operating system commands within ATE.
quit	Quits the ATE program.
receive	Receives files from a remote system.
send	Sends files to a remote system.
terminate	Terminates the ATE connection.

All three ATE control key sequences can be used from the ATE Connected Main Menu.

ATE control key sequences

Use the following control keys with ATE. Change the key sequence for each function by editing the `ate.def` file.

Item	Description
CAPTURE_KEY	Starts or stops saving data displayed on the screen during a connection. The default key sequence for the CAPTURE_KEY is Ctrl-B. The CAPTURE_KEY has a switch or toggle effect. Pressing this control key starts saving data. Pressing this control key a second time stops saving data. Data is saved in the capture file defined in the `ate.def` file. The default capture file name is the `$HOME/kapture` file. Use the **modify** subcommand to temporarily change the capture file name. Edit the ATE default file to permanently change the name of the capture file. See "Editing the ATE default file" on page 605. The CAPTURE_KEY key sequence does not function while the terminal is performing a file transfer operation, and it is valid only when a connection is established. If you press the CAPTURE_KEY key sequence before a connection is established, the next command entered is unsuccessful, and an error message is displayed.
PREVIOUS_KEY	Returns to the previously displayed screen. The PREVIOUS_KEY is also used to stop a file transfer operation. The default key sequence for the PREVIOUS_KEY is Ctrl-R. The PREVIOUS_KEY can be used from either ATE Main Menu.
MAINMENU_KEY	Displays the Connected Main Menu so you can issue an ATE subcommand. The default key sequence for the MAINMENU_KEY is Ctrl-V. Use this control key to display the Connected Main Menu after a connection to a remote system is established. If you press the MAINMENU_KEY key sequence before a connection is established, the next command entered is unsuccessful, and an error message is displayed. By customizing the ATE default file, you can permanently change the control key settings and the capture file name. See "Editing the ATE default file" on page 605.

ATE customization

ATE creates the `ate.def` default file in the current directory the first time the user runs ATE. Edit the `ate.def` file to customize various aspects of ATE.

For example, the user can change the name of the dialing directory file, the type of transfer protocols used to send and receive files from the remote system, and the baud rate ATE expects the modem to use. Refer to "Editing the ATE default file" on page 605 for more information on the `ate.def` file.

Users can also make temporary changes to certain aspects of ATE with the **modify** and **alter** subcommands. These subcommands can change all of the ATE default values except the control key sequences (which can only be changed by editing the default file) and the name of the dialing directory (which can be changed with the **directory** subcommand or by editing the default file). Any changes made

with the **modify**, **alter**, or **directory** subcommands are effective only for that session of ATE. The next time the user runs ATE, the settings used are those defined in the default file.

When using a modem with ATE, the user can create a dialing directory of up to 20 phone numbers. The **directory** subcommand displays the telephone numbers in menu form and allows the user to select the desired system to call. Refer to "Setting up an ATE dialing directory" on page 601 for more information.

By using a dialing directory, the user avoids having to look up the telephone number when calling a particular system. The user can also specify certain data transmission characteristics in the dialing directory file. This is useful if some connections use characteristics that differ from the ATE defaults.

You can create a personalized dialing directory, and the system administrator can create a system-wide dialing directory. Specify which dialing directory to use in the ATE default file. See "Setting up an ATE dialing directory" on page 601 for more information.

ate.def configuration file:

The `ate.def` file sets the defaults for use in asynchronous connections and file transfers.

This file is created in the current directory during the first run of ATE. The `ate.def` file contains the default values in the ATE program uses for the following:

- Data transmission characteristics
- Local system features
- Dialing directory file
- Control keys.

The first time the ATE program is run from a particular directory, it creates an `ate.def` file in that directory.

```
LENGTH          8
STOP            1
PARITY          0
RATE            1200
DEVICE          tty0
INITIAL         ATDT
FINAL
WAIT            0
ATTEMPTS        0
TRANSFER        p
CHARACTER       0
NAME            kapture
LINEFEEDS       0
ECHO            0
VT100           0
WRITE           0
XON/XOFF        1
DIRECTORY       /usr/lib/dir
CAPTURE_KEY     002
MAINMENU_KEY    026
PREVIOUS_KEY    022
```

Edit the `ate.def` file with any ASCII text editor to permanently change the values of these characteristics. Temporarily change the values of these characteristics with the ATE **alter** and **modify** subcommands, accessible from the ATE Main Menu.

Type parameter names in uppercase letters in the `ate.def` file. Spell the parameters exactly as they appear in the original default file. Define only one parameter per line. An incorrectly defined value for a parameter causes ATE to return a system message. However, the program continues to run using the default value. These are the `ate.def` file parameters:

LENGTH
Specifies the number of bits in a data character. This length must match the length expected by the remote system.

```
Options:  7 or 8
Default:  8
```

STOP Specifies the number of stop bits appended to a character to signal that character's end during data transmission. This number must match the number of stop bits used by the remote system.

```
Options:  1 or 2
Default:  1
```

PARITY
Checks whether a character is successfully transmitted to or from a remote system. Must match the parity of the remote system.

For example, if the user selects even parity, when the number of 1 bits in the character is odd, the parity bit is turned on to make an even number of 1 bits.

```
Options: 0 (none), 1 (odd), or 2 (even)
Default: 0.
```

RATE Determines the baud rate, or the number of bits transmitted per second (bps). The speed must match the speed of the modem and that of the remote system.

```
Options: 50,75,110,134,150,300,600,1200,1800,2400,4800,9600,19200
Default: 1200
```

DEVICE
Specifies the name of the asynchronous port used to make a connection to a remote system.

```
Options: Locally created port names.
Default: tty0.
```

INITIAL
Defines the dial prefix, a string that must precede the telephone number when the user autodials with a modem. For the proper dial commands, consult the modem documentation.

```
Options: ATDT, ATDP, or others, depending on the type of modem.
Default: ATDT.
```

FINAL
Defines the dial suffix, a string that must follow the telephone number when the user autodials with a modem. For the proper dial commands, consult the modem documentation.

```
Options: Blank (none) or a valid modem suffix.
Default: No default.
```

WAIT Specifies the time to wait between redialing attempts. The wait period does not begin until the connection attempt times out or until it is interrupted. If the ATTEMPTS parameter is set to 0, no redial attempt occurs.

```
Options: 0 (none) or a positive integer designating the number of seconds to wait.
Default: 0
```

ATTEMPTS
Specifies the maximum number of times the ATE program tries redial to make a connection. If the ATTEMPTS parameter is set to 0, no redial attempt occurs.

```
Options: 0 (none) or a positive integer designating the number of attempts.
Default: 0
```

TRANSFER
Defines the type of asynchronous protocol that transfers files during a connection.

p (pacing)
File transfer protocol controls the data transmission rate by waiting for a specified

character or for a certain number of seconds between line transmissions. This helps prevent loss of data when the transmission blocks are either too large or sent too quickly for the system to process.

x (xmodem)
An 8-bit file transfer protocol to detect data transmission errors and retransmit the data.

```
Options: p (pacing), x (xmodem)
Default: p.
```

CHARACTER
Specifies the type of pacing protocol to be used. Signal to transmit a line. Select one character.

When the **send** subcommand encounters a line-feed character while transmitting data, the subcommand waits to receive the pacing character before sending the next line.

When the **receive** subcommand is ready to receive data, it sends the pacing character, then waits 30 seconds to receive data. The **receive** subcommand sends a pacing character again whenever it finds a carriage return character in the data. The **receive** subcommand ends when it receives no data for 30 seconds.

```
Options: any character
Default: 0
```

Interval
Number of seconds the system waits between each line it transmits. The value of the Interval variable must be an integer. The default value is 0, indicating a pacing delay of 0 seconds.

```
Default: 0.
```

NAME
File name for incoming data (capture file).

```
Options: A valid file name less than 40 characters long.
Default: kapture
```

LINEFEEDS
Adds a line-feed character after every carriage-return character in the incoming data stream.

```
Options: 1 (on) or 0 (off).
Default: 0.
```

ECHO Displays the user's typed input. For a remote computer that supports echoing, each character sent returns and displays on the screen. When the ECHO parameter is on, each character is displayed twice: first when it is entered, and again when it returns over a connection. When the ECHO parameter is off, each character displays only when it returns over the connection.

```
Options: 1 (on) or 0 (off).
Default: 0.
```

VT100 The local console emulates a DEC VT100 terminal so DEC VT100 code can be used with the remote system. With the VT100 parameter off, the local console functions like a workstation.

```
Options: 1 (on) or 0 (off).
Default: 0.
```

WRITE
Captures incoming data and routes it to the file specified in the NAME parameter as well as to the display. Carriage-return or line-feed combinations are converted to line-feed characters before they are written to the capture file. In an existing file, data is appended to the end of the file.

The CAPTURE_KEY (usually the Ctrl-B key sequence) can be used to toggle capture mode on or off during a connection.

```
Options: 1 (on) or 0 (off).
Default: 0.
```

XON/XOFF

Controls data transmission at a port as follows:

- When an XOFF signal is received, transmission stops.
- When an XON signal is received, transmission resumes.
- An XOFF signal is sent when the receive buffer is nearly full.
- An XON signal is sent when the buffer is no longer full.

```
Options: 1 (On), or 0 (Off).
Default: 1.
```

DIRECTORY

Names the file that contains the user's dialing directory.

```
Default: the /usr/lib/dir file.
```

CAPTURE_KEY

Defines the control key sequence that toggles capture mode. When pressed, the CAPTURE_KEY (usually the Ctrl-B key sequence) starts or stops capturing (saving) the data that is displayed on the screen during an active connection.

```
Options: Any ASCII control character.
Default: ASCII octal 002 (STX).
```

MAINMENU_KEY

Defines the control key sequence that returns the Connected Main Menu so the user can issue a command during an active connection. The MAINMENU_KEY (usually the Ctrl-V key sequence) functions only from the connected state.

```
Options: Any ASCII control character.
Default: ASCII octal 026 (SYN).
```

PREVIOUS_KEY

Defines the control key sequence that displays the previous screen anytime during the program. The screen displayed varies, depending on the screen in use when the user presses PREVIOUS_KEY (usually the Ctrl-R key sequence).

```
Options: Any ASCII control character.
Default: ASCII octal 022 (DC2).  The ASCII control character is mapped to the interrupt signal.
```

Setting up an ATE dialing directory

The ATE dialing directory file lists phone numbers that the ATE program uses to establish remote connections by modem.

To set up an ATE dialing directory, the following prerequisites must be met:

- The Asynchronous Terminal Emulation (ATE) program must be set up on the system
- To set up a system-wide dialing directory, the user must have write access to the `/usr/lib/dir` file

Users name the dialing directory file with any valid file name and place it in any directory where read and write access is owned. Edit the dialing directory file with any ASCII text editor. The default dialing directory information for the ATE program is contained in the `/usr/lib/dir` file, as shown in the following:

Note: In the following content, some ATE entries have been broken into separate lines for readability. In an actual dialing directory file, however, all elements of an entry must be declared on a single, continuous line.

```
#   COMPONENT_NAME: BOS dir
#
#   FUNCTIONS:
#
#   ORIGINS: 27
#
#
```

```
#   (C) COPYRIGHT International Business Machines Corp. 1985, 1989
#   Licensed Materials - Property of IBM
#
#   US Government Users Restricted Rights - Use, duplication or
#   disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
#
#   dir - sample dialing directory
#
#
# Micom   9,555-9400 1200 7 1 2 0 0
# R20     9,555-9491 1200 7 1 2 0 0
# QT      9,555-8455 1200 7 1 2 0 0
# Dallas1 9,555-7051 1200 8 1 0 0 0
```

Users can access the dialing directory information from within ATE by using the **directory** subcommand available in the **UNCONNECTED MAIN MENU**. The screen will show the directory information as it would appear from within the ATE program.

Users can have more than one dialing directory. To change the dialing directory file the ATE program uses, the user must modify the `ate.def` file in the current directory.

> **Note:** The dialing directory file can contain up to 20 lines (one entry per line). ATE ignores subsequent lines.

The dialing directory file is similar to a page in a telephone book that contains entries for the remote systems called with the ATE program. The format of a dialing directory entry is:

```
Name Phone Rate Length StopBit Parity Echo Linefeed
```

The fields must be separated by at least one space. More spaces can be used to make each entry easier to read. The fields are:

Name Identifies a telephone number. The name can be any combination of 20 or fewer characters. Use the _ (underscore) instead of a blank between words in a name, for example, data_bank.

Phone The telephone number to be dialed. The number can be up to 40 characters. Consult the modem documentation for a list of acceptable digits and characters. For example, if a 9 must be dialed to access an outside line, include a 9, (the numeral 9 and a comma) before the telephone number as follows: 9,1112222.

Although the telephone number can be up to 40 characters long, the directory subcommand displays only the first 26 characters.

Rate Transmission or baud rate in bits per second (bps). Determines the number of characters transmitted per second. Select a baud rate that is compatible with the communication line being used. The following are acceptable rates:

50, 75, 110, 134, 150, 300, 600, 1200, 1800, 2400, 4800, 9600, or 19200.

For non-POSIX baud rates, setting the rate at 50 causes the ATE to use the configured baud rate set through SMIT for that device.

Length Number of bits that make up a character. The entry for the Length field can be 7 or 8.

StopBit
Stop bits signal the end of a character. The entry for the StopBit field can be 1 or 2.

Parity Checks whether a character was successfully transmitted to or from a remote system. The entry for the Parity field can be 0 (none), 1 (odd), or 2 (even).

Echo Determines whether typed characters display locally. The entry for the Echo field can be 0 (off) or 1 (on).

Linefeed
Adds a line-feed character at the end of each line of data coming in from a remote system. The line-feed character is similar in function to the carriage-return and new-line characters. The entry for the Linefeed field can be 0 (off) or 1 (on).

Note: Changing or remapping may be necessary if control keys conflict across applications. For example, if the control keys mapped for the ATE program conflict with those in a text editor, remap the ATE control keys.

Note: The ASCII control character selected may be in octal, decimal, or hexadecimal format, as follows:

octal 000 through 037. The leading zero is required.

decimal
0 through 31.

hexadecimal
0x00 through 0x1F. The leading 0x is required. The x may be uppercase or lowercase.

Create an `ate.def` file that defines those characteristics to change characteristics of ATE emulation. For example, to change the RATE to 300 bps, the DEVICE to tty3, the TRANSFER mode to x (xmodem protocol), and the DIRECTORY to my.dir, create an `ate.def` with the following entries, in the directory running the ATE program:

```
RATE          300
DEVICE        tty3
TRANSFER      x
DIRECTORY     my.dir
```

The program uses the defined values from time the ATE program starts from that directory.

1. Create the dialing directory file:
   a. Change to the directory where the dialing directory file will reside.
   b. Copy the `/usr/lib/dir` file to use as a template. Rename the file to any valid file name.
   c. Create telephone number entries using the format given in the dialing directory file format.
   d. Save the file.

   Note: If the new dialing directory file is to be the system-wide default file, save the file with the name `/usr/lib/dir`.
2. If the dialing directory file name is not the default (`/usr/lib/dir`), edit the `ate.def` file in the directory from which the ATE program is run. Change the *DIRECTORY* parameter in the `ate.def` file to the new dialing directory file. See "Editing the ATE default file" on page 605
3. Start ATE, and view the dialing directory with the **directory** subcommand.

Dialing-out with ATE

Use this procedure to dial out of a system using ATE and a customized `/usr/lib/dir` dialing directory file.

Verify that all of the following prerequisites and conditions are met before attempting to dial out.

- The ATE is installed on the system.
- A modem is attached, configured, and ready for use.
- The user is a member of the UUCP group (see "Setting up ATE" on page 594 for more information).
- The `/usr/lib/dir` dialing directory file is already customized with the correct information.
- User's present working directory (`pwd`) contains an `ate.def` file that is properly updated.
- The `/dev/tty` port must have its ENABLE login field in SMIT set to disable, share, or delay.

1. Enter:

```
ate
```

2. At the main menu, type `d` and press Enter.
3. Type the file name of the directory you want to display and press Enter. To use the current directory, just press Enter.
4. Enter the appropriate directory entry number under the **#** column to dial the corresponding phone number.

Transferring a file using ATE

Use this procedure to transfer a file from a local host to the remote system.

Verify that all of the following prerequisites and conditions are met before attempting to transfer a file using ATE:

- A connection must already be established using the **ATE** program.
- The Xmodem file transfer protocol must already exist on both the local and remote systems. On the operating system, Xmodem is located in the `/usr/bin` directory.

1. Run the following **xmodem** command on the remote system after logging in:

   ```
   xmodem -r newfile
   ```

   where **r** is the Xmodem flag to receive and *newfile* is the name of the file to be received. This name does not need to be the same as the file being transferred.
2. Press Enter.
3. The following message is displayed:

   ```
   ate: 0828-005 The system is ready to receive file newfile. Use Ctrl-X to stop xmodem.
   ```

   If the message is not displayed, the system might not have the **xmodem** program installed or located in its command PATH.
4. Press Ctrl-V to return to the ATE CONNECTED MAIN MENU.
5. Press the S key to send a file.
6. The following message is displayed:

   ```
   Type the name of the file you wish to send and press Enter. To use the last
   file name (), just press Enter.
   ```
7. Enter the name and full path of the file to be transferred.
8. Press Enter.
9. ATE will display the following message and begin to transfer the file:

   ```
   ate: 0828-024 The program is ready to send file newfile. You will receive another
   message when the file transfer is complete.
   ate: 0828-025 The system is sending block 1.
   ate: 0828-025 The system is sending block 2.
   ate: 0828-015 The file transfer is complete.
   ate: 0828-040 Press Enter
   ```
10. Press Enter when the transfer is complete.

Receiving a file using ATE

Use this procedure to receive a file transferred from a remote host.

Verify that all of the following prerequisites and conditions are met before attempting to receive a file using ATE:

- A connection must already be established using the ATE program.
- The Xmodem file transfer protocol must already exist on both the local and remote systems. On the operating system, Xmodem is located in the `/usr/bin` directory.

1. Run the following **xmodem** command on the remote system after logging in:

   ```
   xmodem -s newfile
   ```

where **s** is the **xmodem** command to send and *newfile* is the name and full path of the file to be transferred.

2. Press Enter.
3. The following message is displayed:

   ```
   ate: 0828-005 The system is ready to send file newfile. Use ctrl-X to stop xmodem.
   ```

   If the message is not displayed, the system may not have the **xmodem** program installed or located in its command PATH.
4. Press Ctrl-V to return to the ATE CONNECTED MAIN MENU.
5. Press the R key to receive the file.
6. The following message is displayed:

   ```
   Type the name of the file you wish to store the received data in and press
   Enter.  To use the last file name (), just press Enter.
   ```
7. Enter the name and full path of the file to be transferred.
8. Press Enter.
9. ATE will display the following message and begin to transfer the file:

   ```
   ate: 0828-020 The program is ready to receive file newfile. You will
   receive another message when the file transfer is complete.
   ate: 0828-028 The system is receiving block 1.
   ate: 0828-028 The system is receiving block 2.
   ate: 0828-040 Press Enter.
   ```
10. Press Enter when the transfer is complete.

Editing the ATE default file

To edit the ATE default file, the ATE program must be set up on the system.

To change the settings in the `ate.def` file:

1. Open the `ate.def` file with an ASCII text editor.
2. Enter new values for the parameters to be changed. Other values can be deleted or left alone. The system uses its defaults for any parameters that are deleted.
3. Save the modified `ate.def` file.

The changes made to the `ate.def` file take effect the next time ATE is run from the directory containing the customized `ate.def` file.

You can keep a copy of the `ate.def` file in any directory to which you have read and write permissions. For example, if you need to run the ATE program with different defaults at different times, keep multiple copies of the `ate.def` file, with the appropriate settings, in different subdirectories of the `$HOME` directory. However, multiple copies of the `ate.def` file use system storage. As an alternative, temporarily change most settings with the ATE **alter** and **modify** subcommands. Use a dialing directory entry to change settings for an individual modem connection. See “Setting up an ATE dialing directory” on page 601.

ATE troubleshooting

When encountering the following common ATE problems, consider these solutions.

Problem:

When transferring or receiving files, the **xmodem** command appears to hang. A Ctrl-X corrects the problem.

Solution:

Examine the Alter menu to verify that xmodem protocol (or Transfer method) is being used.

Problem:

When transferring or receiving files, the file scrolls across the screen and a message is displayed stating that the transfer or receipt was complete when, in fact, it was not.

Solution:

Examine the Alter menu to verify that **xmodem** protocol (or Transfer method) is being used.

Problem:

When starting ATE, the user receives the following error:

```
ate: 0828-008 The system tried to open port /dev/tty0 but failed. If the port name is not correct,
change it using the Alter menu. Or, take the action indicated by the system message shown below.

Connect: The file access permissions do not allow the specified action.
ate: 0828-040 Press Enter.
```

Solution:

The Connect: line in the error message narrows down the problem. Verify that the user attempting to run ATE is a member of the UUCP group. To check this, the user can enter *id* on the command line; `uucp` should appear in the output listing.

Problem:

When attempting to make a connection with ATE, the following error is received:

```
ate: 0828-008 The system tried to open port /dev/tty0 but failed. If the port name is not
correct, change it using the Alter menu. Or, take the action indicated by the system
message shown below.

Connect: A file or directory in the path name does not exist.
ate: 0828-040 Press Enter.
```

Solution:

An incorrect or unavailable tty was selected for use by ATE. Examine the Alter screen in ATE.

Problem:

The file transfers correctly, but the file size is larger than the original file.

Solution:

The xmodem protocol pads the file during transfer. To avoid this, use the **tar** command to compress the file and transfer it. This is also a means of overcoming another xmodem limitation where only one file is sent at a time. The user can **tar** several files together into a single tar image and transfer it using xmodem.

ATE commands and subcommands

This is a list of the ATE commands and subcommands with a brief description of what they do.

See "ATE file formats" on page 607 for additional reference information.

Item	Description
ate	Starts the ATE program. For the definitions of its subcommands, which follow, refer to the **ate** command:
	break Inputs current activity on a remote system.
	connect Connects to a remote computer.
	directory Displays the ATE dialing directory and lets you choose an entry from the directory to connect to a remote system.
	help Provides help for the ATE subcommands.
	perform Lets you issue workstation operating system commands while using ATE.
	quit Exits the ATE program.
	receive Receives a file from a remote system.
	send Sends a file to a remote file system.
	terminate Terminates an ATE connection to a remote system.

In addition, the **xmodem** command is useful for transferring files with the xmodem protocol, which detects data transmission errors during asynchronous transmission.

ATE file formats

Asynchronous Terminal Emulation (ATE) file formats include the `ate.def` and dialing directory formats.

Item	Description
`ate.def`	Sets default settings for connections.
Dialing directory	Defines telephone numbers and settings for specific modem connections.

See "ATE commands and subcommands" on page 606 for additional reference information.

Dynamic screen utility

The dynamic screen utility, or **dscreen** command, is a utility that allows a single physical terminal to be connected to several virtual terminal sessions (screens) at one time.

It is mainly intended for use with terminals that have two or more pages of screen memory (for example, the IBM 3151 Models 310 or 410 display with the Cartridge for Expansion). With such terminals, switching between virtual screens also switches between physical terminal screen pages allowing each virtual screen's image to be saved and restored. On terminals without multiple pages of screen memory, the **dscreen** command can still be used to switch among virtual screen sessions although the appearance of the screen will not be maintained.

Note: For full support of **dscreen** utility, the terminal must be able to switch internal screen pages on command and must remember the cursor position for each page. While the **dscreen** utility will work on both smart and dumb terminals, screen images are not saved during screen changes on dumb terminals.

dscreen terminal configuration information file

The **dscreen** utility terminal configuration information file (or `dsinfo` file) is used to define a different set of keys to be used with the **dscreen** utility.

This might be done, for example, when the originally defined **dscreen** utility keys conflict with a software application in use on the system.

The `dsinfo` file terminal type assumes a single page of screen memory. Therefore, if a terminal supports additional pages of screen memory, the `dsinfo` file must be customized to use the appropriate sequence for page memory control. Consult the appropriate terminal reference guide for the specific control sequence.

The default `dsinfo` file is `/usr/lbin/tty/dsinfo`. Use the **-i** flag to specify a different `dsinfo` file. This remainder of this section refers to the default file. However, the same information applies to any customized `dsinfo` file you create.

For more information concerning the `dsinfo` file, see "Dynamic screen assignment" on page 609.

dscreen key action assignments

When the **dscreen** command is executed, it starts a virtual screen. Some of the keys on the terminal keyboard are not passed through to the virtual screen; instead, the **dscreen** utility intercepts these keys and performs certain actions when they are pressed.

The actions include:

Item	Description
Select (see “dscreen Select keys”)	Selects a specified screen.
Block (see “dscreen Block keys”)	Blocks all input and output.
New (see “dscreen New keys”)	Starts a new screen session.
End (see “dscreen End and Quit keys”)	Ends the **dscreen** utility.
Quit (see “dscreen End and Quit keys”)	Quits the **dscreen** utility.
Previous (see “dscreen Previous key” on page 609)	Switches to previous screen.
List (see “dscreen List key” on page 609)	Lists the **dscreen** assigned keys and their actions.

The function of each key is dependent on the terminal and the terminal description in the `/usr/lbin/tty/dsinfo` file.

dscreen Select keys:

When a new virtual screen is created, it is assigned a select key.

Pressing the select key causes the following actions:
- A switch from the physical terminal to the video page associated with the particular virtual screen.
- Input and output is directed appropriately between the physical terminal and the virtual screen.

After all of the select keys defined in the `dsinfo` file have virtual screens assigned to them, no more screens can be created. Individual screen sessions end when the original shell process exits. This frees the associated select key for use with another virtual screen. The **dscreen** utility is ended when there are no more active screens.

dscreen Block keys:

Block keys are used to stop output in a fashion similar to Ctrl-S key when using IXON flow control.

The purpose of these keys is to allow for transparently setting up terminal sessions on two computers using a terminal that has two serial ports.

dscreen New keys:

Pressing a new screen key creates a new logical screen and assigns it to one of the select keys.

Each new screen requires:
- A select key as defined in the dsinfo file
- A **dscreen** pseudo-terminal device
- Enough memory for the various structures used in screen tracking
- A process to run the shell from.

If any of these are not available, the new screen operation fails with a message indicating the reason for the failure.

dscreen End and Quit keys:

When the End and Quit keys are pressed, a sequence of actions occurs.

Pressing an End key causes the following to occur:
- Send a **SIGHUP** signal to all the screen sessions
- Clean up
- Exit with a status of 0.

Pressing a Quit key performs the same actions but exits with status of 1.

dscreen Previous key:

Pressing a previous key switches the terminal to the screen that was last displayed.

Note:

1. Do not switch screens when the current screen is being written to; an escape sequence might be truncated and leave the terminal in an unknown state.
2. Some terminal displays can save the cursor position for individual screens but might not save other states such as insert mode, inverse video, and so forth. If this is the case, users should avoid these modes while switching screens.

dscreen List key:

Pressing a list key displays a list of keys and their actions on the terminal display.

Only those keys recognized by the **dscreen** utility will be shown. When a new screen is created using the **dscreen** utility, the message `Press KEY for help`, where `KEY` is the name of the list key displayed on the terminal. Note that the message is displayed *only* if there is a list key defined.

Dynamic screen assignment

The terminal description entry in the `/usr/lbin/tty/dsinfo` file has the same number of screen selection keys as the terminal has physical screen pages. If more screen selection keys are defined than the number of physical screen pages, the **dscreen** utility will dynamically assign physical screen pages to virtual screens.

When a virtual screen is selected that does not have an associated page of screen memory, the **dscreen** utility assigns the least recently used physical screen to the virtual screen. Depending on the specifications maintained in the `/usr/lbin/tty/dsinfo` description file, an indication that the physical screen is connected to a different virtual screen might be noticeable; for example, the screen is cleared.

dsinfo file

The `dsinfo` file is a database of terminal descriptions used by the **dscreen** multiple screen utility.

The file contains the following information:

- The **dscreen** utility keys and the functions they perform
- Number of screen memory pages for the terminal
- Code sequences sent or received to use the above features.

The terminal type entries in the default `dsinfo` file resemble the following 3151 ASCII terminal values:

```
# The Cartridge for Expansion (pn: 64F9314) needed for this entry
ibm3151|3151|IBM 3151,
dsks=\E!a^M|Shift-F1|,            # Selects first screen
dsks=\E!b^M|Shift-F2|,            # Selects second screen
dsks=\E!c^M|Shift-F3|,            # Selects third screen
dsks=\E!d^M|Shift-F4|,            # Selects fourth screen
dskc=\E!e^M|Shift-F5|,            # Creates a new screen
dske=\E!f^M|Shift-F6|\E pA\EH\EJ, # Go to screen 1 and end
dskl=\E!g^M|Shift-F7|,            # Lists function keys (help)
dskp=\E!h^M|Shift-F8|,            # Go to previous screen
dskq=\E!i^M|Shift-F9|\E pA\EH\EJ, # Go to screen 1 and quit
dsp=\E pA|\EH\EJ,                 # Terminal sequence for screen 1
dsp=\E pB|\EH\EJ,                 # Terminal sequence for screen 2
dsp=\E pC|\EH\EJ,                 # Terminal sequence for screen 3
dsp=\E pD|\EH\EJ,                 # Terminal sequence for screen 4
dst=10,                           # Allow 1 second timeout buffer
```

Entry format for dsinfo:

Entries in the `dsinfo` file consist of comma-separated fields.

The first field is a list of alternate names for the terminal, each name is separated by a pipe (|) character. Any text preceded by a pound (#) character is regarded as a comment and ignored by **dscreen**. The remaining fields are strings describing the capabilities of the terminal to the **dscreen** utility. Within these strings, the following escape codes are recognized:

Table 108. dsinfo file fields

Escape Sequence	Description
\E,\e	escape character
\n,\l	newline (or linefeed) character
\r	carriage return
\t	tab character
\b	backspace character
\f	formfeed character
\s	space character
nnn	character with octal value *nnn*
^*x*	Ctrl-X for any appropriate *x* value

Any other character preceded by a backslash will yield the character itself. The strings are entered as *type=string*, where *type* is the type of string as listed below, and *string* is the string value.

It is important that the entry fields in the`dsinfo` file be separated by commas. If a comma is omitted or truncated from the end of a `dsinfo` file entry, the file will become unreadable by the **dscreen** utility and an error will be returned to the display.

disinfo string types:

The disinfo string types can be referenced here.

The string types are as follows:

Item **Description**

dskx A string type that starts with dsk describes a key. The type must be four letters long, and the fourth letter *x* indicates what action is taken when the key is received. The key types are:

Type	Action
dsks	Switch Screens
dskb	Block Input and Output
dske	End **dscreen**
dskq	Quit **dscreen** (exit status=1)
dskc	Create New Screen
dskp	Switch to Previous Screen
dskl	List Keys and Actions

Any other key type (that is, a string type dsk*x* that does not end in s, b, e, q, p, or l) will cause no internal **dscreen** action, but will show up in the key listing and will be recognized and acted on. A type of dskn (n for No Operation) should be used when no internal **dscreen** action is desired.

The value string for each key has three substrings, which are separated by pipe (|) characters.
Note: Use \| to include the | character in one of the substrings.

The first substring is the sequence of characters that the terminal sends when the key is pressed. The second substring is a label for the key that is printed when a list of keys is displayed. The third substring is a sequence of characters that **dscreen** sends to the terminal when this key is pressed before performing the action this key requests.

dsp A string type of dsp describes a physical screen in the terminal. One dsp string should be present for each physical screen in the terminal. The value string for each physical screen has two substrings, which are separated by a pipe (|) character.

The first substring is the sequence of characters to send to the terminal to display and output to the physical page on the terminal.

The second substring is sent to the terminal when the page is used for something new. This second substring is often set to the clear screen sequence. It is sent under the following two conditions:

1. When a new virtual terminal session is being created.
2. When there are more virtual terminals than there are physical screens. If a virtual terminal is selected that requires **dscreen** to reuse one of the physical screens, it will send this sequence to the screen to indicate that the screen contents do not match the output of the virtual terminal connected.

Note: Running with more virtual terminals than physical screens can be confusing and is not recommended; it can be avoided by defining no more screen selection keys (dsks=) than physical screens (dsp=) in the dsinfo entry.

dst A String with a type of dst adjusts **dscreen**'s input timeout. The value of the string is a decimal number. The timeout value is in tenths of seconds and has a maximum value of 255 (default=**1** [or 0.1 seconds]).

When **dscreen** recognizes a prefix of an input key sequence but does not have all the characters of the sequence, it will wait for more characters to be sent until it is recognizable. If the timeout occurs before more characters are received, the characters are sent on to the virtual screen and **dscreen** will not consider these characters as part of an input key sequence.

It may be necessary to raise this value if one or more of the keys **dscreen** is to trigger on is actually a number of keystrokes (that is assigning Ctrl-Z 1, Ctrl-Z 2, Ctrl-Z 3, etc., for screen selection and Ctrl-Z N for new screen and so on).

dysinfo examples:

The following `dysinfo` examples are for Wyse-60 with three screen sessions.

```
wy60|wyse60|wyse model 60,
dsks=^A`^M|Shift-F1|,
dsks=^Aa^M|Shift-F2|,
dsks=^Ab^M|Shift-F3|,
dskc=\200|Ctrl-F1|,
dske=\201|Ctrl-F2|\Ew0\E+,
```

```
dskl=\202|Ctrl-F3|,
dsp=\Ew0|\E+,
dsp=\Ew1|\E+,
dsp=\Ew2|\E+,
```

With this entry:

- Shift-F1 through Shift-F3 are used for selecting screens 1 through 3.
- Ctrl-F1 creates a new screen.
- Ctrl-F2 sends: `Esc w 0 Esc +` to the screen (switching to window 0 and clearing the screen) and then ends **dscreen**.
- Ctrl-F3 lists the keys and their functions.

Each time a physical screen is used for a new screen, the sequence `Esc +` will be sent to the terminal, which will clear the screen.

The following example is for a Wyse-60 with three screen sessions, but one of the screens is on a second computer communicating through the second serial port on the terminal:

```
wy60-1|wyse60-1|wyse model 60 - first serial port
dsks=^A^M|Shift-F1|,
dsks=^Aa^M|Shift-F2|,
dsks=^Ab^M|Shift-F3|\Ed#^Ab\r^T\Ee9,
dskc=\200|Ctrl-F1|,
dske=\201|Ctrl-F2|\Ed#\201^T\Ew0\E+,
dskl=\202|Ctrl-F3|,
dsp=\Ew0|\E+,dsp=\Ew1|\E+,
wy60-2|wyse60-2|wyse model 60 - second serial port
dsks=^A^M|Shift-F1|\Ed#^A^\r^T\Ee8,
dsks=^Aa^M|Shift-F2|\Ed#^Aa\r^T\Ee8,
dsks=^Ab^M|Shift-F3|,
dskc=\200|Ctrl-F1|,
dske=\201|Ctrl-F2|\Ed#\201^T\Ew0\E+,
dskl=\202|Ctrl-F3|,
dsp=\Ew2|\E+,
```

dscreen must be run on both computers, with terminal type wy60-1 on the first computer and terminal type wy60-2 on the second computer (using the **-t** option to **dscreen**). The wy60-1 entry will be examined first.

The first two key entries are unchanged from the original wy60 entry. The third key, however, has type dskb, which means block both input and output. When this key is pressed, the sequence:

```
Esc d # Ctrl-A b CR Ctrl-T Esc e 9
```

is sent to the terminal; after this output is blocked and **dscreen** continues scanning input for key sequences but discards all other input.

The sequence `Esc d #` puts the terminal in transparent print mode, which echoes all characters up to a Ctrl-T out through the other serial port.

The characters `Ctrl-A b CR` are sent out the other serial port, informing the **dscreen** process on the other computer that it should activate the window associated with the Shift-F3 key.

The Ctrl-T key sequence exits the transparent print mode. The Esc 9 key sequence causes the terminal to switch to the other AUX serial port for data communications.

At this point, the other computer takes over, sends an `Esc w 2` to switch to the third physical screen, and then resumes normal communication.

The wy60-2 entry follows the same general pattern for keys Shift-F1 and Shift-F2:

- Switch to transparent print mode
- Send function key string to other computer
- Switch transparent print off
- Switch to the other serial port

The end key, Ctrl-F2, works the same for both computers; it sends the end key sequence to the other computer through the transparent print mechanism, switches the terminal to window 0, clears the screen, then exits.

Generic data link control environment

Generic data link control (GDLC) is a generic interface definition that allows application and kernel users a common set of commands to control data link control (DLC) device managers within the operating system.

For problem determination, see GDLC Problem Determination in *Communications Programming Concepts*.

Generic data link control (GDLC) is a generic interface definition that provides application and kernel users a common set of commands to control DLC device managers within the operating system.

The GDLC interface specifies requirements for entry point definitions, functions provided, and data structures for all DLC device managers. DLCs that conform to the GDLC interface include:

- 8023 (IEEE 802.3 for Ethernet)
- ETHER (Standard Ethernet)
- SDLC (Synchronous Data Link Control)
- TOKEN (Token-Ring)
- FDDI (Fiber Distributed Data Interface)

DLC device managers perform higher-layer protocols and functions beyond the scope of a kernel device driver. However, the managers reside within the kernel for maximum performance and use a kernel device driver for their I/O requests to the adapter. A DLC user is located above or within the kernel.

Synchronous data link control (SDLC) and IEEE 802.2 Data Link Control are examples of DLC device managers. Each DLC device manager operates with a specific device driver or set of device drivers. SDLC, for example, operates with the multiprotocol device driver for the system's product and its associated adapter.

The basic structure of a DLC environment is shown in the "DLC Device Manager Environment" figure. Users within the kernel have access to the communications memory buffers (mbufs) and call the add entry points through the **fp** kernel services. Users above the kernel access the standard interface-to-kernel device drivers, and the file system calls the **dd** entry points. Data transfers require a move of data between user and kernel space.

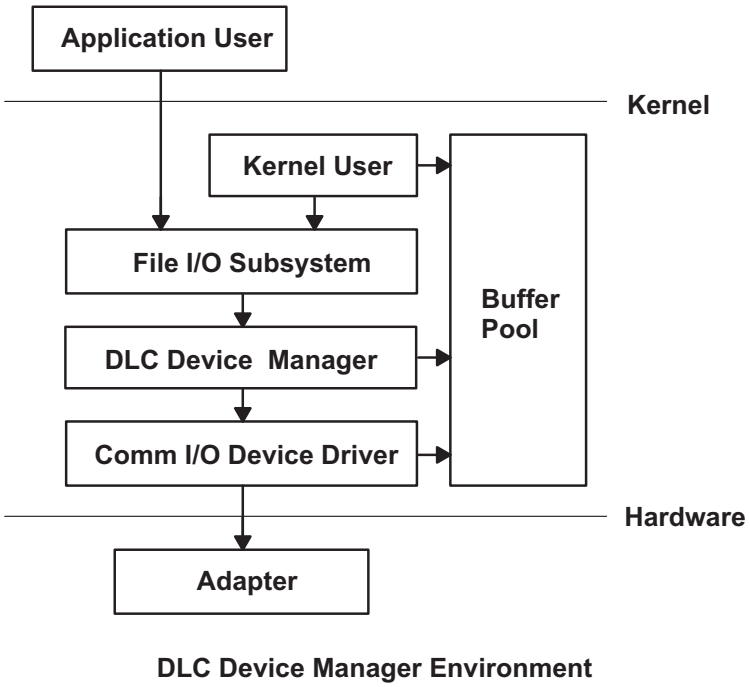


Figure 42. DLC device manager environment

This illustration shows the link between the application user and the adapter (hardware level). The areas in between are Kernel User, File I/O Subsystem, DLC Device Manager, Comm I/O Device Driver, and the Buffer Pool. These "in-between" entities are at the Kernel Level.

The components of the DLC device manager environment are:

Item	Description
Application User	Resides above the kernel as an application or access method.
Kernel User	Resides within the kernel as a kernel process or device manager.
File I/O Subsystem	Converts the file-descriptor and file-pointer subroutines to file pointer accesses of the switch table.
Buffer Pool	Provides data-buffer services for the communications subsystem.
Comm I/O Device Driver	Controls hardware adapter I/O and direct memory access (DMA) registers, and routes receive packets to multiple DLCs.
Adapter	Attaches to the communications media.

A device manager written in accordance with GDLC specifications runs on all the operating system hardware configurations containing a communications device driver and its target adapter. Each device manager supports multiple users above as well as multiple device drivers and adapters below. In general, users operate concurrently over a single adapter, or each user operates over multiple adapters. DLC device managers vary based on their protocol constraints.

Figure 43 on page 615 illustrates a multiple user configuration:

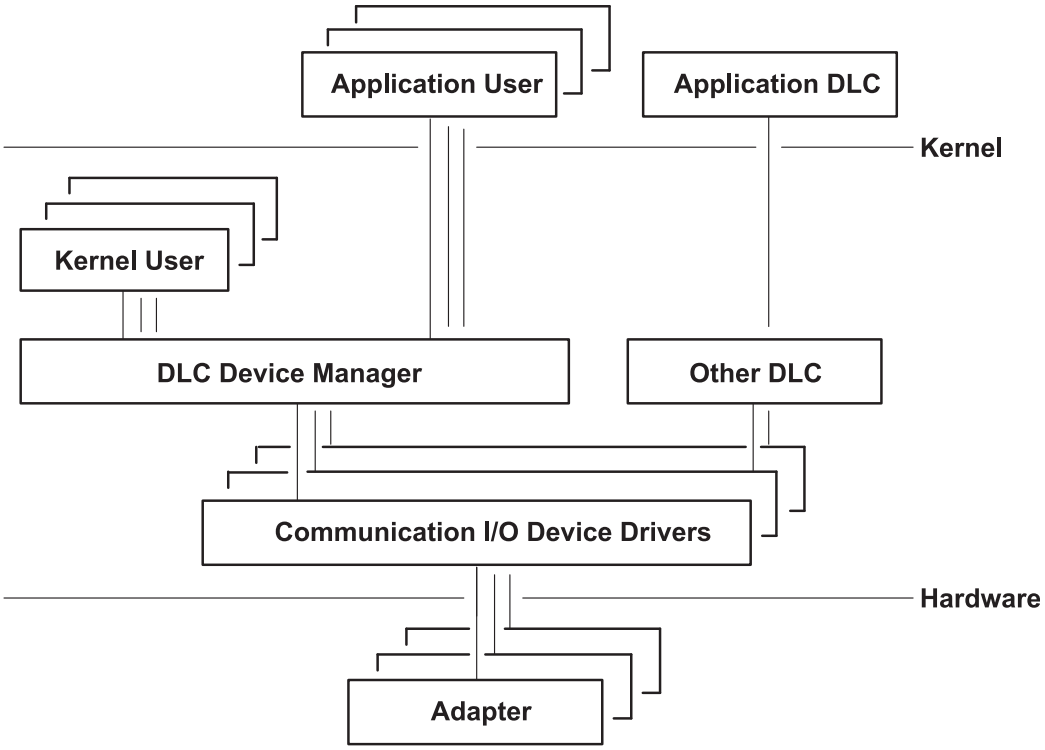


Figure 43. Multiple user and multiple adapter configuration

This illustration is another view of the kernel level between the application user and the adapter. It shows multiple entities representing multiple users.

GDLC criteria

A GDLC interface must meet the following criteria.

- Be flexible and accessible to both application and kernel users.
- Have multiple user and multiple adapter capability, allowing protocols to take advantage of multiple sessions and ports.
- Support both connection-oriented and connectionless services where possible.
- Allow transparent data transfer for special requirements beyond the scope of the DLC device manager in use.

GDLC interface

Each DLC device manager is a standard `/dev` entry operating in the kernel as a multiplexed device manager for a specified protocol.

For an adapter not in use by DLC, each **open** subroutine to a DLC device manager creates a kernel process. An **open** subroutine is also issued to the target adapter device handler. If needed, issue additional **open** subroutines for multiple DLC adapter ports of the same protocol. Any **open** subroutine targeting the same port does not create additional kernel processes, but links the **open** subroutine with the existing process. There is always one kernel process for each port in use.

The internal structure of a DLC device manager has the same basic structure as a kernel device handler, except that a kernel process replaces the interrupt handler in asynchronous events. The read, write, I/O control, and select blocks function as shown in the "Standard Kernel Device Manager" figure.

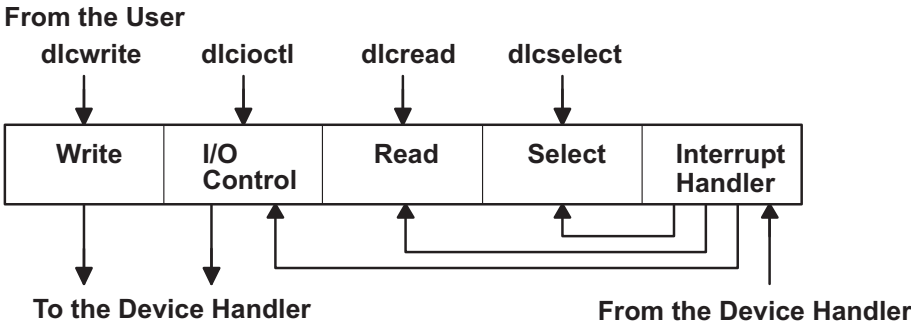


Figure 44. Standard kernel device manager

This illustration shows the internal structure of a DLC device manager. This structure consists of a Write, I/O Control, Read, Select, and Interrupt Handler. The Device Manager receives information from the user where it is passed to the various areas before it is passed on to the Device Handler.

GDLC data link controls

You can install DLCs separately or in a group. A DLC device manager is automatically added into the kernel and set to the "Available" state for each type of DLC installed.

Installation can be verified by issuing the **lslpp** command as follows:

```
lslpp -h dlctype
```

where *dlctype* is one of the following:

Item	Description
bos.dlc.8023	IEEE Ethernet (802.3) Data Link Control
bos.dlc.ether	Standard Ethernet Data Link Control
bos.dlc.fddi	FDDI Data Link Control
bos.dlc.sdlc	SDLC Data Link Control
bos.dlc.token	Token-Ring Data Link Control

Information about an installed DLC can be displayed through the System Management Interface Tool (SMIT), or the command line. On heavily used systems or communications ports, it might be necessary to change the DLC attributes to fine-tune the DLC performance. If receive performance is slow, and the system error log indicates that ring queue overflow is occurring between the DLC and its device handler, increase the DLC queue depth for incoming data. Finally, it is advisable to remove an installed DLC from the kernel when it is not needed for a lengthy period of time. This removal does not remove the DLC from the system, but allows kernel resources to be freed for other tasks until the DLC is needed again. Instructions for all of these tasks are in “DLC device driver management” on page 619.

GDLC interface ioctl entry point operations

The generic data link control (GDLC) interface supports these **ioctl** subroutine operations.

Item	Description
DLC_ENABLE_SAP	Enables a service access point (SAP).
DLC_DISABLE_SAP	Disables a SAP.
DLC_START_LS	Starts a link station on a particular SAP as a caller or listener.
DLC_HALT_LS	Halts a link station.
DLC_TRACE	Traces a link station's activity for short or long activities.
DLC_CONTACT	Contacts a remote station for a particular local link station.
DLC_TEST	Tests the link to a remote for a particular local link station.
DLC_ALTER	Alters a link station's configuration parameters.
DLC_QUERY_SAP	Queries statistics of a particular SAP.
DLC_QUERY_LS	Queries statistics of a particular link station.
DLC_ENTER_LBUSY	Enters local-busy mode on a particular link station.
DLC_EXIT_LBUSY	Exits local-busy mode on a particular link station.
DLC_ENTER_SHOLD	Enters short-hold mode on a particular link station.
DLC_EXIT_SHOLD	Exits short-hold mode on a particular link station.
DLC_GET_EXCEP	Returns asynchronous exception notifications to the application user. **Note:** This **ioctl** subroutine operation is not used by the kernel user since all exception conditions are passed to the kernel user through their exception handler.
DLC_ADD_GRP	Adds a group or multicast receive address to a port.
DLC_DEL_GRP	Removes a group or multicast receive address from a port.
DLC_ADD_FUNC_ADDR	Adds a group or multicast receive functional address to a port.
DLC_DEL_FUNC_ADDR	Removes a group or multicast receive functional address from a port.
IOCINFO	Returns a structure that describes the GDLC device manager. See the `/usr/include/sys/devinfo.h` file format for more information.

GDLC service access point

A service access point (SAP) identifies a particular user service that sends and receives a specific class of data.

This allows different classes of data to be routed separately to their corresponding service handlers. Those DLCs that support multiple concurrent SAPs have addresses known as Destination SAP and Source SAP imbedded in their packet headers. DLCs that can only support a single SAP do not need or use SAP addressing, but still have the concept of enabling the one SAP. In general, there is a SAP enabled for each DLC user on each port.

Most SAP address values are defined by IEEE standardized network-management entities or user-defined values as specified in the *Token-Ring Network Architecture Reference*. Some of the common SAP addresses are:

Item	Description
Null SAP (0x00)	Provides some ability to respond to remote nodes even when no SAP has been enabled. This SAP supports only connectionless service and responds only to exchange identification (XID) and TEST Link Protocol Data Units (LPDUs).
SNA Path Control (0x04)	Denotes the default individual SAP address used by Systems Network Architecture (SNA) nodes.
PC Network NETBIOS (0xF0)	Used for all DLC communication that is driven by Network Basic Input/Output System (NetBIOS) emulation.
Discovery SAP (0xFC)	Used by the local area network (LAN) name-discovery services.
Global SAP (0xFF)	Identifies all active SAPs.

GDLC link station

A link station (LS) identifies an attachment between two nodes for a particular SAP pair.

This attachment can operate as a connectionless service (datagram) or connection-oriented service (fully sequenced data transfer with error recovery). In general, there is one LS started for each remote attachment.

GDLC Local-Busy mode

When an LS is operating in a connection-oriented mode, it needs to stop the remote station's sending of information packets for reasons such as resource outage. Notification can then be sent to the remote station to cause the local station to enter local-busy mode.

After resources become available, the local station notifies the remote that it is no longer busy and that information packets can flow again. Only sequenced information packets are halted with local-busy mode. All other types of data are unaffected.

GDLC Short-Hold mode

You can use the short-hold mode of operation when operating over certain data networks.

Short-hold mode is useful with data networks that have the following characteristics:

- Short call-setup time
- Tariff structure that specifies a relatively small fee for the call setup compared to the charge for connect time.

During short-hold mode, an attachment between two stations is maintained only while there is data available for transfer between the two stations. When there is no data to send, the attachment is cleared after a specified timeout period and only reestablished when there is new data to transfer.

GDLC link testing and tracing

To test an attachment between two stations, instruct an LS to send a test packet from the local station. This packet is echoed back from the remote station if the attachment is operating correctly.

Some data links are limited in their support of this function due to protocol constraints. SDLC, for example, only generates the test packet from the host or primary station. Most other protocols, however, allow test packets to be initiated from either station.

To trace a link, line data, and special events (such as station activation, termination, and time outs), obtain a generic trace channel and instruct an LS to write its trace logs into the generic trace facility for each LS. This function helps determine the cause of certain communications attachment problems. Both short and long trace entries are supported.

GDLC statistics

Both SAP and LS statistics can be queried by a GDLC user.

The statistics for a SAP consist of the current SAP state and information about the device handler. LS statistics consist of the current station states and various reliability, availability, and serviceability counters that monitor the activity of the station from the time it is started.

GDLC special kernel services

Generic data link control (GDLC) provides special services for a kernel user.

However, a trusted environment must exist within the kernel. Instead of the DLC device manager copying asynchronous event data into user space, the kernel user must specify function pointers to special routines called function handlers. Function handlers are called by the DLC at the time of execution. This allows maximum performance between the kernel user and the DLC layers. Each kernel user is required to restrict the number of function handlers to a minimum path length and use the communications memory buffer (mbuf) scheme.

A function handler must never call another DLC entry directly. This is because direct calls are made under lock, causing a fatal sleep. The only exception to this rule is that a kernel user might call the **dlcwritex** entry point during its service of any of the four receive data functions. Calling the **dlcwritex** entry point allows immediate responses to be generated without an intermediate task switch. Special

logic is required within the DLC device manager to check the process identification of the user calling a write operation. If it is a DLC process and the internal queuing capability of the DLC has been exceeded, the write is sent back with a bad return code (**EAGAIN** return value) instead of putting the calling process (DLC) to sleep. It is then up to the calling user subroutine to return a special notification to the DLC from its receive data function to ensure a retry of the receive buffer at a later time.

The user-provided function handlers are:

Item	Description
Datagram Data Received Routine	Called any time a datagram packet is received for the kernel user.
Exception Condition Routine	Called any time an asynchronous event occurs that must notify the kernel user, such as SAP Closed or Station Contacted.
I-Frame Data Received Routine	Called each time a normal sequenced data packet is received for the kernel user.
Network Data Received Routine	Called any time network-specific data is received for the kernel user.
XID Data Received Routine	Called any time an exchange identification (XID) packet is received for the kernel user.

The **dlcread** and **dlcselect** entry points for DLC are not called by the kernel user because the asynchronous functional entries are called directly by the DLC device manager. Generally, any queuing of these events must occur in the user's function handler. If, however, the kernel user cannot handle a particular receive packet, the DLC device manager may hold the last receive buffer and enter one of two special user-busy modes:

User-Terminated Busy Mode (I-frame only)

If the kernel user cannot handle a received I-frame (due to problems such as queue blockage), a DLC_FUNC_BUSY return code is given back, and DLC holds the buffer pointer and enters local-busy mode to stop the remote station's I-frame transmissions. The kernel user must call the Exit Local Busy function to reset local-busy mode and start the reception of I-frames again. Only normal sequenced I-frames can be stopped. XID, datagram, and network data are not affected by local-busy mode.

Timer-Terminated Busy Mode (all frame types)

If the kernel user cannot handle a particular receive packet and wants DLC to hold the receive buffer for a short period and then re-call the user receive function, a DLC_FUNC_RETRY return code is given back to DLC. If the receive packet is a sequenced I-frame, the station enters local-busy mode for that period. In all cases, a timer is started; once the timer expires, the receive data functional entry is called again.

DLC device driver management

A DLC must be added to the system prior to use.

Each installed DLC is automatically added after installation and at each system restart (see “GDLC data link controls” on page 616). If a DLC has been removed without a subsequent restart, it can be re-added.

Table 109. Tasks for managing DLC device drivers

Task	SMIT fast path	Command or file
Add an Installed DLC	Choose one (by device driver name): `smit cmddlc_sdlc` `smit cmddlc_token` `smit cmddlc_qllc` `smit cmddlc_ether`[1] `smit cmddlc_fddi` then select **Add**	**mkdev**[2]
Change DLC Attributes[3,4]	Choose one (by device driver name): `smit cmddlc_sdlc_ls` `smit cmddlc_token_ls` `smit cmddlc_qllc_ls` `smit cmddlc_ether_ls`[1] `smit cmddlc_fddi_ls`	**chdev**[2]
Start DLC Local Area Network Monitor Trace[5]	`smit trace`	**trace -j** *nnn* where the value *nnn* is the hook ID to be traced
Stop DLC Local Area Network Monitor Trace	`smit trcstop`	**trcstop**[2]
Generate DLC Local Area Network Monitor Trace Report	`smit trcrpt`	**trcrpt -d** *nnn* where the value *nnn* is the hook ID to be reported
List Current DLC Information[3]	Choose one (by device driver name): `smit cmddlc_sdlc_ls` `smit cmddlc_token_ls` `smit cmddlc_qllc_ls` `smit cmddlc_ether_ls`[1] `smit cmddlc_fddi_ls`	**lsdev**[2] or **lsattr**[2]
Remove a DLC[3,6]	Choose one (by device driver name): `smit cmddlc_sdlc_rm` `smit cmddlc_token_rm` `smit cmddlc_qllc_rm` `smit cmddlc_ether_rm`[1] `smit cmddlc_fddi_rm`	**rmdev**[2]

Note:

1. The SMIT fast path for an Ethernet device manager includes both Standard Ethernet and IEEE 802.3 Ethernet device managers.
2. Details about command line options are provided in the command descriptions for **mkdev**, **chdev**, **trace**, **trcstop**, **trcrpt**, **lsdev**, **lsattr**, or **rmdev** in *Commands Reference, Volume 4*.
3. A DLC must be installed and added before you can list, show, change, or remove its attributes (see "GDLC data link controls" on page 616). An attribute change is only successful if there are no active opens against the target DLC. Before issuing the change action, the user might have to stop services such as SNA, OSI, or NetBIOS from using the DLC.
4. Changing the receive-queue size directly affects system resources. Make this change only if the DLC is having receive-queue problems, such as sluggish performance or overflows between the DLC and its device handler.
5. Exercise caution when enabling the monitor trace since it directly affects the performance of the DLCs and their associates.
6. Removing a DLC is only successful if there are no active opens against the target DLC. Before issuing the remove action, the user may have to stop services such as SNA, OSI, or NetBIOS from using the DLC.

Communications and networks adapters reference

Different configuration scenarios for both PCI adapters and asynchronous adapters can be referenced here.

PCI adapters

Installation and configuration information for PCI adapters is introduced here.

Topics discussed are support and configuration for the PCI Wide Area Network (WAN) adapters ("2-Port Multiprotocol HDLC network device driver" on page 621 and "ARTIC960Hx PCI adapter" on page 621).

2-Port Multiprotocol HDLC network device driver

The 2-Port Multiprotocol adapter high-level data link control (HDLC) device driver is a component of the communication I/O subsystem. This device driver provides support for the HDLC operation over the 2-Port Multiprotocol adapter at speeds up to 1.544Mbps.

The following options provide access to the 2-Port Multiprotocol HDLC Network device driver:

- Systems Network Architecture (SNA)
- The synchronous data link control (SDLC) version of the GDLC Programming Interface
- User-written applications compatible with the SDLC MPQP-API (Multiprotocol Quad Port-Application Programming Interface)

Note: The above options require use of the `mpc`*n* special file, which allows access to the 2-Port Multiprotocol adapter's HDLC device driver through the SDLC COMIO device driver emulation subsystem. This subsystem must be installed and configured for each HDLC network device.

- User-written applications compatible with the HDLC Common Data Link Interface (CDLI) API

The 2-port Multiprotocol adapter device driver allows connectivity to remote host systems using the 2-port Multiprotocol adapter, either directly over a leased line or over switched circuits. The device driver can provide a gateway between work group environments and remote data processing facilities.

2-Port Multiprotocol adapter configuration

Use these explanations to configure a 2-Port Multiprotocol adapter.

Table 110. Tasks for configuring the 2-port Multiprotocol adapter

Task	SMIT fast path
Add a device driver to the adapter	`smit mkhdlcdpmpdd`
Reconfigure the device driver on the adapter	`smit chhdlcdpmpdd`
Remove a device driver on the adapter	`smit rmhdlcdpmpdd`
Make a defined device driver available	`smit cfghdlcdpmpdd`
Add an SDLC COMIO emulator on the adapter	`smit mksdlcsciedd`
Reconfigure the SDLC COMIO emulator on the adapter	`smit chsdlcsciedd`
Remove an SDLC COMIO emulator on the adapter	`smit rmsdlcsciedd`
Make a defined SDLC COMIO emulator available	`smit cfgsdlcsciedd`

ARTIC960Hx PCI adapter

The ARTIC960Hx PCI adapter MPQP COMIO device driver emulator is a component of the communication I/O subsystem. This device driver provides support for the ARTIC960Hx PCI adapter at a maximum speed of 2M bps.

The modems used must provide the clocking, since only external clocking is supported.

The following options provide access to the ARTIC960Hx PCI adapter MPQP COMIO device driver:

- Systems Network Architecture (SNA)
- The generic data link control (GDLC) Programming Interface
- User-written applications compatible with the MPQP-API (Multiprotocol Quad Port-Application Programming Interface), such as SDLC and BiSync applications.

These options require use of the `mpq`*x* special file, which allows access to the ARTIC960Hx PCI adapter through the MPQP COMIO emulation device driver. This device driver must be installed and configured for each port on the ARTIC960Hx PCI adapter. The `mpq`*x* special file resides in the `/dev` directory.

Note: The *x* in `mpq`*x* specifies the instance of the device driver; for example, `mpq0`.

The MPQP COMIO emulation device driver allows connectivity to remote host systems using the ARTIC960Hx PCI adapter, either directly over a leased line. The device driver can provide a gateway between work group environments and remote data processing facilities.

MPQP COMIO emulation driver configuration over the ARTIC960Hx PCI adapter

Use these explanations to configure the MPQP COMIO emulation driver over the ARTIC960Hx PCI adapter.

Table 111. Tasks for configuring the MPQP COMIO emulation driver

Task	SMIT fast path
Add a device driver	`smit mktsdd`
Reconfigure the MPQP COMIO emulation driver	`smit chtsdd`
Remove a device driver	`smit rmtsdd`
Configure a defined device driver	`smit cfgtsdd`
Add a port	`smit mktsdports`
Reconfigure an MPQP COMIO emulation port	`smit chtsdports`
Remove a port	`smit rmtsdports`
Configure a defined port	`smit cfgtsdports`
Trace the MPQP COMIO emulation driver	`smit trace_link`

Asynchronous adapters

The standard, 8-port, and 16-port asynchronous adapters that are listed in this table.

The following table summarizes these products:

Table 112. Asynchronous adapters

Asynchronous Attachment	Bus Type	Feature Code or Machine Type (Model)	Maximum Data Rate per Port (KBits/sec)	Salient Features
8-Port EIA 232	Micro Channel	2930	76.8	Pervasive standard
8-Port EIA 422A	Micro Channel	2940	76.8	Greater distance
8-port MIL-STD 188	Micro Channel	2950	Selectable based on baud rate generator clock speed of UART.	MIL-STD 188-114 for unbalanced voltage digital interface
8-port EIA 232	ISA	2931	115.2	Greater efficiency
8-port EIA 232	ISA	2932	115.2	Greater efficiency
8-port EIA 422	PCI	2943	230	Greater efficiency
16-Port EIA 232	Micro Channel	2955	76.8	Local connection focus
16-Port EIA 422A	Micro Channel	2957	76.8	Greater distance
-	ISA	2933	-	-
-	PCI	2944	-	-

The following table shows the detailed product characteristics.

Table 113. Asynchronous Attachment Product Characteristics

	Native Serial Ports	8-port		16-port	128-port with RAN	
		MC	ISA		MC	ISA
Number of asynchronous ports per adapter	n/a	8	8	16	128	128
Maximum number of adapters	n/a	8	7	8	7	7
Maximum number of asynchronous ports	2 or 3	64	56	128	896	896
Number of asynchronous ports per RAN	n/a	n/a	n/a	n/a	16	16
Maximum number of RANs	n/a	n/a	n/a	n/a	56	56
Maximum speed (KBits/sec)	Selectable based on baud rate generator clock speed of UART.	76.8	115.2	76.8	230	230
Attachment method	standard	direct	direct	direct	node	node
Asynchronous electrical interfaces supported	EIA 232	EIA 232 EIA 422A[4]MIL-STD[4]188-114[4]	EIA 232 EIA 422A	EIA 232 EIA 422A	EIA 232 EIA 422	EIA 232 EIA 422
Standard connector	DB25M/ MODU	DB25M	DB25M	DB25M	RJ-45[2]	RJ-45[2]
DB25 cable options	n/a	n/a	n/a	n/a	RJ-45-DB25	RJ-45-DB25
Rack mount option	n/a	n/a	n/a	n/a	yes	yes
Power supply	n/a	n/a	n/a	n/a	external	external
Signals supported (EIA 232)	TxD RxD RTS CTS DTR DSR DCD RI	TxD RxD RTS RTS CTS DTR DSR DCD RI	TxD RxD RTS CTS DTR DSR DCD RI	TxD RxD RTS[3]-DTR -DCD -	TxD RxD RTS CTS DTR DSR DCD RI	TxD RxD RTS CTS DTR DSR DCD RI

Note:

1. Socket accepts 8p RJ-45, 6p RJ-11, or 4p RJ-11 plugs with a reduction in signals supported.
2. RTS is tied high (+12V) in the fanout connector box of the 16-port interface cable EIA 232 (FC 2996).
3. Micro Channel Only.

Each product offering is characterized by a representative scenario for its strengths. The following are recommendations for each:

Related concepts:

"TTY terminal device" on page 547
A tty terminal device is a character device that performs input and output on a character-by-character basis.

Related information:

2-Port Asynchronous EIA-232 PCI Adapter Installation and Using Guide

8-Port Micro Channel

The features of the 8-port Micro Channel adapter include a Micro Channel bus slot available for asynchronous I/O.

Other features include the following:

- Fewer than eight ports with little or no expansion.

- All local terminals located within 61 meters (200 feet) from the system.
- Remote terminals needed (support through OEM multiplexer/modem).
- Device bandwidth demand low to moderate (up to 76.8 Kbps).

8-Port ISA bus EIA 232 or EIA 232/EIA 422

The features of the 8-Port ISA bus EIA 232 or EIA 232/EIA 422 adapter include an ISA slot.

Other features include the following:

- Fewer than eight ports required with little or no expansion.
- Requires all EIA 232, all EIA 422, or a mix of EIA 232 and EIA 422 ports.
- Offload character interrupt and terminal I/O processing from the main CPU.
- Asynchronous speeds to 115.2 Kbps.
- Maximum performance for high speed (28.8 Kbps) modems with data compression.

16-Port Micro Channel

The features of the 16-port Micro Channel adapter include a Micro Channel bus slot available for asynchronous I/O.

Other features include the following:

- Eight ports now, fewer than 16 ports with little or no expansion.
- All local terminals located within 61 meters (200 feet) from the system.
- Remote terminals needed (support through OEM multiplexer/modem).
- Devices do not need all EIA 232 signals.
- Device bandwidth demand low to moderate (up to 38.4 Kbps for asynchronous devices).

128-Port adapter (Micro Channel, ISA)

The features of the 128-port Micro Channel or ISA adapter include sixteen ports with expansion of up to 128 ports without additional slots.

Other features include the following:

- A Micro Channel, ISA, or PCI bus slot available for asynchronous I/O. (For further information about PCI, see "Product selection considerations" on page 538.)
- Most distant terminal located about 300 meters (1000 feet) from the system at maximum data rate for Micro Channel and ISA Adapters.
- Terminals planned: nearby or on premises, distant on premises, and remote.
- Need high asynchronous throughput with low processor demand.
- Need terminal attached printer capability.
- Need to connect to remote premises through fiber-optic or synchronous modems.

Listing defined Micro Channel 128-Port asynchronous adapters using SMIT

Use this procedure to list all defined 128-port asynchronous adapters regardless of whether they are available.

1. Use the `smit lsd128psync` fast path. The system scans for the information and displays it.
2. Exit the SMIT interface.

8-Port asynchronous ISA/PCI adapter

The 8-port asynchronous ISA adapter is a multichannel, intelligent, serial communications feature available for POWER processor-based computers.

The ISA adapters contain 128K of dual-ported high-speed Random Access Memory (RAM) used for program code and data buffering. The asynchronous ports are run by a 32-bit 16 MHz IDT 3041 processor that supports throughput speeds of 115 Kbps.

The 3041 processor and dual-ported RAM help to offload much of the character processing from the system. Large blocks of data are transferred directly to the adapter, and then sent out on serial ports one character at a time.

The dual-ported RAM is accessible for read and write operations by both the adapter and the computer. The computer sees the dual ported RAM as its own memory and accesses it using the same high-speed memory referencing commands it uses for internal memory.

The 8-port EIA 232 ISA adapter supports EIA 232 devices only. This adapter requires the device package `devices.isa.cxia` to be installed on the system.

The 8-port EIA 232/422 ISA adapter supports EIA 232 and EIA 422 devices. Both device types may be configured in any combination on a per-port basis. This adapter requires the device package `devices.isa.pc8s` to be installed on the system.

The above packages require the `devices.common.IBM.cx` package.

Installing 8-Port adapters:

ISA adapters cannot be autodetected by the operating system and must be manually installed.

1. To configure the IBM 8-Port asynchronous EIA 232/EIA 422 ISA adapters, use the `smit mkdev_isa` fast path to access the **Add an ISA Adapter screen**.
2. Select **pcxr** (for the 8-port EIA 232 adapter) or **pc8s** (for the 8-port EIA 232/EIA 422 adapter) and press Enter.
3. Select the appropriate bus and press Enter.
4. In the Bus I/O Address field, set the address to the address of the adapter (set by DIP switches on the adapter). For additional information on DIP switches, refer to the *8 Port Asynchronous ISA Adapter Installation Guide*. The remainder of the adapter configuration is done automatically when the system displays `saX Available`.
5. When you have finished, select **Do**.

stty-cxma is a utility program that sets and displays the terminal options for the Micro Channel 128-port and the ISA 8- and 128-port adapters and is located in `/usr/lbin/tty` directory. The format is:

```
stty-cxma [-a] [option(s)] [ttyname]
```

With no options, **stty-cxma** displays all special driver settings, modem signals, and all standard parameters displayed by **stty(1)** for the tty device referenced by standard input. Command options are provided to change flow control settings, set transparent print options, force modem control lines, and display all tty settings. Any unrecognized options are passed to **stty(1)** for interpretation. The options are the same as those used for PCI adapters. For more information, see “stty-cxma terminal options” on page 575.

Standard I/O ports

Most system unit models have two integrated (standard) EIA 232 asynchronous serial ports.

The model M20/M2A features a single integrated asynchronous serial port that can be converted to support two serial devices using an optional fanout cable. EIA 232 asynchronous serial devices can be attached directly to the standard serial ports using standard serial cables with 9-pin or 25-pin D-shell connectors.

Note: For the Itanium-based platform, EIA 232 asynchronous serial devices can be attached directly to the standard serial ports using standard serial cables with 9-pin D-shell connectors.

The machines capable of multiprocessing have three serial ports.

Configuring an EIA 232 asynchronous terminal device:

This procedure allows you to define and configure a tty device connected to a standard serial port, an 8-port, or 16-port asynchronous adapter.

1. Use the `smit mktty` fast path to access the **Add a TTY** menu.
2. Select **Add a TTY**.
3. Select **tty rs232 Asynchronous Terminal**.
4. Make a selection from the available standard I/O, 8-port, or 16-port adapters displayed on the screen. If no adapters are displayed or if they are in a defined state, check the configuration, cabling, and setup again.
5. In the displayed dialog fields, you can add or change the tty attributes.
6. When you have finished, select **Do**.

Configuring an EIA 232 asynchronous printer/plotter device:

This procedure allows you to define and configure a printer/plotter device connected to a standard serial port, an 8-port asynchronous adapter, or a 16-port asynchronous adapter.

1. To create a printer/plotter device on an asynchronous adapter, use the `smit pdp` fast path to access the **Printer/Plotter Devices** menu.
2. Select **Add a Printer/Plotter**.
3. Make a selection from the list of printer and plotter types shown on the screen, and press Enter. For this example, the following selection was made:
   ```
   osp Other serial printer
   ```
4. Select the **rs232** option.
5. Make a selection from the available 8-port controllers on the screen. If no controllers are displayed or if they are shown in a defined state, check the configuration, cabling, and setup again.
6. In the displayed dialog fields, you can add or change the printer/plotter device attributes.
7. When you have finished, select **Do**.

8-Port Micro Channel asynchronous adapters

The family of asynchronous adapters is based on a common functional design. The individual adapter characteristics, however, are determined by the supported device interfaces.

Note: The following section is not applicable to the Itanium-based platform.

The family consists of three adapters:

- 8-Port Asynchronous Adapter - EIA 232
- 8-Port Asynchronous Adapter - MIL-STD-188
- 8-Port Asynchronous Adapter - EIA 422A

The family of 8-port adapters is based on the dual universal asynchronous receiver and transmitter (DUART) chip providing two serial communications channels.

The following sections contain detailed information about 8-port adapters.

8-Port asynchronous adapter - EIA 232:

The EIA 232 is an 8-Port asynchronous adapter that provides support for attaching a maximum of eight EIA 232D asynchronous serial devices (such as modems, terminals, plotters, and printers) to a system unit.

The system must be based on a Micro Channel bus or an ISA bus and support up to eight 8-port adapters.

This adapter is fully programmable and supports asynchronous communications only. It can also add and remove start and stop bits and supports even, odd, or no parity on serial data. A programmable baud rate generator allows operation from 50 to 38,400 bps for the Micro Channel bus and 50 to 115,200 bps for the ISA bus. The adapters support 5-, 6-, 7-, or 8-bit characters with 1, 1.5, or 2 stop bits. A priority interrupt system controls transmit, receive, error, line status, and data set interrupts.

Installing the 8-Port asynchronous adapter:

The 8-port asynchronous adapter fits into a single Micro Channel slot in the system. Use these steps to install the adapter.

1. Verify that all users are logged off the system and run the following command:
   ```
   shutdown -F
   ```
2. When the **shutdown** command completes, turn the system power switch to the off position.
3. Open the system case and insert the 8-port asynchronous adapter into a free Micro Channel slot.
4. Attach the 78-pin D-shell connector from the 8-port interface cable to the 8-port adapter.
5. Put the cover panels back on the system unit.
6. Push the system power switch to the on position. The system will recognize and configure the 8-port adapter during the boot process.
7. After the boot completes, log in using the root user ID boot process.
   ```
   lsdev -Cc adapter | pg
   ```

Only those adapters that are in an available state are ready for use by the system.

If the newly installed adapter is *not* available, then verify:

- The adapter is installed correctly into the Micro Channel slot.
- All necessary cabling is attached and fitted tightly into place.
- Run the command: **errpt -a | pg** and examine the system error report for problems relating to the adapters.
- Run the command: **cfgmgr -v | pg**. This command will attempt to reconfigure the adapter without rebooting. Observe the paged output for errors.

If running **cfgmgr** fails, a reboot will be necessary.

8-Port asynchronous adapter hardware information:

The system interface presents a 3-bit address and 8-bit data as well as control lines to the DUART chip. Data from the system interface is serialized for transmission to an external device. The serial data can include a parity bit at the byte boundary. Conversely, data from an external device is deserialized for transmission to the system interface. This data may also include a parity bit, which can be optionally checked. As an option, the channel can operate in first-in-first-out (FIFO) mode.

In FIFO mode, up to 16 bytes can be buffered in both the transmitter and receiver. The serial interface uses start-stop protocol for both data transmission and reception. That is, each byte (plus the parity bit) is framed by one or more start bits and stop bits, which allows synchronization on an individual character (byte) basis.

The DUART chip uses a 12.288 MHz oscillator to generate its internal timing to drive the transmitter and receiver logic. The channel supports full duplex operation. Four DUART chips are implemented on each 8-port adapter.

Thirteen system-accessible registers are available. Programmable features on each channel include:

- Character length: 5, 6, 7, or 8 bits
- Parity generation/detection: Even, odd, or none
- Number of stop bits: 1, 1.5, or 2
- Enable/disable interrupts. Received data available
- Transmitter holding register empty
- Line status
- Overrun error
- Parity error
- Framing error
- Break.

The following table is a summary of port (device interface) characteristics for the adapters.

Table 114. 8-Port asynchronous adapter port characteristics

Parameter	EIA 232	MIL-STD 188	EIA 422A
Topology	Point to Point	Point to Point	Point to Point
Maximum data rate	138.4Kbps (MC)/115.2 (ISA)	138.4Kbps	138.4Kbps
Transmission media	Multiconductor	Multiconductor	Multiconductor
Number of cable wires	9 including signal ground	9 including signal ground	5 including signal ground
Maximum cable length	61 m (200 feet)	130 m at 38.4Kbps	1200 m < 90Kbps
Device connector	25-pin D	25-pin D	25-pin D
Electrical interface	Unbalanced	Unbalanced	Balanced
Bit encoding	Digital bi-level	Digital bi-level	Digital bi-level

The interrupt arbitration logic sets priority for adapters according to the following scheme:

```
Adapter   Priority
   1      Highest
   2        |
   3        |
   4        |
   5        |
   6        |
   7        |
   8      Lowest
```

Communications channel priority:

The DUART channels with pending interrupts are serviced according to a fixed-priority scheme.

The highest priority is assigned to port 0. Next in priority is port 1, and so forth. The lowest priority is port 7.

8-Port asynchronous adapters interrupt logic description:

The interrupt logic is divided into interrupt generation logic and interrupt arbitration logic.

Both logic sections are implemented on every 8-port adapter. The interrupt generation logic provides the interface to the system. This logic generates the system interrupt requests and contains the interrupt-sharing circuitry.

The function of the interrupt arbitration logic is to identify the 8-port adapter with the highest priority interrupt pending. The logic then places the interrupt information for the highest priority port in the Interrupt Arbitration register. This is accomplished in one read operation.

The interrupt arbitration logic is unique to the 8-port adapter and should not be confused with the Micro Channel arbitration logic.

8-Port interrupt generation logic:

The asynch adapter implements eight system interrupt request lines.

The adapter implements the following eight system interrupt request lines:
- IRQ 3
- IRQ 5
- IRQ 9
- IRQ 10
- IRQ 11
- IRQ 12
- IRQ 14
- IRQ 15

Only one request line is active during normal operation. All 8-port adapters in one system should use the same interrupt level for optimal system performance. The active line is selected by writing to the appropriate POS register during the setup cycle. The adapter supports interrupt sharing and implements an open collector configuration. In this arrangement, the interrupt line is pulled high by a system pull-up resistor. The adapter pulls the line low to indicate an active interrupt request.

8-Port interrupt arbitration logic:

The interrupt arbitration logic determines the priority for software service when two or more 8-port or 16-port adapters generate interrupts.

Up to eight 8-port adapters can co-reside and concurrently operate in a system. This logic provides the system with adapter and port identification as well as the interrupt type in a single read operation. After an interrupt request is detected, the system reads the 16-bit interrupt arbitration register, which is located at I/O address 0130.

8-Port asynchronous adapters MIL-STD 188 interface signals:

These interface signals are implemented on each port of the adapter.

Signal	Definition
Tx Data	Transmit Data
RTS	Request To Send
CTS	Clear To Send
DSR	Data Set Ready
Rx Data	Receive Data
DCD	Data Carrier Detect
DTR	Data Terminal Ready
RI	Ring Indicator
Sig Gnd	Signal Ground

8-Port MIL-STD 188 signal voltage levels:

Voltage levels for the MIL-STD 188 Adapter can be explained through normal mark and space polarity or mark and space polarity inversion.

Voltage levels for the MIL-STD 188 Adapter are explained in the following sections:

- Normal Mark and Space Polarity
- Mark and Space Polarity Inversion.

The signal is in the mark state when the voltage on the interchange circuit, measured at the interface point, is less than -4 V dc with respect to the signal ground. The signal is in the space state when the voltage is greater than +4 V dc with respect to the signal ground. The region between +4 V dc and -4 V dc is defined as the transition region and is not a valid level. The voltage that is less than -6 V dc or greater than +6 V dc is also not a valid level.

During the transmission of data, the mark state denotes binary 1 and the space state denotes binary 0.

For interface control circuits, the function is "on" when the voltage is greater than +4 V dc with respect to the signal ground and is "off" when the voltage is less than -4 V dc with respect to the signal ground. MIL-STD 188 signal levels are shown in the following table:

Table 115. MIL-STD 188 signal levels

Interchange Voltage	Binary State	Signal Condition	Interface Control Function
+ Voltage	0	Space	On
- Voltage	1	Mark	Off

Military standard MIL-STD 188 requires that adapters provide the capability to optionally invert the polarities of the mark and space states of the transmit and receive lines. The capability is provided independently on each port.

The DUART modem control register bit 3 (Out 2) is used for this purpose. When bit 3 is set to a value of 1, the polarities for the mark and space states are set to the normal state. When bit 3 is set to a value of 0, the polarities for the mark and space states are inverted.

The signal is in the *space state* when the voltage is less than -4 V dc with respect to the signal ground. The signal is in the mark state when the voltage is greater than +4 V dc with respect to the signal ground.

The region between +4 V dc and -4 V dc is defined as the *transition region* and is not a valid level. The voltage that is less than -6 V dc or greater than +6 V dc is also not a valid level.

The electrical characteristics of the 8-Port asynchronous MIL-STD 188 adapter ports conform to those sections of MIL-STD 188-114 that address an unbalanced voltage interface. The standard is dated March 24, 1976.

The adapter ports meet the functional requirements for asynchronous operation (start-stop protocol) as described in the EIA Standard 232C dated October 1969 and in the EIA Standard 232D dated January 1987.

8-Port asynchronous adapters EIA 422A interface signals:

The following EIA 422A interface signals are implemented on each port of the adapter.

Signal	Definition
TxA	Transmit Data
TxB	Transmit Data
RxA	Receive Data
RxB	Receive Data
Sig Gnd	Signal Ground

8-Port EIA 422A signal voltage levels:

The line driver produces a differential voltage in the range of 2 to 6 volts (measured at the generator interface point). The magnitude of the differential voltage at the receiver must be in the range of 200 millivolts to 6 volts (measured at the load interface point).

Measurements are taken at terminal A (positive lead) with respect to terminal B (negative lead). The following table describes the signal states with respect to voltage levels:

Table 116. 8-Port EIA 422A signal states

Interchange Voltage	Binary State	Signal Condition
+ Voltage	0	Space
- Voltage	1	Mark

The 8-Port asynchronous EIA 422A adapter supports indoor cabling up to 1200 m (4000 ft) in length. Cables of such lengths are susceptible to sudden voltage surges due to induced voltages such as indirect lightning strikes. Secondary surge protection circuitry is implemented on the EIA 422A adapter to protect it from these voltage surges. The surge protection circuitry is implemented on the adapter interface data lines.

Fail-safe circuitry has been added to the input leads of each EIA 422A receiver to prevent fault conditions when the receiver is not connected to a driver (open cable). The fail-safe circuitry sets the receiver to the mark state (binary 1) whenever the receiver is not connected to a driver.

The electrical characteristics of the 8-Port asynchronous EIA 422A adapter ports comply with the EIA Standard 422A dated December 1978.

8-Port asynchronous adapters EIA 232 interface signals:

These interface signals are implemented on each port of the 8-port asynchronous adapter.

The following interface signals are implemented on each port of the adapter:

Signal	Definition
TxD	Transmit Data
RTS	Request To Send
CTS	Clear To Send
DSR	Data Set Ready
RxD	Receive Data
DCD	Data Carrier Detect
DTR	Data Terminal Ready
RI	Ring Indicator
Sig Gnd	Signal Ground

8-Port EIA 232 signal voltage levels:

The signal is in the mark state when the voltage on the interchange circuit, measured at the interface point, is less than -3 V dc with respect to the signal ground. The signal is in the space state when the voltage is greater than +3 V dc with respect to the signal ground. The region between +3 V dc and -3 V dc is defined as the *transition region* and is not a valid level. Voltage less than -15 V dc or greater than +15 V dc is also not a valid level.

During the transmission of data, the mark state denotes binary state 1 and the space state denotes binary state 0.

For interface control circuits, the function is on when the voltage is greater than +3 V dc with respect to the signal ground and is off when the voltage is less than -3 V dc with respect to the signal ground. See the following table for EIA 232 signal levels:

Table 117. EIA 232 signal levels

Interchange Voltage	Binary State	Signal Condition	Interface Control Function
+ Voltage	0	Space	On
- Voltage	1	Mark	Off

The electrical characteristics of the 8-Port asynchronous EIA 232 adapter ports conform to the EIA Standard 232C dated October 1969 and to the EIA Standard 232D dated January 1987.

The adapter ports meet the functional requirements for asynchronous operation (start-stop protocol) as described in the EIA Standard 232C dated October 1969 and in the EIA Standard 232D dated January 1987.

8-Port asynchronous adapters control logic:

The PAL-based control logic section coordinates the activities of all major adapter functions.

It is clocked with a 40 MHz square-wave generator. It interfaces with the Micro Channel, and its functions include decoding addresses, checking address parity, responding with the proper I/O control signals, and driving the selected interrupt request (IRQ) line (one of eight IRQ lines).

The control logic interfaces with the other adapter logic blocks and in this capacity provides the control lines to the communication channels (DUART) and the interrupt arbitration logic. The control logic also interfaces with the data bus driver logic and provides control for the direction of data flow and for the selection data bytes, which are placed onto the local bus. It controls the data parity generator, parity checker, and latches.

16-Port asynchronous adapters

The family of adapters is based on a common functional design. The individual adapter characteristics, however, are determined by the supported device interfaces. The family consists of two adapters, the 16-Port EIA 422A asynchronous adapter and the 16-Port EIA 232 asynchronous adapter.

Note: The following section is not applicable to the Itanium-based platform.

The family of 16-port adapters is based on the dual universal asynchronous receiver and transmitter (DUART) chip, which provides two serial communications channels. More information on the DUART chip and its functionality can be found in the 16-Port Asynchronous Adapter Hardware Information.

16-Port asynchronous adapter - EIA 422A:

The 16-Port asynchronous adapter - EIA 232 provides support for attaching a maximum of 16 EIA 232 asynchronous serial devices (printers and terminals) to a system unit.

Up to eight adapters (any combination within the family) can be used in a single system unit.

This adapter is fully programmable and supports asynchronous communications only. It adds and removes start bits and stop bits. The adapters support even, odd, or no parity on serial data. A programmable baud-rate generator allows operation from 50 to 38400 bps. The adapters support 5-, 6-, 7-, or 8-bit characters with 1, 1.5, or 2 stop bits. A priority interrupt system controls transmit, receive, error, line status, and data set interrupts. The 16 connectors for device attachment are provided in the EIA 422A 16-port cable assembly.

The EIA 422A 16-port adapter has the following characteristics:

- Standard Micro Channel form factor card.
- Data rates up to 38.4K bps per port.
- 16 byte buffering on transmit and receive.
- Single 78-pin output connector (Multiport interface cable attaches to this connector).
- Surge protection circuitry.
- Supports cabling up to 1200 m (4000 ft).
- Supports the TxD and RxD interface signals.
- 8-bit/16-bit Micro Channel slave interface.

Installing the 16-Port asynchronous adapter:

The 16-port asynchronous adapter fits into a single Micro Channel slot in the server. To install the adapter, use these steps.

1. Verify that all users are logged off the system and run the following command:
   ```
   shutdown -F
   ```
2. When the **shutdown** command completes, turn the system power switch to the "off" position.
3. Open the server case and insert the 16-port asynchronous adapter into a free Micro Channel slot.
4. Attach the 78-pin D-shell connector from the 16-port interface cable to the 16-port adapter.
5. Put cover panels back on the system unit.
6. Push the system power switch to the On position. The system will recognize and configure the 16-port adapter during the boot process.

After the boot completes, log in using the root user ID and issue the following command to check the adapter availability:

```
lsdev -Cc adapter | pg
```

Only those adapters in the available state are ready for use by the system.

If the newly installed adapter is NOT available, then verify the following:

1. The adapter is installed correctly into the Micro Channel slot.
2. All necessary cabling is attached and fitted tightly into place.
3. Run the command: **errpt -a | pg** and examine the system error report for problems relating to the adapters.
4. Run the command: **cfgmgr -v | pg**. This command will attempt to reconfigure the adapter without rebooting. Watch the paged output for errors.
5. If running **cfgmgr** fails, a reboot will be necessary.

16-Port asynchronous adapter hardware information:

The system interface presents a 3-bit address and 8-bit data as well as control lines to the chip. Data from the system interface is serialized for transmission to an external device. The serial data may include a parity bit at the byte boundary. Conversely, data from an external device is deserialized for transmission to the system interface. This data may also include a parity bit, which can be optionally checked. As an option, the channel can operate in first-in-first-out (FIFO) mode.

In FIFO mode, up to 16 bytes can be buffered in both the transmitter and receiver. The serial interface uses start-stop protocol for both data transmission and reception. That is, each byte (plus parity bit) is framed by a start bit and stop bit, which allows synchronization on an individual character (byte) basis.

The DUART chip uses a 12.288 MHz oscillator to generate its internal timing to drive the transmitter and receiver logic. The channel supports full duplex operation. Eight DUART chips are implemented on each 16-port adapter.

Thirteen system-accessible registers are available. Programmable features on each channel include:

- Character length: 5,6, 7, or 8 bits
- Parity generation/detection: even, odd, or none
- Number of stop bits: 1, 1.5, or 2
- Enable/disable interrupts. Received data available
- Transmitter holding register empty
- Line status
- Overrun error
- Parity error
- Framing error
- Break.

The following table is a summary of port (device interface) characteristics for the adapters.

Table 118. 16-Port asynchronous adapter port characteristics

Parameter	EIA 232	EIA 422A
Topology	Point to Point	Point to Point
Maximum data rate (standard)	20Kbps	2Mbps
Maximum data rate (board)	38.4Kbps	38.4Kbps
Transmission media	Multiconductor	Multiconductor
Number of cable wires	5 including signal ground	5 including signal ground
Maximum cable length	61 m (200 ft)	1200 m < 90Kbps
Device connector	25-pin D	25-pin D
Electrical interface	Unbalanced	Balanced

Table 118. 16-Port asynchronous adapter port characteristics (continued)

Parameter	EIA 232	EIA 422A
Bit encoding	Digital bi-level	Digital bi-level

16-Port asynchronous adapters adapter board priority:

The interrupt arbitration logic sets priority for adapters according to a specific scheme.

```
Adapter   Priority
    0     Highest
    1       |
    2       |
    3       |
    4       |
    5       |
    6       |
    7       |
    8       |
    9       |
    10      |
    11      |
    12      |
    13      |
    14      |
    15    Lowest
```

The DUART channels with pending interrupts are serviced according to a fixed-priority scheme. The highest priority is assigned to port 0. Next in priority is port 1, and so forth. The lowest priority is port 15.

16-Port asynchronous adapters interrupt logic:

For 16-port asynchronous adapters, the interrupt logic is divided into interrupt generation logic and interrupt arbitration logic.

Both logic sections are implemented on every 16-port adapter. The interrupt generation logic provides the interface to the system. This logic generates the system interrupt requests and contains the interrupt-sharing circuitry.

The function of the interrupt arbitration logic is to identify the 16-port adapter with the highest priority interrupt pending. The logic then places the interrupt information of the highest priority port in the interrupt arbitration register. This is accomplished in one read operation.

The interrupt arbitration logic is unique to the 16-port adapters and should not be confused with the Micro Channel arbitration logic.

16-Port interrupt generation logic:

The 16-port asynchronous adapter implements eight system interrupt request lines.

The adapter implements the following eight system interrupt request (IRQ) lines:

- IRQ 3
- IRQ 5
- IRQ 9
- IRQ 10
- IRQ 11
- IRQ 12

- IRQ 14
- IRQ 15

Only one request line is active during normal operation. All 16-port adapters in one system should use the same interrupt level for optimal system performance. The active line is selected by writing to the appropriate POS register during the setup cycle. The adapter supports interrupt sharing and implements an open collector configuration as defined in the Micro Channel architecture. In this arrangement, the interrupt line is pulled high by a system pull-up resistor. The adapter pulls the line low to indicate an active interrupt request.

16-Port interrupt arbitration logic:

The interrupt arbitration logic determines the priority for software service when two or more 8-port or 16-port adapters generate interrupts.

Up to eight 8-port or 16-port adapters can co-reside and concurrently operate in a system. This logic provides the system with adapter and port identification as well as the interrupt type in a single read operation. Upon detection of an interrupt request, the system reads the 16-bit interrupt arbitration register located at I/O address 0130.

16-Port asynchronous adapters EIA 232 interface signals:

The following interface signals are implemented on each port of the 16-port asynchronous adapter.

Signal	Definition
TxD	Transmit data
DCD	Data carrier detect
DTR	Data terminal ready
RxD	Receive data
Sig Gnd	Signal ground

16-Port EIA 232 signal voltage levels:

The signal is in the mark state when the voltage on the interchange circuit, measured at the interface point, is less than -3 V dc with respect to the signal ground. The signal is in the space state when the voltage is greater than +3 V dc with respect to the signal ground. The region between +3 V dc and -3 V dc is defined as the transition region and is not a valid level. Voltage less than -15 V dc or greater than +15 V dc is also not a valid level.

During the transmission of data, the mark state denotes binary state 1 and the space state denotes binary state 0.

For interface control circuits, the function is on when the voltage is greater than +3 V dc with respect to the signal ground and is off when the voltage is less than -3 V dc with respect to the signal ground. See the following table for EIA 232 signal levels.

Table 119. EIA 232 signal levels

Interchange Voltage	Binary State	Signal Condition	Interface Control Function
+ Voltage	0	Space	On
- Voltage	1	Mark	Off

The electrical characteristics of the 16-Port asynchronous EIA 232 adapter ports conform to the EIA Standard 232C dated October 1969 and to the EIA Standard 232D dated January 1987.

The adapter ports meet the functional requirements for asynchronous operation (start-stop protocol) as described in the EIA Standard 232C dated October 1969 and in the EIA Standard 232D dated January 1987.

16-Port asynchronous adapters EIA 422A interface signals:

These EIA 422A interface signals are implemented on each port of the 16-Port asynchronous adapter.

Signal	Definition
TxA	Transmit Data
TxB	Transmit Data
RxA	Receive Data
RxB	Receive Data
Sig Gnd	Signal Ground

16-Port EIA 422A signal voltage levels:

The line driver produces a differential voltage in the range of 2 to 6 volts (measured at the generator interface point). The magnitude of the differential voltage at the receiver must be in the range of 200 millivolts to 6 volts (measured at the load interface point).

Measurements are taken at terminal A (positive lead) with respect to terminal B (negative lead). The following table describes the signal states with respect to voltage levels:

Table 120. 16-Port EIA 422A signal states

Interchange Voltage	Binary State	Signal Condition
+ Voltage	0	Space
- Voltage	1	Mark

The 16-Port Asynchronous EIA 422A adapter supports indoor cabling up to 1200 m (4000 ft) in length. Cables of such lengths are susceptible to sudden voltage surges due to induced voltages such as indirect lightning strikes. Secondary surge protection circuitry is implemented on the EIA 422A adapter to protect it from these voltage surges. The surge protection circuitry is implemented on the adapter interface data lines.

Fail-safe circuitry has been added to the input leads of each EIA 422A receiver to prevent fault conditions when the receiver is not connected to a driver (open cable). The fail-safe circuitry sets the receiver to the mark state (binary 1) whenever the receiver is not connected to a driver.

The electrical characteristics of the 16-Port asynchronous EIA 422A adapter ports comply with the EIA Standard 422A dated December 1978.

ASCII, decimal, hexadecimal, octal, and binary conversion table

Helpful information for converting ASCII, decimal, hexadecimal, octal, and binary values can be referenced in this table.

Table 121. Conversions between ASCII, decimal, hexadecimal, octal, and binary values

ASCII	Decimal	Hexadecimal	Octal	Binary
null	0	0	0	0
start of header	1	1	1	1
start of text	2	2	2	10
end of text	3	3	3	11
end of transmission	4	4	4	100
enquire	5	5	5	101
acknowledge	6	6	6	110
bell	7	7	7	111
backspace	8	8	10	1000
horizontal tab	9	9	11	1001
linefeed	10	A	12	1010
vertical tab	11	B	13	1011
form feed	12	C	14	1100
carriage return	13	D	15	1101
shift out	14	E	16	1110
shift in	15	F	17	1111
data link escape	16	10	20	10000
device control 1/Xon	17	11	21	10001
device control 2	18	12	22	10010
device control 3/Xoff	19	13	23	10011
device control 4	20	14	24	10100
negative acknowledge	21	15	25	10101
synchronous idle	22	16	26	10110
end of transmission block	23	17	27	10111
cancel	24	18	30	11000
end of medium	25	19	31	11001
end of file/ substitute	26	1A	32	11010
escape	27	1B	33	11011
file separator	28	1C	34	11100
group separator	29	1D	35	11101
record separator	30	1E	36	11110
unit separator	31	1F	37	11111
space	32	20	40	100000
!	33	21	41	100001
"	34	22	42	100010
#	35	23	43	100011
$	36	24	44	100100
%	37	25	45	100101
&	38	26	46	100110
'	39	27	47	100111
(	40	28	50	101000
)	41	29	51	101001
*	42	2A	52	101010
+	43	2B	53	101011

Table 121. Conversions between ASCII, decimal, hexadecimal, octal, and binary values (continued)

ASCII	Decimal	Hexadecimal	Octal	Binary
,	44	2C	54	101100
-	45	2D	55	101101
.	46	2E	56	101110
/	47	2F	57	101111
0	48	30	60	110000
1	49	31	61	110001
2	50	32	62	110010
3	51	33	63	110011
4	52	34	64	110100
5	53	35	65	110101
6	54	36	66	110110
7	55	37	67	110111
8	56	38	70	111000
9	57	39	71	111001
:	58	3A	72	111010
;	59	3B	73	111011
<	60	3C	74	111100
=	61	3D	75	111101
>	62	3E	76	111110
?	63	3F	77	111111
@	64	40	100	1000000
A	65	41	101	1000001
B	66	42	102	1000010
C	67	43	103	1000011
D	68	44	104	1000100
E	69	45	105	1000101
F	70	46	106	1000110
G	71	47	107	1000111
H	72	48	110	1001000
I	73	49	111	1001001
J	74	4A	112	1001010
K	75	4B	113	1001011
L	76	4C	114	1001100
M	77	4D	115	1001101
N	78	4E	116	1001110
O	79	4F	117	1001111
P	80	50	120	1010000
Q	81	51	121	1010001
R	82	52	122	1010010
S	83	53	123	1010011
T	84	54	124	1010100
U	85	55	125	1010101
V	86	56	126	1010110
W	87	57	127	1010111
X	88	58	130	1011000

Table 121. Conversions between ASCII, decimal, hexadecimal, octal, and binary values (continued)

ASCII	Decimal	Hexadecimal	Octal	Binary
Y	89	59	131	1011001
Z	90	5A	132	1011010
[	91	5B	133	1011011
\	92	5C	134	1011100
]	93	5D	135	1011101
^	94	5E	136	1011110
_	95	5F	137	1011111
`	96	60	140	1100000
a	97	61	141	1100001
b	98	62	142	1100010
c	99	63	143	1100011
d	100	64	144	1100100
e	101	65	145	1100101
f	102	66	146	1100110
g	103	67	147	1100111
h	104	68	150	1101000
i	105	69	151	1101001
j	106	6A	152	1101010
k	107	6B	153	1101011
l	108	6C	154	1101100
m	109	6D	155	1101101
n	110	6E	156	1101110
o	111	6F	157	1101111
p	112	70	160	1110000
q	113	71	161	1110001
r	114	72	162	1110010
s	115	73	163	1110011
t	116	74	164	1110100
u	117	75	165	1110101
v	118	76	166	1110110
w	119	77	167	1110111
x	120	78	170	1111000
y	121	79	171	1111001
z	122	7A	172	1111010
{	123	7B	173	1111011
|	124	7C	174	1111100
}	125	7D	175	1111101
~	126	7E	176	1111110
DEL	127	7F	177	1111111
	128	80	200	10000000
	129	81	201	10000001
	130	82	202	10000010
	131	83	203	10000011
	132	84	204	10000100
	133	85	205	10000101

Table 121. Conversions between ASCII, decimal, hexadecimal, octal, and binary values (continued)

ASCII	Decimal	Hexadecimal	Octal	Binary
	134	86	206	10000110
	135	87	207	10000111
	136	88	210	10001000
	137	89	211	10001001
	138	8A	212	10001010
	139	8B	213	10001011
	140	8C	214	10001100
	141	8D	215	10001101
	142	8E	216	10001110
	143	8F	217	10001111
	144	90	220	10010000
	145	91	221	10010001
	146	92	222	10010010
	147	93	223	10010011
	148	94	224	10010100
	149	95	225	10010101
	150	96	226	10010110
	151	97	227	10010111
	152	98	230	10011000
	153	99	231	10011001
	154	9A	232	10011010
	155	9B	233	10011011
	156	9C	234	10011100
	157	9D	235	10011101
	158	9E	236	10011110
	159	9F	237	10011111
	160	A0	240	10100000
	161	A1	241	10100001
	162	A2	242	10100010
	163	A3	243	10100011
	164	A4	244	10100100
	165	A5	245	10100101
	166	A6	246	10100110
	167	A7	247	10100111
	168	A8	250	10101000
	169	A9	251	10101001
	170	AA	252	10101010
	171	AB	253	10101011
	172	AC	254	10101100
	173	AD	255	10101101
	174	AE	256	10101110
	175	AF	257	10101111
	176	B0	260	10110000
	177	B1	261	10110001
	178	B2	262	10110010

Table 121. Conversions between ASCII, decimal, hexadecimal, octal, and binary values (continued)

ASCII	Decimal	Hexadecimal	Octal	Binary
	179	B3	263	10110011
	180	B4	264	10110100
	181	B5	265	10110101
	182	B6	266	10110110
	183	B7	267	10110111
	184	B8	270	10111000
	185	B9	271	10111001
	186	BA	272	10111010
	187	BB	273	10111011
	188	BC	274	10111100
	189	BD	275	10111101
	190	BE	276	10111110
	191	BF	277	10111111
	192	C0	300	11000000
	193	C1	301	11000001
	194	C2	302	11000010
	195	C3	303	11000011
	196	C4	304	11000100
	197	C5	305	11000101
	198	C6	306	11000110
	199	C7	307	11000111
	200	C8	310	11001000
	201	C9	311	11001001
	202	CA	312	11001010
	203	CB	313	11001011
	204	CC	314	11001100
	205	CD	315	11001101
	206	CE	316	11001110
	207	CF	317	11001111
	208	D0	320	11010000
	209	D1	321	11010001
	210	D2	322	11010010
	211	D3	323	11010011
	212	D4	324	11010100
	213	D5	325	11010101
	214	D6	326	11010110
	215	D7	327	11010111
	216	D8	330	11011000
	217	D9	331	11011001
	218	DA	332	11011010
	219	DB	333	11011011
	220	DC	334	11011100
	221	DD	335	11011101
	222	DE	336	11011110
	223	DF	337	11011111

Table 121. Conversions between ASCII, decimal, hexadecimal, octal, and binary values (continued)

ASCII	Decimal	Hexadecimal	Octal	Binary
	224	E0	340	11100000
	225	E1	341	11100001
	226	E2	342	11100010
	227	E3	343	11100011
	228	E4	344	11100100
	229	E5	345	11100101
	230	E6	346	11100110
	231	E7	347	11100111
	232	E8	350	11101000
	233	E9	351	11101001
	234	EA	352	11101010
	235	EB	353	11101011
	236	EC	354	11101100
	237	ED	355	11101101
	238	EE	356	11101110
	239	EF	357	11101111
	240	F0	360	11110000
	241	F1	361	11110001
	242	F2	362	11110010
	243	F3	363	11110011
	244	F4	364	11110100
	245	F5	365	11110101
	246	F6	366	11110110
	247	F7	367	11110111
	248	F8	370	11111000
	249	F9	371	11111001
	250	FA	372	11111010
	251	FB	373	11111011
	252	FC	374	11111100
	253	FD	375	11111101
	254	FE	376	11111110
	255	FF	377	11111111

uDAPL (user-level Direct Access Programming Library)

uDAPL (user Direct Access Programming Library) is a direct access framework to be run on transports that support direct data access like InfiniBand , RNIC etc.

The DAT Collaborative specifies the **uDAPL** API http://www.datcollaborative.org .

The uDAPL codebase from Open Fabrics is ported to AIX and is currently supported over GX++ HCA and 4X DDR Expansion card (CFFh) InfiniBand adapters.

uDAPL 1.2 version is supported on AIX 6.1 with 6100-06 and later. The **uDAPL** install image is shipped on the expansion pack as *udapl.rte*. This image ships the DAT header files, located under **/usr/include/dat**. The install image also ships two libraries, *libdat.a* and *libdapl.a*.

Applications include the DAT header files and link with the DAT library (*libdat.a* in **/usr/include/dat**). The DAT layer determines the appropriate underlying transport-specific libraries.

An AIX **uDAPL** Provider registers itself with the DAT registry using *dat.conf* entries. The */etc/dat.conf* file is shipped with default entries and the file has details on the format of the entry.

For debugging purposes, the **uDAPL** libraries support AIX system trace. **uDAPL** system trace hook ids include 5C3 (for DAPL events), 5C4 (for DAPL error events), 5C7 (for DAT events), and 5C8 (for DAT error events). The initial trace level can be modified using the environment variables *DAT_TRACE_LEVEL* and *DAPL_TRACE_LEVEL* which could take the numeric values ranging from 0 to 10. The number of events and amount of data traced increases with the level, with the key trace levels being

```
TRC_LVL_ERROR  = 1,
TRC_LVL_NORMAL = 3,
TRC_LVL_DETAIL = 7
```

Other standard AIX serviceability features, such as the AIX error log, can be useful for problem determination. And the serviceability features of the underlying transport layer, such as the ibstat command and InfiniBand component trace, are also helpful for diagnosing issues.

The DAT APIs return the standard return codes that can be decoded with the help of the */usr/include/dat/dat_error.h* file. The detailed explanation of the return codes is located in the **uDAPL** specification from the DAT Collaborative.

"Internet Protocol over InfiniBand (IPoIB)" on page 369

uDAPL APIs supported in AIX

Out of the many **uDAPL** APIs specified by the DAT Collaborative, there are a few APIs that are not supported in AIX.

The following are the APIs that common industry **uDAPL** implementations do not support, and will not be supported on AIX either.

Item	Description
dat_cr_handoff	// In DAT 1.2
dat_ep_create_with_srq	// In DAT 1.2
dat_ep_recv_query	// In DAT 1.2
dat_ep_set_watermark	// In DAT 1.2
dat_srq_create	// In DAT 1.2
dat_srq_post_recv	// In DAT 1.2
dat_srq_resize	// In DAT 1.2
dat_srq_set_lw	// In DAT 1.2
dat_srq_free	// In DAT 1.2
dat_srq_query	// In DAT 1.2

Additional APIs that AIX does not support are,

- dat_lmr_sync_rdma_read
- dat_lmr_sync_rdma_write
- dat_registry_add_provider
- dat_registry_add_provider

For all unsupported APIs, AIX follows the specific mechanisms described in the DAT spec to identify its lack of support. These include attribute values (such as max_srq equaling zero) and specific return codes (such as DAT_MODEL_NOT_SUPPORTED). Consistent with industry implementation and the DAT specification, DAT_NOT_IMPLEMENTED may also be returned for a function which is not supported.

Support of RMR related APIs such as *dat_rmr_create ,dat_rmr_bind ,dat_rmr_free* and *dat_rmr_query* is dependent on the underlying HCA capability, and success or failure is determined by the underlying IB framework. Currently the GX++ HCA and 4X DDR Expansion card (CFFh) InfiniBand adapters do not support these RMR operations.

"uDAPL (user-level Direct Access Programming Library)" on page 643

"Vendor-specific attributes for uDAPL"

"Internet Protocol over InfiniBand (IPoIB)" on page 369

Vendor-specific attributes for uDAPL

There are a few vendor-specific attributes supported in AIX. The attribute names are **delayed_ack_supported** , **vendor_extension**, **vendor_ext_version**, **debug_query** and **debug_modify**.

delayed_ack_supported

The AIX provider for the InfiniBand (IB) transport includes a vendor-specific interface adapter (IA) attribute named **delayed_ack_supported**. The value of this attribute is either **true** or **false**. When **true**, endpoints associated with this IA® have a modifiable provider-specific attribute named **delayed_ack**. When the **delayed_ack_supported** attribute is **false**, an endpoint's **delayed_ack** provider-specific attribute cannot be modified. The default value of an endpoint's **delayed_ack** attribute is **false**. Setting it to **true** (via dat_ep_modify) enables the delayed ack feature of the underlying IB Host Channel Adapter (HCA) for the specific IB queue pair associated with the endpoint. This hardware feature is not implemented by all HCAs, and is therefore not available for all IAs. Enabling the feature causes the HCA to delay sending acknowledgement until a data transfer operation is visible in a server's system memory. This is a slightly stronger semantic than what is provided for in the IB spec, at the potential cost of a very small latency increase.

vendor_extension, vendor_ext_version, debug_query and debug_modify

For debugging purposes, the **uDAPL** libraries support AIX system trace. The initial trace level can be modified using the environment variables *DAT_TRACE_LEVEL* and *DAPL_TRACE_LEVEL*. To change these trace levels dynamically via API, we provide dynamic trace level support on AIX. To verify if the library has dynamic trace level support, applications can query for the vendor-specific IA attribute, **vendor_extension**. On return from query, the presence of the **vendor_extension** attribute indicates dynamic trace level support. The value of the attribute is set to **true**; but irrespective of that, presence of the attribute indicates the support. When **vendor_extension** attribute is present, applications can get the function pointers to **dat_trclvl_query()** and **dat_trclvl_modify()** by querying for the vendor-specific IA attributes, **debug_query** and **debug_modify**. The value of these attributes will have the pointer to the corresponding functions. To make this **vendor_extension** interface extensible for future usage, we have another vendor-specific IA attribute, **vendor_ext_version**. Since we will support only one version right now, the value of this attribute will be set to **1.0**. If the **vendor_extension** attribute does not exist, applications cannot modify the trace levels dynamically.

An example of how to manipulate these attributes is included in the **uDAPL** sample code installed with the AIX implementation.

"uDAPL (user-level Direct Access Programming Library)" on page 643

"Internet Protocol over InfiniBand (IPoIB)" on page 369

PCIe2 10 GbE RoCE Adapter support

The PCIe2 10GbE RDMA Over Converged Ethernet (RoCE) Adapter was first supported in the AIX operating system as a remote direct memory access (RDMA) capable device only. The software to support it was an IBM proprietary software based on the AIX InfiniBand stack. This support was called AIX RoCE. AIX 7 with 7100-02 or later supports that adapter in two modes, which are the AIX RoCE and the 10G Ethernet support also called the network interface card (AIX NIC). The new AIX 7 with 7100-03 now supports RDMA along with the NIC mode and OpenFabrics Enterprise Distribution (OFED). The host bus adapter (HBA), which was not available in earlier versions of the AIX operating systems, manages which mode is enabled.

The following table shows the evolution of the PCIe2 10GbE Adapter software:

AIX Level	MODE 1	MODE 2
Before AIX 7 with 7100-02	AIX RoCE	NA
AIX 7 with 7100-02	AIX RoCE	AIX NIC
AIX 7 with 7100-03	AIX RoCE	AIX NIC + OFED RoCE

To download the latest device driver for this adapter complete the following steps:

1. Go to the IBM website (www.ibm.com)
2. Click **Support and downloads**.
3. Download the latest firmware to the AIX host location (`/etc/microcode`)
4. Run the **diag** tool to update the firmware by choosing one of the following procedures:
   - Short path procedure
     a. Enter the following command:
        ```
        *diag -d entX -T download
        ```
        Note: Replace **entX** with **roceX** if you are using the RoCE stack from a previous version.
     b. Select the microcode that is saved in the `/etc/microcode` directory.
   - Long path procedure
     a. Enter the following command:
        ```
        *diag
        ```
     b. Click: **Task selection** > **Microcode tasks** > **Download microcode**.
     c. Select **entX** or **roceX**.
     d. Select the microcode that is saved in the `/etc/microcode` directory.

By default the adapter is configured to support the AIX RoCE mode. Complete the steps in the "AIX NIC + OFED RDMA" section to change it to the other mode.

AIX NIC + OFED RDMA

As of AIX 7 with 7100-02 the PCIe2 10 GbE RoCE Adapter can be configured to run in the AIX NIC configuration. As of AIX 7 with 7100-03, the OFED RDMA functionality was also added to the AIX NIC configuration. If you do not have the network-intensive applications that benefit from RDMA, then you can use the adapter as a Network Interface Card (NIC) only.

To use the PCIe2 10 GbE RoCE Adapter in the AIX NIC + OFED RoCE configuration or AIX RoCE configuration, the following file sets are required and are available on the AIX 7 with 7100-03 base operating system CD.

devices.ethernet.mlx
Converged Ethernet Adapter main device driver (mlxentdd) to support the AIX NIC + OFED RoCE configuration.

devices.pciex.b315506b3157265
Packaging support for the NGP ITE Converge Ethernet Adapter ASIC2.

devices.pciex.b3155067b3157365
Packaging support for the NGP ITE Converge Ethernet Adapter ASIC1.

devices.pciex.b315506714101604
Packaging for Mellanox 2 Ports 10 GbE Converge Ethernet Adapter with the small form factor pluggable (SFP+) transceivers.

devices.pciex.b315506714106104
Packaging for Mellanox 2 Ports 10 GbE Converge Ethernet Adapter that supports any SFP+ transceivers.

devices.common.IBM.ib
ICM device driver that is required to use the AIX RoCE configuration.

devices.pciex.b3154a63
Mellanox 10 GbE Converge Ethernet Adapter device driver that is required to use the AIX RoCE configuration.

ofed.core
OFED Core Runtime Environment file set that is needed only if OFED RDMA is required.

After the existing AIX RoCE file sets are updated with the new file sets, both the roce and the ent devices might appear to be configured. If both devices appear to be configured when you run the **lsdev** command on the adapters, complete the following steps:

1. Delete the *roceX* instances that are related to the PCIe2 10 GbE RoCE Adapter by entering the following command:
   ```
   # rmdev -dl roce0[, roce1][, roce2,...]
   ```
2. Delete the *entX* instances that are related to the PCIe2 10 GbE RoCE Adapter by entering the following command:
   ```
   # rmdev -dl ent1[,ent2][, ent3...]
   ```
3. If there are one or more converged host bus adapters (hbaX) that are related to the PCIe2 10 GbE RoCE Adapter, delete them by entering the following command:
   ```
   # rmdev -dl hba0[, hba1][,hba2...]
   ```
4. Run the configuration manager to incorporate the changes by entering the following command:
   ```
   # cfgmgr
   ```

Complete the following steps to switch over to the AIX NIC + OFED RoCE configuration from the AIX RoCE configuration:

1. Stop all RDMA applications that are running on the PCIe2 10 GbE RoCE Adapter.
2. Delete or redefine the *roceX* instances by entering one of the following commands:
   - `# rmdev -d -l roce0`
   - `# rmdev -l roce0`

   The `rmdev -l roce0` command retains the definition of the `roce0` configuration so you can use it the next time to create instances.
3. Change the attribute of the hba stack_type setting from `aix_ib` (AIX RoCE) to `ofed` (AIX NIC + OFED RoCE) by entering the following command:
   ```
   # chdev -l hba0 -a stack_type=ofed
   ```
4. Run the configuration manager tool so that the host bus adapter can configure the PCIe2 10 GbE RoCE Adapter as a NIC adapter by entering the following command:
   ```
   # cfgmgr
   ```
5. Verify that the adapter is now running in NIC configuration by entering the following command:
   ```
   # lsdev -C -c adapter
   ```

The following example shows the results when you run the **lsdev** command on the adapter when it is configured in AIX NIC + OFED RoCE mode:

```
ent1 Available 00-00-01 PCIe2 10GbE RoCE Converged Network Adapter
ent2 Avaliable 00-00-02 PCIe2 10GbE RoCE Converged Network Adapter
hba0 Available 00-00 PCIe2 10GbE RoCE Converged Host Bus Adapter (b315506714101604)
```

Figure 45. Example output of **lsdev** *command on an adapter with the AIX NIC + OFED RoCE configuration*

Because as of AIX 7 with 7100-03, AIX also supports OFED RDMA in the AIX NIC mode, if OFED RDMA needs to be enabled, you need to complete the following two additional steps:

1. Install the package ofed.core.
2. Set the RDMA mode in the ent1, ent2 devices by entering the following command:

   ```
   # chdev –l ent1 –a rdma=desired
   # chdev –l ent2 -a rdma=desired
   ```

   The RDMA mode is set before the en1 or en2 interfaces are configured.
3. You can disable the RDMA mode by entering the following command:

   ```
   # chdev –l ent1 –a rdma=disabled
   # chdev –l ent2 –a rdma=disabled
   ```

AIX RoCE

The PCIe2 10 GbE RoCE Adapter is preconfigured to operate in the AIX RoCE mode. A network that uses RDMA provides better performance than an adapter that is used as a NIC for network-intensive applications. This mode is often helpful for network storage or high-performance computing.

AIX RoCE configuration requires the use of libraries or interfaces such as the following:

- The Direct Access Programming Library (uDAPL), which is used by the DB2® database system
- The Message Passing Interface (MPI), which is used by high performance computing (HPC)

Figure 46 on page 649 displays the output when the adapter is running in the AIX RoCE mode.

The PCIe2 10 GbE RoCE Adapter shows only one adapter instance when it is in the AIX RoCE mode, but it can have up to two ports. Use the **ibstat** command to determine how many ports are configured by completing the following steps:

1. Determine whether the `icm` kernel extension is configured by entering the following command:

   ```
   # lsdev -C | grep icm
   ```
2. If the icm kernel is not configured, configure it by entering the following command:

   ```
   # mkdev -c management -s infiniband -t icm
   ```
3. Run the **ibstat** command by entering the following command:

   ```
   # ibstat roce0
   ```

While the PCIe2 10 GbE RoCE Adapter is initially configured to use the AIX RoCE mode, you might need to switch back from the AIX NIC + OFED RoCE configuration. To switch over from the AIX NIC + OFED RoCE configuration to the AIX RoCE configuration, complete the following steps:

1. Verify that the adapter is in the AIX NIC + OFED RoCE mode by entering the following command:

   ```
   # lsdev -C -c adapter
   ```

   The output of the **lsdev** command is similar to the example in Figure 45.
2. Stop the TCP/IP traffic and detach the IP interfaces by entering the following command:

   ```
   # ifconfig en1 down detach; ifconfig en2 down detach
   ```
3. Delete or put the NIC instances in a defined state by entering one of the following commands:

- `# rmdev -d -l ent1; rmdev -d -l ent2`
- `# rmdev -l ent1; rmdev -l ent2`

The `rmdev -l ent1; rmdev -l ent2` command retains the definition of the Ethernet devices so that you can use it the next time that you create instances.

4. Change the attribute of the hba stack_type from `ofed` (AIX NIC + OFED RoCE) to `aix_ib` (AIX RoCE) by entering the following command:
   ```
   # chdev -l hba0 -a stack_type=aix_ib
   ```
5. Run the configuration manager tool so that the host bus adapter can configure the PCIe2 10 GbE RoCE Adapter as an AIX RoCE adapter by entering the following command:
   ```
   # cfgmgr
   ```
6. Verify that the adapter is now running in the AIX RoCE configuration by entering the following command:
   ```
   # lsdev -C -c adapter
   ```

   The following example shows the results when you run the **lsdev** command for adapters, and the adapter is configured in the AIX RoCE mode.

```
roce0 Avaliable 00-00-00 PCIe2 10GbE RoCE Converged Network Adapter
hba0 Available  00-00-00 PCIe2 10GbE RoCE Converged Host Bus Adapter (b315506714101604)
```

Figure 46. Example output of the **lsdev** *command for the adapters when it is using the AIX RoCE configuration*

PCIe3 40 GbE RoCE Adapter support

The PCIe3 40 GbE RDMA Over Converged Ethernet (RoCE) Adapter supports remote direct memory access (RDMA) with the OpenFabrics Enterprise Distribution (OFED) in the normal NIC mode. RDMA is supported and enabled by default if the OpenFabrics software is installed.

To download the latest device driver for this adapter, complete the following steps:

1. Go to the IBM website (www.ibm.com).
2. Click **Support and downloads**.
3. Download the latest firmware to the AIX host location (`/etc/microcode`).
4. Run the **diag** tool to update the firmware by choosing one of the following procedures:
   - Short path procedure
     a. Enter the following command:
        ```
        *diag -d entX -T download
        ```

        Note: If the Ethernet device belongs to the same host bus adapter (for example, hba0, hba1, and so on), download the firmware to one of the **ent** devices.
     b. Select the microcode that is saved in the `/etc/microcode` directory.
   - Long path procedure
     a. Enter the following command:
        ```
        *diag
        ```
     b. Click **Task selection** > **Microcode tasks** > **Download microcode**.
     c. Select **entX**.
     d. Select the microcode that is saved in the `/etc/microcode` directory.

To use the PCIe3 40 GbE RoCE Adapter and the AIX NIC + OFED RoCE, the following filesets are required. These filesets are available n the AIX 7 with 7100-03 base operating system CD.

devices.ethernet.mlx	Converged Ethernet Adapter main device driver (mlxentdd) to support the AIX NIC + OFED RoCE configuration.
devices.pciex.b31503101410b504	Packaging for Mellanox 2 Ports 40 Gb Converged Ethernet Adapter that uses passive copper Quad Small Form-factor Pluggable (QSFP) ports.
ofed.core	OFED Core Runtime Environment fileset that is needed only if OFED RDMA functionality is required.

To disable the RDMA functionality, enter the following command:

```
chdev  –l  <Ethernet_device> rdma=disabled
```

Example:

```
# chdev –l ent1 –a rdma=disabled
# chdev –l ent2 –a rdma=disabled
```

To enable the RDMA functionality, enter the following command:

```
chdev –l <Ethernet_device> rdma=desired
```

Notices

This information was developed for products and services offered in the US.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

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 Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as follows:

© (your company name) (year).

Portions of this code are derived from IBM Corp. Sample Programs.

© Copyright IBM Corp. _enter the year or years_.

Privacy policy considerations

IBM Software products, including software as a service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as the customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM’s Privacy Policy at http://www.ibm.com/privacy and IBM’s Online Privacy Statement at http://www.ibm.com/privacy/details the section entitled “Cookies, Web Beacons and Other Technologies” and the “IBM Software Products and Software-as-a-Service Privacy Statement” at http://www.ibm.com/software/info/product-privacy.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at Copyright and trademark information at www.ibm.com/legal/copytrade.shtml.

INFINIBAND, InfiniBand Trade Association, and the INFINIBAND design marks are trademarks and/or service marks of the INFINIBAND Trade Association.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Index

Special characters

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Printed in USA