

Interface Program for use with TCP/IP

Programming Family



Personal
Computer
Software

Interface Program for use with TCP/IP

Programming Family



Personal
Computer
Software

First Edition (January 1987)

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

This edition applies to Version 2.1 of IBM RT PC Interface Program for use with TCP/IP, and to all subsequent releases until otherwise indicated in new editions or technical newsletters. Changes are made periodically to the information herein; these changes will be reported in technical newsletters or in new editions of this publication.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent program may be used instead.

International Business Machines Corporation provides this manual "as is," without warranty of any kind, either express or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this manual at any time.

Products are not stocked at the address given below. Requests for copies of this product and for technical information about the system should be made to your authorized IBM RT PC dealer or your IBM marketing representative.

A reader's comment form is provided at the back of this publication. If the form has been removed, address comments to IBM Corporation, Department 997, 11400 Burnet Road, Austin, Texas 78758-3493. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Permission to use, copy, modify, and distribute this software only for the purposes and only in the manner set forth in the appertaining agreement is hereby granted, provided that this copyright and permission notice appear on all copies and supporting documentation. Further, the name of M.I.T. is not to be used in any advertising or publicity pertaining to distribution of the software without specific prior permission.

© Copyright International Business Machines Corporation 1985, 1987

© Copyright Massachusetts Institute of Technology 1984

© Copyright Paul G. Milazzo 1985, all rights reserved

About This Book

This book provides information about the IBM RT PC Interface Program for use with TCP/IP (Transmission Control Protocol/Internet Protocol). With the Interface Program for use with TCP/IP, you can communicate with:

- Other IBM RT Personal Computer¹ systems that also have the Interface Program installed
- Other host systems that support TCP/IP.

Note: In this book, the term Interface Program refers to the IBM RT PC Interface Program for use with TCP/IP.

Who Should Use This Book

This book is intended for Interface Program users, system managers, and communications programmers. The major tasks covered in this book are:

- Transferring data between an IBM RT PC and another host computer
- Using another IBM RT PC or host, or its facilities, remotely
- Managing networks.

For programmers, this book also describes the Application Programming Interfaces (APIs) for the Interface Program, including the library routines and the `/dev/net0` device driver.

Readers require a thorough understanding of the IBM RT PC functions. In addition, programmers will find knowledge of the AIX¹ Operating System and the Virtual Resource Manager (VRM) helpful.

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

Before You Begin

Before you can use the Interface Program, you must have installed the following software and hardware on each IBM RT PC that is to be on the network:

Software

- IBM RT PC Virtual Resource Manager Licensed Program, Version 2.1
- IBM RT PC AIX Operating System Licensed Program, Version 2.1
- VRM Baseband Adapter Device Driver, VRM Token-Ring Device Driver, or both
- IBM RT PC Interface Program for use with TCP/IP.

Hardware

- IBM RT PC Baseband Adapter for use with Ethernet, IBM Token-Ring Network RT PC Adapter, or both
- Cables and connectors.

Refer to the following publications for information about installing the software and hardware:

- *IBM RT PC Installing the Virtual Resource Manager*
- *IBM RT PC Installing and Customizing the AIX Operating System*
- *IBM RT PC Options Installation*
- *IBM RT PC User Setup Guide.*

After you install the necessary software and hardware, see Appendix A, “Customizing the Program” for information about creating an Interface Program network to meet your requirements.

How to Use This Book

Once you have installed and customized the Interface Program, you should become familiar with the information in Chapter 1, “General Information” and Chapter 2, “User Commands.” With the information in these two chapters, you should be able to use the Interface Program commands to transfer data, log in remotely, and manage the network. If you want to use the Application Programming Interfaces (APIs) of the Interface Program, you also should become familiar with the information in Chapter 4, “Protocol Library Routines,” Chapter 5, “`/dev/net0` Device Driver,” and Appendix B, “Samples.”

Organization

This book is divided into the following chapters:

- Chapter 1, “General Information” provides an overview of the Interface Program.
- Chapter 2, “User Commands” describes Interface Program user commands.
- Chapter 3, “Server Commands” describes the Interface Program server commands, or daemons.
- Chapter 4, “Protocol Library Routines” describes the main Interface Program libraries.
- Chapter 5, “`/dev/net0` Device Driver” describes the interface to the `net` driver.

The following appendixes contain supplemental information:

- Appendix A, “Customizing the Program” explains how to adapt the Interface Program to your requirements after you install it.
- Appendix B, “Samples” provides programming examples for the library routines and tasking system.
- Appendix C, “`tcp` Library Routines” describes a supplemental Interface Program library.

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

Typography

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

New terms

The first occurrence of each new term is printed in this style.

System parts

Names of commands, files, and other parts of the system are printed in this style.

Variable information

Names for information you must provide are printed in this style.

Information you are to type

To run the examples in this book, enter the information printed in this style.

Related Publications

The following documents in the IBM RT PC series contain additional information that may prove helpful in understanding and using the Interface Program:

- *IBM RT PC AIX Operating System Technical Reference.*
- *IBM RT PC Virtual Resource Manager Technical Reference.*

See *IBM RT PC Bibliography and Master Index* for order numbers of IBM RT PC publications and diskettes.

For general information about TCP/IP, the following publications are recommended. These publications are distributed by the Network Information Center on behalf of the Defense Communications Agency and Defense Advanced Research Projects Agency (DARPA). The mailing address is:

Network Information Center
SRI International
Menlo Park, CA 92025

- *An Ethernet² Address Resolution Protocol*, RFC 826, D. Plummer
- *Assigned Numbers*, RFC990, J. Reynolds, J. Postel
- *Broadcasting Internet Datagrams*, RFC 919, J. Mogul
- *File Transfer Protocol*, RFC959, J. Postel
- *Internet Control Message Protocol*, RFC792, J. Postel
- *Internet Name Server Protocol*, IEN116, J. Postel
- *Internet Protocol*, RFC791, J. Postel
- *Internet Standard Subnetting Procedure*, RFC 950, J. Mogul
- *Name/Finger*, RFC742, K. Harrenstien
- *Official ARPA-Internet Protocols*, RFC 944, J. Reynolds, J. Postel
- *Simple Mail Transfer Protocol*, RFC821, J. Postel
- *Telnet Protocol Specification*, RFC854, J. Postel, J. Reynolds
- *The TFTP Protocol*, RFC783, K. R. Sollins
- *Time Protocol*, RFC 868, J. Postel., K. Harrenstien
- *Trailer Encapsulations*, RFC 893, S. Leffler, M. Karels

² Ethernet is a registered trademark of Xerox Corporation.

-
- *Transmission Control Protocol*, RFC793, J. Postel
 - *Trivial File Transfer Protocol*, RFC783, K. R. Sollins
 - *User Datagram Protocol*, RFC768, J. Postel

For additional information about the IBM Token-Ring Network, you may want to order the *IBM Token-Ring Architecture Reference* (Order Number 6165877) from your IBM representative.

Ordering Additional Copies of This Book

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

- To order from your IBM representative, use Order Number SBOF-0148.
- To order from your IBM dealer, use Part Number 79X3893.

A binder and the *IBM RT PC Interface Program for use with TCP/IP* manual are included with the order. For information on ordering the binder and/or manuals separately, contact your IBM representative or your IBM dealer.

Contents

Chapter 1. General Information	1-1
About This Chapter	1-2
Overview	1-3
Protocols	1-6
Internet Router	1-10
Commands	1-13
Addresses and Names	1-15
File Formats	1-22
Assigned Numbers	1-38
Security	1-39
Program Customization	1-41
Chapter 2. User Commands	2-1
Chapter 3. Server Commands	3-1
Chapter 4. Protocol Library Routines	4-1
Chapter 5. /dev/net0 Device Driver	5-1
Appendix A. Customizing the Program	A-1
Appendix B. Samples	B-1
Appendix C. tcp Library Routines	C-1
Figures	X-1
Glossary	X-3
Index	X-7

Chapter 1. General Information

CONTENTS

About This Chapter	1-2
Overview	1-3
The Internet Environment	1-4
Structure of the Interface Program	1-4
Protocols	1-6
Internet Protocol (IP)	1-6
Address Resolution Protocol (ARP)	1-7
Transmission Control Protocol (TCP)	1-7
User Datagram Protocol (UDP)	1-7
Other Network Protocols	1-8
Internet Router	1-10
Commands	1-13
File Transfer	1-13
Mail	1-14
Remote Login, Command Execution, and Printing	1-14
Network Management	1-15
Addresses and Names	1-15
IP Addressing	1-15
TCP Addressing	1-20
Sub-Networks	1-21
Broadcast Messages	1-22
File Formats	1-22
gateways	1-23
hosts	1-25
hosts.equiv	1-28
net	1-29
networks	1-31
rc.tcpip	1-33
.netrc	1-34
.3270keys	1-36
Assigned Numbers	1-38
Port Numbers	1-38
Version Numbers	1-39
Protocol Numbers	1-39
Security	1-39
Security Features	1-40
Data Security and Information Protection	1-41
Program Customization	1-41

About This Chapter

The IBM RT PC Interface Program for use with TCP/IP consists primarily of protocols and commands (application programs) that enable the RT PC system to communicate with:

- Other RT PC systems with the Interface Program installed
- Host systems that support TCP/IP.

This chapter provides an overview of the Interface Program protocols and commands, and also explains the other features and conventions of the program.

Overview

A Brief Look at the Interface Program for use with TCP/IP

The Interface Program for use with TCP/IP includes commands and facilities that allow users to:

- Transfer files across the network
- Send and receive mail across the network
- Log in to remote systems
- Run commands on remote systems
- Print files on remote systems
- Manage an Interface Program network.

This book contains information about creating an Interface Program network and using the Interface Program commands. Also, for those who wish to develop programs using components of the Interface Program, this book describes the following Application Programming Interfaces:

- Protocol library routines for Transmission Control Protocol (TCP), User Datagram Protocol (UDP), and Internet Protocol (IP)
- The `/dev/net0` device driver.

Before continuing, you may find it useful to become familiar with the following terms as they are used in this book.

client A computer that is accessing the data or resources of another computer attached to the network.

host A computer that is attached to the network. The **local host** for a particular user is the computer at which that user is working. A **foreign host** is any other host on the network. From the point of view of the communication network, hosts are the sources and destinations of packets. Any host can be a client, a server, or both. On an Interface Program network, a host is identified by its Internet address.

packet The data of one transaction between a host and its network. Packets are the exchange medium used by processes to send and receive data through the network.

process A program that is running. A process is the active element in a host computer. Terminals, files, and other I/O devices communicate with each other through processes. Thus, network communication is interprocess communication.

server A computer that contains the data or resources that can be accessed by other computers attached to the network.

The Internet Environment

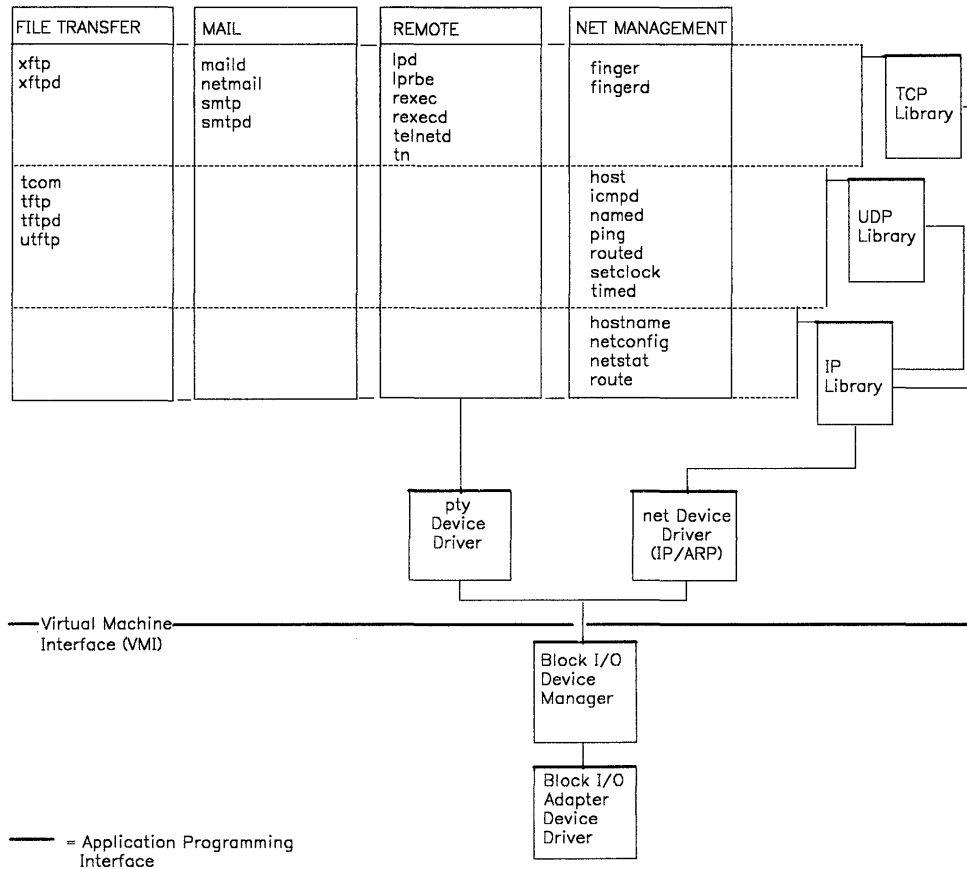
The Interface Program network protocols include the Internet Protocol (IP) transport layer and higher level protocols that use the Internet address format. These protocols use the AIX operating system on the RT PC system. The Interface Program provides facilities that make the RT PC system a host that attaches directly to a network.

The Internet environment consists of hosts connected to networks that use packet switching technology (for example, networks that use the IBM RT PC Baseband Adapter for use with Ethernet). Networks, in turn, can be interconnected via *gateways*. Host processes originate and receive network packets. Protocols at different levels in the networks, gateways, and hosts support interprocess communication, providing two-way data flow on logical connections.

Data is transmitted between process *ports*. Each process may have a number of ports through which it communicates with other processes. Each port provides queues for sending and receiving data. A process may have logical connections to several other processes and, if necessary, can treat each of the connected processes independently.

Structure of the Interface Program

Figure 1-1 on page 1-5 shows the principal Interface Program commands, protocols, and Application Programming Interfaces (APIs).



OLR20025

Figure 1-1. Interface Program for use with TCP/IP Commands, Protocols, and APIs

Note: This publication documents the API for TCP, UDP, IP, and the `/dev/net0` device driver. The API for the PTY device driver is described in *IBM RT PC AIX Operating System Technical Reference*. The API for the Block I/O Device Manager and for the VRM Baseband Adapter Device Driver are described in *IBM RT PC Virtual Resource Manager Technical Reference*.

Protocols

The Interface Program provides the following protocols:

- Internet Protocol (IP)
- Address Resolution Protocol (ARP)
- Transmission Control Protocol (TCP)
- User Datagram Protocol (UDP)
- Other network protocols:
 - Gateway-to-Gateway Protocol (GGP)
 - Internet Control Message Protocol (ICMP).
 - VAX¹ trailer encapsulation protocol
 - Routing Information Protocol (RIP)
 - Remote printing protocol
 - Remote command execution protocol

Following are brief explanations of these protocols.

Internet Protocol (IP)

Internet Protocol (IP) provides the interface from the higher level host-to-host protocols to the local network protocols. Addressing at this level is usually from host to host.

IP is used by host-to-host protocols in an Interface Program environment as a basic transport mechanism. IP uses local area network protocols to carry packets (or datagrams) to the next gateway or destination host.

The IP is designed for use in interconnected systems of packet-switched computer communication networks. The network connecting hosts are called *gateways*. IP provides the means to transmit blocks of data (or a package of bits) from sources to destinations. Sources and destinations are hosts identified by fixed length addresses. Outgoing packets automatically have an IP header prefixed to them and incoming packets have their IP header removed before being sent to the user. This protocol provides the universal addressing of hosts in the network.

IP, however, does not provide a reliable communication facility because it does not provide acknowledgements either from the sending host, the receiving host, or intermediate hosts. IP also does not provide error control for data. It provides only a header checksum. IP

¹ VAX is a trademark of Digital Equipment Corporation.

treats each datagram as an independent entity unrelated to any other datagram. It does not perform retransmissions or flow control. A higher level protocol that uses IP must implement its own reliability procedures if it requires reliable communications. See “internet” on page 4-19 for the programming interface.

Address Resolution Protocol (ARP)

Address Resolution Protocol (ARP) is a protocol that dynamically maps between Internet addresses and Baseband Adapter and Token-Ring Adapter addresses on a local area network. It is used by the Baseband Adapter and Token-Ring Adapter interface drivers and is not directly available to users. Address Resolution Protocol (ARP) allows dynamic distribution of the information needed to build mapping tables. These mapping tables map Internet addresses into Baseband Adapter or Token-Ring Adapter addresses. This protocol provides an interface for IP to the hardware by defining the standard format of the packet interface to the Baseband Adapter or Token-Ring Adapter.

ARP caches Internet to Baseband Adapter and Token-Ring Adapter address mappings. When an interface requests a mapping for an address not in the cache, ARP sends the ARP request packet (broadcast) on the network requesting the address mapping. If a response is provided, a new mapping is cached, and any pending Internet packets are transmitted.

Transmission Control Protocol (TCP)

Transmission Control Protocol (TCP) is used in the Advanced Research Projects Agency (ARPA) Internet and any network following the U.S. Department of Defense standards for inter-network protocols. This protocol provides a reliable host-to-host protocol between hosts in packet-switched computer communication networks, and in interconnected systems of such networks. TCP assumes that the Internet Protocol (IP) is the underlying protocol.

The interface to TCP consists of a set of library calls similar to the calls an operating system provides to an application process in order to manipulate files. TCP communicates asynchronously with application programs. TCP operates in a very general environment of interconnected networks. It supports the transmission of 8-bit bytes. See “tcp” on page C-2 for the TCP programming interface.

User Datagram Protocol (UDP)

User Datagram Protocol (UDP) allows a datagram mode of packet-switched communication in the environment of an interconnected set of computer networks. This protocol assumes that the IP is the underlying protocol. UDP provides a procedure for application programs to send messages to other programs with a minimum of protocol mechanism. The protocol is transaction oriented. It offers no guarantee of delivery and duplicate protection. Applications that require reliable delivery of streams of data should use the Transmission Control Protocol.

The interface to UDP consists of a set of library calls. UDP must be able to determine the source and destination Internet addresses along with the protocol field from the Internet header. The major users of this protocol are the **host** command (in resolving network

names and addresses), the **setclock** command (in providing the network time service), and the Trivial File Transfer Protocol (TFTP) command. See “**udp**” on page 4-14 for the API to UDP.

Other Network Protocols

In addition to the protocols for transmitting user data, two other protocols are used to transmit network monitoring and management data:

- Gateway-to-Gateway Protocol (GGP)
- Internet Control Message Protocol (ICMP).

There is no programming interface for either of these protocols; they are imbedded in the **ping** and **icmpd** commands and in the kernel.

Gateway-to-Gateway Protocol (GGP)

A gateway uses Gateway-to-Gateway Protocol (GGP) to determine connectivity to networks and neighbor gateways. GGP is also used to implement the shortest-path routing algorithm. The gateway sends routing information in GGP routing update messages. The gateway receives and transmits routing information reliably using sequence-numbered messages and a transmission and acknowledge scheme. GGP periodically sends GGP echoes to each neighbor of the gateway to determine neighbor connectivity and sends interface status messages addressed to itself to determine network connectivity. GGP also sends updated routing information when necessary.

Internet Control Message Protocol (ICMP)

Although gateways communicate between themselves for control purposes via a Gateway-to-Gateway Protocol (GGP), occasionally a gateway or destination host communicates with a source host, to report an error in datagram processing for example. Internet Control Message Protocol (ICMP) is used for this purpose. 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. See “**icmpd**” on page 3-5 for related information.

ICMP messages are sent in several situations, for example:

- When a datagram cannot reach its destination
- When the gateway does not have the buffering capacity to forward a datagram
- When the gateway can direct the host to send traffic on a shorter route.

In addition, the user can generate the following messages:

- **timestamp** (to measure the round trip time)
- **echo** (to determine whether the foreign host is available)
- **subnet mask** (to indicate whether a host supports sub-networks).

The purpose of control messages is to provide feedback about problems in the communication environment, not to make IP more reliable. There is no guarantee that a datagram will be delivered or that a control message will be returned.

VAX Trailer Encapsulation Protocol

VAX Trailer Encapsulation Protocol (the protocol that supports VAX trailers) moves all variable length header information in a packet to a position following the data segment. Trailer encapsulation allows the receiving host to receive data on a page-aligned boundary, a requirement for exploiting a page-mapped virtual memory environment. The RT PC receives and processes VAX trailer protocol data, but does not transmit it.

Note: If VAX trailers are to be transmitted, all hosts sharing the network environment must accept them uniformly.

Routing Information Protocol

The routing daemon (**routed**) uses a variant of the Xerox NS Routing Information Protocol to maintain current kernel routing table entries. For related information, see “**routed**” on page 3-17.

Remote Printing Protocol

The **lpd** command provides the remote printing protocol. For information about **lpd** and the remote printing protocol, see “**lpd**” on page 3-7.

Remote Command Execution Protocol

The **rexecd** command provides the remote command execution protocol. For information about **rexecd** and the remote command execution protocol, see “**rexecd**” on page 3-15.

Internet Router

The Internet Router enables an Interface Program host to act as a gateway for routing data between separate networks that use either of the following:

- IBM RT PC Baseband Adapter for use with Ethernet
- IBM Token-Ring Network RT PC Adapter.

Up to two adapters of each type (Baseband Adapter and Token-Ring Adapter) can be installed in the RT PC. The adapters (or interfaces) are defined by adding one entry for each adapter to each of the following files:

`/etc/net`
`/etc/hosts`

Each host has one primary name but, because of the entries in `/etc/hosts`, can have multiple secondary names, which are used in routing. References to gateway hosts can be by primary or secondary name.

Figure 1-2 on page 1-11 represents four networks and two gateway hosts, and it demonstrates how routing can be designed. The networks are:

200 (host2, host4, host5, and host6)
201 (host1, host2, and host3)
202 (host2, host8, host10, host11, and host12)
203 (host7, host8, and host9)

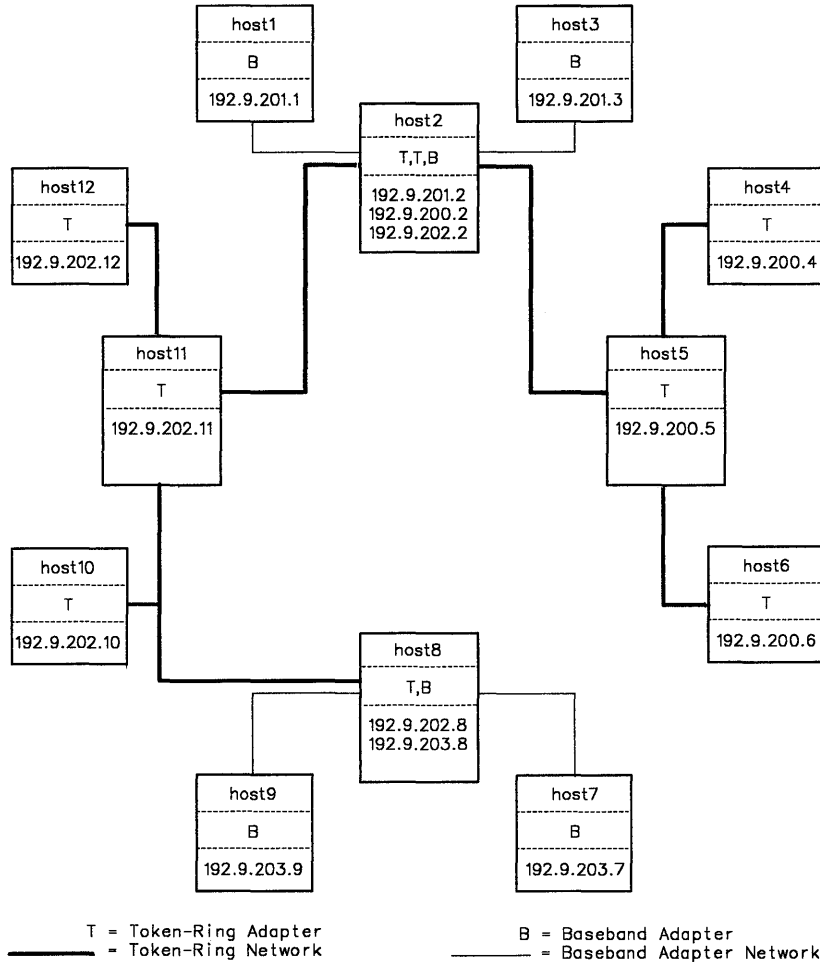
Hosts host2 and host8 are the gateway hosts.

Routes can be established in either of two ways:

- Explicitly, by setting up specific hosts in routing tables
- Dynamically, by running **routed** (the routing daemon) on the gateway hosts.

In Figure 1-2 on page 1-11, the gateway host, host2, contains three interfaces: two Token-Ring Adapters and one Baseband Adapter. The other gateway host, host8, contains two interfaces: one Token-Ring Adapter and one Baseband Adapter.

Because host2 in this example is a gateway host, it is an appropriate host to designate as both the network nameserver and timeserver. Such a host can be reached by all other hosts on the network. In addition, a gateway host is frequently available 24 hours per day. All other hosts on the network should identify host2 with the **nameserver** entry in their `/etc/hosts` files.



OLR20035

Figure 1-2. Network and Gateway Routing

In the Figure 1-2 configuration, routing could be established for host10 as follows: First, establish a default gateway; any packets sent to a host that is not on a network go to the default gateway for that network. To establish host8 as the default gateway on the network that includes host8 and host10, enter the following **route** command on host10:

```
route add 0 host8
```

This route command establishes `host8` as the exit point, or default gateway, from that network.

On the **202** network there are two gateways (`host8`, the default gateway, and `host2`). The preferred form of routing in this network is to the default gateway; if necessary, the gateway can issue ICMP redirect messages to route packets through `host2` to other networks.

Note: This approach to routing works only if the gateway host can send ICMP redirect messages, which the RT PC can do.

An alternate way to establish routing to the other network from the **202** network is to set up routes explicitly with **route** commands like the following ones:

```
route add 192.9.203.0 host8
route add 192.9.201.0 host2
route add 192.9.200.0 host2
```

The size of data packets transmitted through the gateway host may vary, which can affect performance. TCP data transmitted to a gateway host that is not on the same network is sent in 576-byte packets; it would not be efficient to fragment and reassemble TCP packets and some gateways cannot fragment packets. However, IP and UDP packets transmitted through gateways are fragmented and reassembled as necessary.

The Token-Ring Adapter supports both gateways and bridges. With bridging, the routing information field can contain addresses that are up to eight hops away from the sending host. The minimum frame size that can be passed through the bridge is 1K bytes. If the frame size is 1K bytes, **inetlen** should be set to 576. Otherwise, **inetlen** can be set to 1064 or 1576 bytes.

Commands

The Interface Program provides four general types of commands:

- File transfer
- Mail
- Remote login, command execution, and printing
- Network management.

Following are brief explanations of the individual commands in each of these groups.

File Transfer

The Interface Program contains two file transfer commands: **xftp** and **tftp (utftp)**. **xftp** provides more functions and is the preferred command for most file transfer tasks.

xftp

The **xftp** command implements the File Transfer Protocol (FTP), which makes it possible to transfer data among hosts and to use foreign hosts indirectly.

FTP uses a Telnet connection to transfer commands and replies and a data connection to transfer files. **xftp** can be used to transfer files between the user and server or between two hosts. The user must request the close of the Telnet connection when use of the FTP service is finished.

The **xftp** command provides subcommands for such tasks as listing directories (local and foreign), changing directories (local and foreign), transferring multiple files in a single request, creating and removing directories, and escaping the shell (performing shell commands locally). See “**xftp**” on page 2-48 and “**xftpd**” on page 3-27 for related information.

tftp (utftp)

The **tftp** command implements the Trivial File Transfer Protocol (TFTP). Its only function is to read and write files (or mail) to and from a foreign host. **tftp** cannot list directories and it has no provisions for user authentication. **tftp** passes 8-bit bytes of data. **utftp** is a form of **tftp** for use in a pipe.

TFTP is designed to be implemented using UDP. Since UDP is implemented using Internet Protocol, packets have an IP header, a UDP header, and a TFTP header. Additionally, the packets may have a header to allow them through the local transport medium.

For related information, see:

- “**tcom**” on page 2-37
- “**tftp**” on page 2-40

-
- “**tftpd**” on page 3-25
 - “**utftp**” on page 2-46.

Mail

smtp

The **smtp** command implements the Simple Mail Transfer Protocol (SMTP) for transferring mail. **smtp** also provides mail forwarding. SMTP is independent of the TCP subsystem and requires only an ordered data stream channel that is reliable.

The **smtp** command is independent of the AIX **INmail** command and the AIX **mail** command. However, the AIX **mail** command retrieves mail transferred by SMTP.

See “**smtp**” on page 2-34, “**maild**” on page 3-11, and “**smtpd**” on page 3-21 for additional information.

netmail

The **netmail** command is designed to make SMTP simpler to use. Besides making it easier to mail files, **netmail** also allows you to automatically start the editor of your choice, type a message, and then mail the message when you close the file. **netmail** also provides mail forwarding.

See “**netmail**” on page 2-16 for related information.

Remote Login, Command Execution, and Printing

lprbe

The **lprbe** command (a backend program) allows the local host to send a print job to the printer on the foreign host. See “**lprbe**” on page 2-10, “**lpd**” on page 3-7, and the **print** command in *AIX Operating System Commands Reference* for additional information.

rexec

The **rexec** command makes it possible to execute commands remotely without maintaining a **tn** session. For additional information, see “**rexec**” on page 2-26 and “**rexecd**” on page 3-15.

tn

The **tn** command is a terminal emulation program that allows you to log in on a foreign host. It implements the Telnet Protocol, a bi-directional, 8 bit-per-byte communications facility. It provides a standard method to interface terminal devices and terminal-oriented processes to each other. This protocol can also be used for terminal-to-terminal communication (called *linking*) and inter-process communication (called *distributed computation*). See “**tn**” on page 2-43 and “**telnetd**” on page 3-23 for related information.

Network Management

finger

The **finger** command returns information about users on the specified host.

host

The **host** command determines the network address of the specified host. It first searches for the address in a small, local table; then sends requests to the name servers in the network.

hostname

The **hostname** command shows or sets the name and address of the local host.

netconfig

The **netconfig** command specifies which adapter or adapters TCP is to run across and what the adapter characteristics are.

netstat

The **netstat** command shows local and foreign addresses, routing tables, hardware statistics, and summary of packets transferred.

ping

The **ping** command sends an echo request to a network host to determine whether that host is operational and on the network.

route

The **route** command permits you to manually manipulate the routing tables.

setclock

The **setclock** command reads the network time service and sets the time and date of the local host accordingly.

Addresses and Names

This section explains the conventions for assigning addresses and names to hosts on the network.

IP Addressing

The Internet Protocol (IP) uses a two-part, 32-bit *address field*. The first part of the address field contains the network address; the second part contains the local address. There are three different types of address fields (class A, B, or C), depending upon how the bits are allocated.

Figure 1-3 on page 1-16 represents a class A address. It has a 7-bit network number and a 24-bit local address. The highest-order bit is set to 0. There are 128 possible class A networks.

		1	2	3
0	1 2 3 4 5 6 7	8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1		
0	Network	Local Address		

Figure 1-3. Class A Address

Figure 1-4 represents a class B address. It has a 14-bit network number and a 16-bit local address. The highest-order bits are set to 1 and 0. There are 16,384 possible class B networks.

	1	2	3
0 1	2 3 4 5 6 7 8 9 0 1 2 3 4 5	6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1	
1 0	Network	Local Address	

Figure 1-4. Class B Address

Figure 1-5 represents a class C address. It has a 21-bit network number and an 8-bit local address. The three highest-order bits are set to 1, 1, and 0. There are 2,097,152 possible class C networks. In a class C address, a 0 in the last (local address) field is a wildcard; that is, the address 192.9.200.0 addresses all hosts on network 192.9.200.

	1	2	3
0 1 2	3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3	4 5 6 7 8 9 0 1	
1 1 0	Network	Local Address	

Figure 1-5. Class C Address

Notes:

1. Class C addresses starting at 192 are recommended for use with the Interface Program.
2. There is a set of reserved network addresses (for example, for accessing ARPA).
3. No addresses are allowed with the highest-order bits set to 1-1-1. These addresses (sometimes called class D) are reserved.

A 2-byte *type* field in the local header of a packet distinguishes IP addresses from ARP

addresses. The type numbers are:

Protocol	Type Number
IP	0800
ARP	0806

Figure 1-6. IP and ARP Type Numbers

Figure 1-7 represents an IP or ARP local header for the Baseband Adapter:

Source Address	Destination Address	Type Field
6 bytes	6 bytes	2 bytes

Figure 1-7. Local Header, IP or ARP Packet, Baseband Adapter

Figure 1-8 represents an IP or ARP local header for the Token-Ring Adapter:

Medium Access Control (MAC) Header	Logical Link Control (LLC) Header
------------------------------------	-----------------------------------

Figure 1-8. Local Header, IP or ARP Packet, Token-Ring Adapter

The Medium Access Control (MAC) header is composed of five fields, as Figure 1-9 shows:

MAC Header				
AC	FC	DA	SA	RI
1 byte	1 byte	6 bytes	6 bytes	≤ 18 bytes

Figure 1-9. Medium Access Control (MAC) Header, Token-Ring Adapter Local Address

The MAC header fields are:

- AC** Access Control. The value in this field is x'60', which gives the header priority 3.
- FC** Field Control. The value in this field is x'40', which specifies the Logical Link Control frame.
- DA** Destination Address
- SA** Source Address. If bit 0 of this field is set to 1, it indicates that routing information (RI) is present.
- RI** Routing Information. The RI fields are shown in Figure 1-10.

Routing Information (RI)	
RC	Segment Numbers (up to 8)
2 bytes	2 bytes each

Figure 1-10. MAC Header Routing Information, Token-Ring Adapter Local Address

The RI fields are:

RC Routing Control. 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:

bit (0) = 0 Use the non-broadcast 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.

Segment Numbers Up to eight segment numbers of two bytes each to specify recipients of a limited broadcast.

The Logical Link Control (LLC) header is composed of five fields, as Figure 1-11 shows:

LLC Header				
DSAP	SSAP	CONTROL	PROT_ID	TYPE
1 byte	1 byte	1 byte	3 bytes	2 bytes

Figure 1-11. Logical Link Control (LLC) Header, Token-Ring Adapter Local Address

The fields of the LLC header are:

DSAP Destination Service Access Point. The value in this field is x'aa'.

SSAP Source Service Access Point. The value in this field is x'aa'.

CONTROL Determines the LLC commands and responses. There are three possible values for this field:

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 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.

PROT_ID Protocol ID. This field is reserved. It has a value of x'0'.

TYPE Specifies whether the packet is IP or ARP.

A commonly used notation for Internet host addresses is the **dotted decimal**, which divides the 32-bit address into four 8-bit fields. The value of each field is specified as a decimal number and the fields are separated by periods (for example, 010.002.000.052, or 10.2.0.52).

Examples in this publication use the dotted decimal notation in the following forms:

Class A *nnn.lll.lll.lll*

Class B *nnn.nnn.lll.lll*

Class C *nnn.nnn.nnn.lll*

where *nnn* represents part or all of a network number and *lll* represents part or all of a local address.

Internet addresses may also be specified in an octal format of four fields. In octal format, fields are separated by commas. The following octal and dotted decimal addresses are equivalent:

Octal	Dotted Decimal
300,11,310,2	192.9.200.2

Host Names

Each host on the network has a unique name and Internet address. Names are a maximum of 24 characters and cannot contain embedded blanks.

Names and addresses are associated with each other by entries in the `/etc/hosts` file. For more information about entries in `/etc/hosts`, see “**hosts**” on page 1-25.

The **hostname** command must be run to identify the local host to IP. **hostname** takes the name of the host to be identified as a parameter and sets the host name in the **uname** structure. The **hostname** command is typically run at system start by **rc.tcpip**. All host names and addresses to be set by **hostname** must appear in the `/etc/hosts` file. For more information, see “**hostname**” on page 2-8.

The **chparm** command can permanently change the name for the local host that is set in the `/etc/system` and `/etc/master` files. The **hostname** command can make a temporary change to the local host name that is identified to IP.

Warning: If there are any differences in the name set for a host in `/etc/hosts`, `/etc/master`, `/etc/system`, or by the `chparm` command, results will be unpredictable.

For more information about `rc.tcpip`, see “`rc.tcpip`” on page 1-33. For more information about the `chparm` command, see `chparm` in *AIX Operating System Commands Reference*.

Routes

A *route* defines a path for sending information through the network. With the `route` command, you can add and delete the routes defined for a particular host or network.

Routes can specify:

- The host on the network that is the default gateway for sending information to a host on another network
- The gateway from a particular host on one network to a different network
- The path from a particular host on one network to a particular host on a different network.

Routes are defined in a *route table*, which can hold up to 32 routing definitions. The route table is a dynamic structure maintained in the kernel by IP. It is updated by the `route` command or the routing daemon.

The routing daemon (`routed`), which can be run on a gateway host, queries other defined gateway hosts periodically for the information necessary to generate, update, and maintain routing tables. `routed` uses two files, `/etc/gateways` and `/etc/networks`, to determine with which hosts to exchange routing information. For more information, see “`routed`” on page 3-17, “`gateways`” on page 1-23, and “`networks`” on page 1-31.

TCP Addressing

TCP provides a set of 16-bit port numbers within each host. Each host generates port numbers independently and, therefore, it is possible for port identifiers not to be unique. To create an identifier that is unique throughout all networks, TCP concatenates the port number with the IP address, producing a *socket*. A connection is fully specified by the pair of sockets it joins.

Sub-Networks

The sub-network capability of the Interface Program 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 configure its network internally into departmental subnets. Fewer Internet network addresses are required while local routing capabilities are enhanced.

As is explained under “IP Addressing” on page 1-15, a standard Internet 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 (or *mask*) and a host number, for example:

network_number subnet_number host_number

where:

network_number is Internet address for the network.

subnet_number is a field of a constant width for a given network.

host_number is a field that is at least one bit wide.

If the width of the *subnet_number* field is zero, the network is not organized into subnets, and addressing to the network is done with the Internet network address (*network_number*).

The **subnetmask** keyword must be set in the */etc/net* file of a host if that host is to support subnets. Before the sub-network capability can be used, all hosts on the network must support it.

Figure 1-12 represents a class B network address with a 6-bit wide subnet field:

	1	2	3
0 1	2 3 4 5 6 7 8 9 0 1 2 3 4 5	6 7 8 9 0 1	2 3 4 5 6 7 8 9 0 1
1 0	Network	Subnet	Host

Figure 1-12. Class B Address with Subnet

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.

Broadcast Messages

The Interface Program can send data to all hosts on a local network, or to all hosts on all connected networks. Such transmissions are called ***broadcast messages***. For example, the routing daemon (**routed**) uses broadcast messages to maintain routing tables.

For data to be broadcast to all hosts on connected networks, UDP and IP are used to send the data, and the destination address in the IP header is set to **x'FFFFFFFF'**. For data to be broadcast to all hosts on a specific network, the local address part of the IP address is set to zero. There are no user commands that use the broadcast capability, although such commands, or programs, can be developed.

File Formats

This section describes the Interface Program files that you may need to monitor or modify. The files are:

/etc/gateways

Contains information about network gateways for use by the **routed** command.

/etc/hosts

Defines the *hostname* and associated addresses for hosts in the network.

/etc/hosts.equiv

Contains a list of foreign hosts that are allowed to execute commands on a particular host.

/etc/net

Defines all IP characteristics for each adapter card.

/etc/networks

Contains information about known networks.

/etc/rc.tcpip

Sets up network interfaces, host names, and addresses; initializes routes; starts the Interface Program daemons.

\$(HOME)/.netrc

Defines user ID and password information for the **xftp** and **rexec** automatic login features.

\$(HOME)/.3270keys

Defines a user keyboard mapping for Telnet (3270). The **/etc/3270.keys** file defines a default keyboard mapping for Telnet (3270) which is used when **\$(HOME)/.3270keys** does not exist.

gateways

Purpose

Defines and maintains routing information.

Synopsis

/etc/gateways

Description

The */etc/gateways* file identifies gateways for the **routed** command. Ordinarily, **routed** queries the network, building routing tables from routing information transmitted by other hosts that are directly connected to the network. However, there may be gateways that **routed** cannot identify through its queries (*distant* gateways). Such gateways should be identified in */etc/gateways*, which **routed** reads when it starts.

The general format of an entry in */etc/gateways* is:

destination name1 gateway name2 metric value type

Following is a brief description of each element in an */etc/gateways* file entry:

- destination* A keyword that indicates whether the route is to a network or to a specific host. The two possible keywords are **net** and **host**.
- name1* The name associated with *destination*. *name1* can be either a symbolic name (as used in */etc/hosts* or */etc/networks*) or an Internet address specified in dotted decimal format.
- gateway** Indicator that the following string identifies the gateway host.
- name2* Name or address of the gateway host to which messages should be forwarded.
- metric** Indicator that the next string represents the *hop count* to the destination host or network.
- value* The hop count.
- type* A keyword that indicates whether the gateway should be treated as active or passive. The two possible keywords are **active** and **passive**. An active gateway is treated like a network interface (that is, it is expected to exchange routing information and if it does not do so for a period of time, the route associated with it is deleted). A passive gateway is not expected to exchange

gateways

routing information; information about it is maintained in the routing tables indefinitely and is included in any routing information that is transmitted.

Examples

Following are four sample `/etc/gateways` entries:

1. A route to a network, `net2`, through the gateway, `host4`. The hop count metric to `net2` is 4 and the gateway is treated as `passive`.

```
net net2 gateway host4 metric 4 passive
```

2. A route like the one in the previous example except that it is to a specific host (rather than to a network):

```
host net2 gateway host4 metric 4 passive
```

3. A route to a specific host, `host10`, through the gateway, `192.9.201.5`. The hop count metric to `host10` is 9 and the gateway is treated as `active`.

```
net host10 gateway 192.9.201.5 metric 9 active
```

4. A route like the one in the previous example except that the gateway is treated as `passive` (rather than `active`):

```
net host10 gateway 192.9.201.5 metric 9 passive
```

File

`/etc/gateways`

Related Information

In this book: “**routed**” on page 3-17.

hosts

Purpose

Defines *hostname* and associated addresses for hosts in the network.

Synopsis

/etc/hosts

Description

This file contains the *hostnames* and their addresses for hosts in the network. This file is used to resolve a name into an address (that is, to translate a *hostname* into its Internet address).

This file can contain three additional entries (reserved, *well-known* host names):

nameserver
timeserver
printserver

If a *hostname* is not in the **hosts** file, a request to resolve *hostname* to an address is sent to another host, the host associated with the **nameserver** entry. Generally, most hosts in the network have a **nameserver** entry. For example, in a small network, one host can run the **nameserver** daemon; the */etc/hosts* file on that host contains an entry for all hosts in the network. Each of the other hosts in the network contains an entry for itself and for the **nameserver**.

Note: A **nameserver** entry must point to a foreign host.

The host associated with **timeserver** responds to **setclock** requests (a means for synchronizing the time among hosts in the network). Each host may or may not run **timeserver**. If network time is to be used on a particular host, that host must have a **timeserver** entry in its */etc/hosts* file. The **printserver** entry identifies the default host for receiving print requests.

To tailor the network environment for a particular host, modify its */etc/hosts* file. Each entry is of the form:

address hostname

where *address* can be specified in decimal or octal and *hostname* is a string with a maximum length of eight characters and no embedded blanks.

hosts

Examples

Following are sample entries in the `/etc/hosts` files for three different hosts in a network:

Host 1	
192.9.200.1	host1
192.9.200.2	host2
192.9.200.3	host3
128.114.1.15	
128.114.1.14	
128.114.2.7	
192.9.200.2	timeserver
192.9.200.3	printserver

Host 2	
192.9.200.2	host2
192.9.200.1	nameserver
192.9.200.2	timeserver
192.9.200.3	printserver

Host 3	
192.9.200.3	host3
192.9.200.1	nameserver
192.9.200.2	timeserver

In this sample network, the `/etc/hosts` file for `host1` contains address entries for all hosts in the network; `host1` runs the **nameserver** daemon. (The `/etc/hosts` file of `host1` can contain a **nameserver** entry if the entry specifies some host other than `host1`.) The entries in the `host1` `/etc/hosts` file that begin with `128.114` indicate that `host1` also resolves names for hosts on more than one network.

The `/etc/hosts` file of `host2` contains an address entry only for `host2` itself; `host2` runs the **timeserver** daemon. The `/etc/hosts` file of `host3` contains an address entry only for `host3` itself. All three hosts use `host1` to perform the **nameserver** function and `host2` to perform the **timeserver** function. `host3` runs the **printserver** daemon and receives remote print requests from `host1` and `host2`.

File

/etc/hosts

Related Information

“Addresses and Names” on page 1-15.

hosts.equiv

hosts.equiv

Purpose

Defines foreign hosts that are permitted to execute commands.

Synopsis

`/etc/hosts.equiv`

Description

The `/etc/hosts.equiv` file of a particular host defines which foreign hosts are permitted to execute commands on it remotely. The format of the `/etc/hosts.equiv` file is a simple list of host names.

Example

Following are sample entries in an `/etc/hosts.equiv` file:

```
host1
host2
host3
host4
```

File

`/etc/hosts.equiv`

Related Information

In this book: “`rexec`” on page 2-26.

net

Purpose

Defines adapter cards for TCP.

Synopsis

/etc/net

Description

The */etc/net* file contains a keyword associated with each adapter card (device) that TCP can use and a stanza that describes the characteristics for the adapter. There is one stanza entry for each adapter defined for use with TCP. The **netconfig** command processes the */etc/net* file to establish the connection between the */dev/net0* device driver and the adapters.

The format of a stanza in */etc/net* is:

```
sys_stanza_name:  
    netaddr =  
    inetlen =  
    subnetmask =
```

The information contained in the stanza is:

- netaddr** The IP network address to be used for this adapter. It can be in either of the following forms:
- A dotted decimal address
 - An octal address.
- inetlen** The maximum IP packet length for transmission to this adapter. The default packet size for the Baseband Adapter and the IBM Token-Ring Network RT PC Adapter is 1576 bytes. The minimum frame size that can be passed through a bridge is 1K bytes. If the frame size is 1K bytes, **inetlen** should be set to 576. Otherwise, **inetlen** can be set to 1064 or 1576 bytes.
- subnetmask** An optional, hexadecimal mask that defines a subnetwork within a local address. A mask is specified only for the local address portion of an Internet address. If **subnetmask** is not set, subnetworks are not possible.

net

If subnetworks are to be used, all hosts on the network must have **subnetmask** set.

Example

Following is a sample entry in the */etc/net* file:

```
net0:
    netaddr = 192.9.200.1
    inetlen = 1576
    subnetmask = F0
```

The four high-order bits of the local address represent the subnet number. The four low-order bits refer to the local host on the subnetwork.

File

/etc/net

Related Information

In this book: “**netconfig**” on page 2-13.

networks

Purpose

Contains a network name data base.

Synopsis

/etc/networks

Description

The **networks** file contains information about the known networks that comprise the DARPA Internet. Each network is represented by a single line in the **networks** file. The format for the entries in **networks** is:

name number aliases

where:

name is the official network name.

number is the network number.

aliases are the unofficial names used for the network.

Items on a line are separated by one or more blanks or tab characters. Comments begin with the # character, and routines that search **networks** do not interpret characters from the beginning of a comment to the end of that line. Network numbers are specified in dotted decimal notation. A network name can contain any printable character except a field delimiter, new line character, or comment character.

The **networks** file is normally created from the official network data base maintained at the Network Information Control Center (NIC). The file may need to be modified locally to include unofficial aliases or unknown networks.

networks

File

/etc/networks

Related Information

In this book: “**routed**” on page 3-17.

rc.tcpip

Purpose

Sets up host names and addresses, initializes routes, and starts the Interface Program daemons.

Synopsis

`/etc/rc.tcpip`

Description

The **rc.tcpip** file contains the commands necessary for a host to run the Interface Program. The **netconfig** command defines the interfaces (adapter cards) to be configured. The **hostname** command defines the host name and address (the parameter in **rc.tcpip** for **/bin/hostname** must be a name for your system that can be resolved by an entry in **/etc/hosts** to an address in **/etc/net**). The **route** command sets the routing tables. The remaining entries execute the Interface Program daemons:

fingerd	maild	routed	tftpd
icmpd	named	smtpd	timed
lpd	rexecd	telnetd	xftpd

Note: The **routed** daemon should run only on hosts that function as gateways.

If the Interface Program is to be initialized when the system is started, the following line must be in the **/etc/rc** file:

```
sh /etc/rc.tcpip
```

File

`/etc/rc.tcpip`

.netrc

.netrc

Purpose

Contains information used by **rexec** and **xftp** for automatic login.

Synopsis

\$(HOME)/.netrc

Description

The **.netrc** file contains the user ID and password information required for the automatic login features of **rexec** and **xftp**. It is a hidden file in the user's home directory, and its permissions must be set to 600 (read and write by owner only).

The format of an entry in **.netrc** is:

machine *hostname* login *userid* password *password*

where:

hostname is the name of the host on which the user ID exists.

userid is the user ID on that host.

password is the password for that user ID.

The **/usr/lpp/tcpip/samples/netrc** file is a sample **.netrc** file. Use the following procedure to create a **.netrc** file based on the sample:

1. Copy **/usr/lpp/tcpip/samples/netrc** to your home directory.
2. Edit **netrc** to supply the appropriate *hostname*, *userid*, and *password*.
3. Set the permissions on **netrc** to 600.
4. Rename the file **.netrc** (the initial **.**, or dot, causes the file to be hidden).

Files

\$(HOME)/.netrc

/usr/lpp/tcpip/samples/netrc

Related Information

In this book: “**rexec**” on page 2-26 and “**xftp**” on page 2-48.

.3270keys

.3270keys

Purpose

Defines a user-specific keyboard mapping for Telnet (3270).

Synopsis

\$(HOME)/.3270keys

Description

The **.3270keys** file allows a user to have a Telnet (3270) key mapping that is different from the default mapping contained in the **/etc/3270.keys** file. (The mapping in **/etc/3270.keys** is generic; the **.3270keys** file allows users to tailor a mapping to the RT PC keyboard.) The **.3270keys** file exists as a hidden file (the name begins with a **.**, or dot) in the user's home directory.

The **/usr/lpp/tcpip/samples/3270keys.rt** file is a sample that can be used to create a **.3270keys** file. To create a **\$(HOME)/.3270keys** file, use the following procedure:

1. Copy **/usr/lpp/tcpip/samples/3270keys.rt** to the name **.3270keys** in your home directory.
2. Edit the **local key seq** column of **.3270keys** as necessary to create the new mapping. Following is a general procedure for modifying **.3270keys**:

Note: The specific methods for performing these steps may be different for different editors. If you have questions about how your editor performs a task, consult the documentation provided with the program.

- a. Open the **.3270keys** file with an editor.
- b. Move the cursor to the **local key seq** column and then to the 3270 key that you want to map.
- c. Put the editor in **insert** mode (in the **vi** editor, use the **i** subcommand).
- d. Press the key or key sequence that allows you to enter nonprinting characters (in the **vi** editor, press **Ctrl-V**).
- e. Press the key that is to be mapped to the **3270** key.
- f. End **insert** mode (in the **vi** editor, enter **ESC**).
- g. Repeat the steps for each key that is to be mapped.

h. Save the **.3270keys** file and stop the editor.

Note: You can also change the default key mappings by editing `/etc/3270.keys`.

Files

`/etc/3270.keys`

`/usr/lpp/tcpip/samples/3270keys.rt`

`$(HOME)/.3270keys`

Related Information

In this book: “**tn**” on page 2-43.

Assigned Numbers

For compatibility with the general network environment, numbers are assigned for ports, the version, and protocols. The following three sections explain the assigned numbers.

Port Numbers

Ports are used in the TCP to name the ends of logical connections that carry long term conversations. For the purpose of providing services to unknown callers, a service contact port is defined. This list specifies the port used by the server process as its contact port. The contact port is sometimes called the *well-known port*.

The assigned ports use a small portion of the possible port numbers. On an assigned port, all but the low-order 8 bits are cleared to 0 (zero). The following table specifies the low-order 8 bits of the assigned port numbers:

Port Number (decimal)	Keyword	Description
21	FTP	File Transfer [Control]
23	TELNET	Telnet
25	SMTP	Simple Mail Transfer
37	TIME	Time
42	NAMESERVER	Host Name Server
69	TFTP	Trivial File Transfer
79	FINGER	Finger
512	REXEC	Remote execution
515	LPRBE	Remote print
520	ROUTED	Routing daemon

Version Numbers

The Internet Protocol (IP) includes a 4-bit field to identify the version of the general inter-network protocol in use. Following the assigned version number:

Version Number (decimal)	Keyword	Version
4	IP	Internet Protocol

Protocol Numbers

The Internet Protocol (IP) includes an 8-bit field, called **protocol**, which identifies the next level protocol. Following are the assigned protocol numbers:

Protocol Number (decimal)	Keyword	Protocol
1	ICMP	Internet Control Message
3	GGP	Gateway-to-Gateway
6	TCP	Transmission Control
17	UDP	User Datagram

Security

This section describes the security features provided with the Interface Program for use with TCP/IP and some security considerations that are appropriate in a network environment.

Security Features

The **lprbe**, **rexec**, Telnet, and **xftp** functions provide the following forms of system and data security:

lprbe The **/etc/hosts.equiv** file provides a secure environment for the **lprbe** (remote print) command. The only hosts that can use a particular host as a print server are the ones listed in the **/etc/hosts.equiv** file of that host. For more information about **/etc/hosts.equiv**, see “**hosts.equiv**” on page 1-28.

rexec The **rexec** command provides a secure environment for executing commands on a foreign host. The user is prompted for both a login ID and a password.

To provide automatic login, the **-n** flag causes **rexec** to search the **\$(HOME)/.netrc** file for the user’s ID and password on a foreign host. For security, the permissions on **\$(HOME)/.netrc** must be set to 600 (read and write by owner only). Otherwise, automatic login fails.

For more information, see “**rexec**” on page 2-26 and “**.netrc**” on page 1-34.

tn The **tn** (Telnet) function provides a secure environment for login to a foreign host. The user is prompted for both a login ID and a password. The user’s terminal is treated just like a terminal connected directly to the host. That is, access to the terminal is controlled by permission bits; other users (group and other) do not have read access to the terminal, but they can write messages to it if the owner gives them write permission. For more information and an example, see “**tn**” on page 2-43.

xftp The **xftp** function provides a secure environment for transferring files. When a user invokes **xftp** to a foreign host, the user is prompted for a login ID. A default login ID is shown—the user’s current login ID on the local host. If a password is associated with that login ID at the foreign host, the user is prompted for a password.

To provide automatic login, the **-n** flag causes **xftp** to search the **\$(HOME)/.netrc** file for the user’s ID and password on a foreign host. For security, the permissions on **\$(HOME)/.netrc** must be set to 600 (read and write by owner only). Otherwise, automatic login fails. For more information, see “**.netrc**” on page 1-34 and “**xftp**” on page 2-48.

Data Security and Information Protection

The Interface Program for use with TCP/IP does not encrypt user data transmitted through the network. Therefore, it is suggested that users identify any risk in communication that could result in the disclosure of passwords and other sensitive information, and based on that risk, apply appropriate countermeasures.

The use of this product in a Department of Defense environment may require adherence to DoD 5200.5 and NCSD - 11 for communications security.

Program Customization

This section describes the usual means for customizing the Interface Program to meet your requirements.

EDITOR

The **netmail** command uses an environment variable called **EDITOR**, which defines the default editor invoked by **netmail**. If this variable is not defined, **netmail** reads from standard input. For more information, see “**netmail**” on page 2-16.

EMULATE

The **tn** command uses an environment variable called **EMULATE** that defines the emulation mode used by Telnet. For more information, see “**tn**” on page 2-43.

/etc/hosts

The **/etc/hosts** file contains the names and associated addresses for hosts on the network. That is, **/etc/hosts** lists the foreign hosts with which a local host can communicate. For more information, see “**hosts**” on page 1-25.

/etc/hosts.equiv

The **/etc/hosts.equiv** file contains the names of the foreign hosts that are permitted to perform remote operations on a particular host. For more information, see “**hosts.equiv**” on page 1-28 and “**lpd**” on page 3-7.

/etc/net

The **/etc/net** file contains a description of the interfaces available on a particular host. For more information, see “**net**” on page 1-29.

/etc/networks

The **/etc/networks** file contains information about known networks that comprise the DARPA network. For more information, see “**networks**” on page 1-31.

/etc/rc.tcpip

The **/etc/rc.tcpip** file is a shell script that defines which adapters are used for the Interface Program, initializes the host names and routing tables, and starts the Interface Program daemons. The **/etc/rc.tcpip** file may be tailored to the requirements of a particular host. For more information, see “**rc.tcpip**” on page 1-33 and Appendix A, “Customizing the Program.”

/etc/3270.keys

The **/etc/3270.keys** file contains the default key mapping for use with Telnet (3270).

finger

The **finger** command provides information about a specific user. This information is retrieved from the full name and site information fields of the **/etc/passwd** file. The format of the full name is the user’s real name (or whatever is entered in the full name field of **/etc/passwd**, not the user’s user ID. The site information field is 20 characters long and has the following format (no imbedded blanks, fields separated by commas):

office_number,office_phone_extension,home_phone

The office_phone_extension field can be up to four characters long. The home_phone field can be either seven or 10 characters long.

Note: To update the **/etc/passwd** file, use the **adduser** command.

If you want to send additional information to other users who run **finger** on your user ID, you can include the following files in your home directory:

.plan A file that contains plans. The **.plan** file can contain more than one line.

.project A file that states what project you are currently working on. The **.project** file can contain only one line.

For additional information about the **finger** command, see “**finger**” on page 2-3.

\$(HOME)/.netrc

The **\$(HOME)/.netrc** file contains the user ID and password information required by the automatic login features of **rexec** and **xftp**. For more information, see “**.netrc**” on page 1-34.

\$(HOME)/.3270keys

The **\$(HOME)/.3270keys** file contains a user-specific keyboard mapping for Telnet (3270). For more information, see “**.3270keys**” on page 1-36.

Chapter 2. User Commands

CONTENTS

About This Chapter	2-2
finger	2-3
host	2-6
hostname	2-8
lprbe	2-10
netconfig	2-13
netmail	2-16
netstat	2-19
ping	2-23
rexec	2-26
route	2-29
setclock	2-32
smtp	2-34
tcom	2-37
tftp	2-40
tn	2-43
utftp	2-46
xftp	2-48

About This Chapter

This chapter describes the IBM RT PC Interface Program for use with TCP/IP user commands (application programs). Chapter 3, "Server Commands" on page 3-1 describes the Interface Program server commands, or daemons.

Following is a list of the user commands, grouped according to function:

- File transfer
 - tcom**
 - tftp**
 - utftp**
 - xftp**
- Mail
 - netmail**
 - smtp**
- Remote login, command execution, and printing
 - lprbe**
 - rexec**
 - tn**
- Network management
 - finger**
 - host**
 - hostname**
 - netconfig**
 - netstat**
 - ping**
 - route**
 - setclock**

In this chapter, the user commands are organized alphabetically.

finger

Purpose

Shows user information.

Syntax

```
finger [-d] user @host
```

OLR20004

Description

The **finger** command displays information about the users currently using the specified *host*. If you specify a *user* name, **finger** displays information about that user, including:

- login ID
- full name
- terminal name and write status (a * before terminal name indicates that write permission is denied)
- idle time
- login time
- office_location, office_phone_number, and home_phone (if known)
- The contents of a **.plan** file in the user's home directory, if one exists
- The contents of a **.project** file in the user's home directory, if one exists.

Otherwise, it provides a list of all users, like the **who** command. For other information about **finger**, see "Program Customization" on page 1-41.

finger

Flag

-d Records packets in the file `./finger.log`.

Examples

1. To get information about all users logged in to the host, `host1`:

```
$ finger @host1
  smith      console  Mar 15 13:19
  green      pts0      Mar 15 13:01
  thomas     tty0      Mar 15 13:01
$ -
```

User `smith` is logged in at the console, user `green` is logged in from a pseudo teletype line (Telnet connection), and user `thomas` is logged in from a `tty`.

2. To get information about user `smith` at `host1`:

```
$ finger smith@host1
Login name:  smith           In real life:  Sam T. Smith
Office:     3D08 ext5555     Home phone:   987-6543
Directory:  /u/smith
Never logged in.
No plan.
$ -
```

3. To get information about user `thomas` at `host1`:

```
$ finger thomas@host1
Login name:  thomas         In real life:  A. B. Thomas
Office:     3D10 ext5322    Home phone:   210-9876
Directory:  /
On since April 16 11:06:18 on console 1 minute 28 seconds Idle Time
Project.:   aquatic entomology
Plan:
Complete Phase 1 research by end of second quarter.
Produce draft report by end of year.
$ -
```

Files

<code>/dev/net0</code>	Network device
<code>./finger.log</code>	Packets from finger transaction

Related Information

The **who** command in *AIX Operating System Commands Reference*.

In this book: “**fingerd**” on page 3-3.

host

host

Purpose

Resolves the network host name.

Syntax

`host — hostname —`

OLR20007

Description

The **host** command determines the network address of the specified host. It looks in a small local table first, then sends requests to the name servers in the network, which is defined in `/etc/hosts` as **nameserver**. The network address displays as four bytes in decimal, separated by dots, and in octal notation, separated by commas. The **host** command accepts both unique host names and the well-known host names **nameserver** and **timeserver** as parameters.

Example

To display the address of host2:

```
$ host host2
host2 is 192.9.200.5 (300,11,310,5)
$ -
```

Files

<code>/dev/net0</code>	Network device
<code>/etc/hosts</code>	Name to address map table

Related Information

In this book: “**named**” on page 3-13.

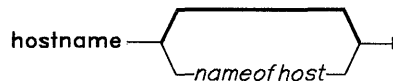
hostname

hostname

Purpose

Sets or displays the name of the local host system.

Syntax



OLR20008

Description

The **hostname** command displays the name of the local host. If you have superuser authority, you can change the host name and address by supplying the *nameofhost* parameter. This is usually done in the startup script **/etc/rc.tcpip**. The *nameofhost* parameter must be supplied in the **/etc/hosts** file to set the host name and the Internet address of this host. The *nameofhost* parameter is a character string of up to 24 characters. If no parameter is supplied, the **hostname** command prints the name and Internet address of the primary interface. **hostname** does not modify the **/etc/hosts** file.

Note: Before you can set the name of the local host, you must use the **netconfig** command to map an address to *nameofhost*. Otherwise, **hostname** displays an error message.

A **gateway host** (a host that connects independent networks) has multiple interfaces, each with a different name and address. The **hostname** command sets the primary name of the gateway host. References to gateway hosts can be by primary or secondary name.

Warning: The host name set by **hostname** for a particular host must be the same as the name listed **/etc/master** (where it is called node rather than hostname) and the name set by **chparm**, or results will be unpredictable.

Examples

1. To display the name and address of the local host:

```
$ hostname
Node name: host1 Internet Address: 192.9.200.2 (300,11,310,2)
$ -
```

2. To change the name of the local host:

```
$ hostname west
Nodename: west Internet Address: 192.9.200.3 (300,11,310,3)
$ -
```

File

`/etc/hosts` Name to address map table

Related Information

In this book: “**host**” on page 2-6.

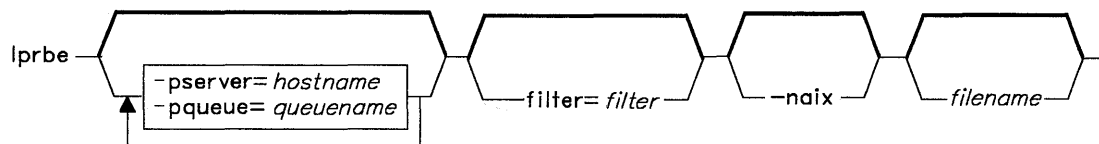
lprbe

lprbe

Purpose

Sends a file to a print server.

Syntax



OLR20028

Description

The **lprbe** command is a backend program that sends print requests to a remote print server (the **lpd** program and printer on a foreign host). **lprbe** is normally called by the **qdaemon** command after you have enqueued a file with the **print** command. The flags and parameters that you enter with **print** are passed to **lprbe**, and it is **lprbe** that determines where and how the remote print job will be done. **lprbe** supports the AIX **pio** and **print** flags as well as a set of filters that may exist on non-AIX systems. For more information on **pio** and **print**, see *AIX Operating System Commands Reference*.

Flags

- pserver=hostname** Specifies which host is to receive the print request. If **-pserver=** is not specified, **lprbe** sends the print request to the host specified in the **printserver** entry in the **/etc/hosts** file or to the remote print server identified in the **/etc/qconfig** file.
- pqueue=queueName** Specifies a particular remote printing queue that is to receive the print request.
- filter=filter** *filter* can be either a user-defined program (with or without its own flags) that pipes its output to **print**, or one of the following flags (which indicate that the files to be printed are not standard text files). If *filter* contains embedded blanks, it must be enclosed in double quotes (" "). These flags generate a control file that is

compatible with non-AIX systems; if they are sent to an AIX system, they are ignored.

- c** Handles files that contain data produced by **cifplot**.
 - d** Handles files that contain **tex** data.
 - f** Uses a filter that interprets the first character of each line as a standard FORTRAN carriage control character.
 - g** Handles files that contain standard plot data as produced by the **plot** routines.
 - l** Uses a filter which allows control characters to be printed and suppresses page breaks.
 - n** Handles files that contain **ditroff** data.
 - p** Uses **pr** to format the files. The results are equivalent to those obtained with the **print** command.
 - r** Handles files that contain FORTRAN carriage control characters.
 - t** Handles files that contain **troff** data.
 - v** Handles files that contain a raster image for devices like the Benson Varian.
- naix** Generates a print control file for a non-AIX system.
- filename* Names one or more files to be printed. If you enter **lprbe** without a *filename*, it reads from standard input.

Examples

1. To print the file `testcase` on a foreign AIX host (when `rp0` is the name of the local host queue that handles outbound print requests):


```
$ print rp0 testcase
$ -
```
2. To print on a foreign host (`host1`) other than the default **printserver**:


```
$ print rp0 -pserver=host1 testcase
$ -
```


lprbe

3. To select a specific print queue (lp1) on a foreign host:

```
$ print rp0 -pqueue=lp1 testcase
$ -
```

4. To perform preprocessing on the foreign host (by the **pr** command) before printing:

```
$ print rp0 -filter"pr -w60 -i5" testcase
$ -
```

This filter generates the following command on the foreign host:

```
pr -w60 -i5 testcase | print
```

5. To print on a non-AIX foreign host (that is configured as the default **printserver**):

```
$ print rp0 -naix testcase
$ -
```

Files

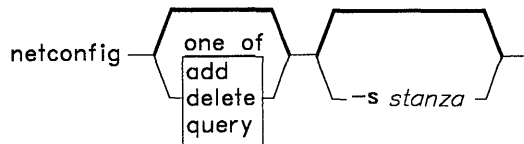
/etc/qconfig	Queueing system configuration
/dev/net0	Network device

netconfig

Purpose

Configures `/dev/net0` for use with multiple adapter cards.

Syntax



OLR20030

Description

The **netconfig** command processes the `/etc/net` file, establishing which adapter cards the `/dev/net0` device driver can communicate with. When issued without parameters, **netconfig** processes the entire `/etc/net` file and adds each stanza.

The **netconfig** command reads the specified stanza (or all stanzas) in the `/etc/net` file and, from the stanza name, locates the corresponding stanza in the `/etc/system` file. The `/etc/system` file provides the IODN and device type required by the IOCCONFIG structure (which is defined in the file, `/usr/include/sys/iocfg.h`). To complete the configuration, **netconfig** invokes the IOCCONFIG IOCTL routine of the `/dev/net0` device driver; the IOCCONFIG IOCTL routine performs a NIOCSHOST for each IODN to set the local IP address.

Flags

- | | |
|---------------|--|
| add | Adds the interface specified by <code>-s stanza_name</code> . If <code>-s stanza_name</code> is not specified, netconfig processes all stanzas in the <code>/etc/net</code> file and adds those interfaces. |
| delete | Deletes the interface specified by <code>-s stanza_name</code> . If <code>-s stanza_name</code> is not specified, netconfig processes all stanzas in the <code>/etc/net</code> file and deletes those interfaces. |

netconfig

query Returns the state of the interface specified in **-s stanza_name**. If **-s stanza_name** is not specified, **netconfig** returns the state of all interfaces defined by stanzas in **/etc/net** via the **IOCINFO** IOCTL.

-s stanza_name Indicates that only the specified stanza, *stanza_name*, is to be processed.

Examples

1. To add all interfaces defined in **/etc/net**:

```
# netconfig
# -
```

or

```
# netconfig add
# -
```

2. To add a specified interface, **net1**, that is defined in **/etc/net**:

```
# netconfig add -s net1
# -
```

3. To delete all interfaces defined in **/etc/net**:

```
# netconfig delete
# -
```

4. To delete a specified interface, **net1**, that is defined in **/etc/net**:

```
# netconfig delete -s net1
# -
```

5. To query all interfaces defined in **/etc/net**:

```
# netconfig query
```

```
net0:
Internet Address: 192.9.200.1 inetlen: 1576
hardware type=ethernet subnet mask: 0
```

```
net1:
Internet Address: 192.9.201.1 inetlen: 1576
hardware type=ethernet subnet mask: 0
```

```
net2:
Internet Address: 192.9.202.1 inetlen: 1576
hardware type=token ring subnet mask: 0
```

```
net3:
Internet Address: 192.9.203.1 inetlen: 1576
hardware type=token ring subnet mask: 0
```

```
# -
```

6. To query a specified interface, net0:

```
# netconfig query -s net0
net0:
Internet Address: 192.9.200.1 inetlen: 1576
hardware type=ethernet subnet mask: 0
```

Files

<code>/dev/net0</code>	Network device
<code>/etc/net</code>	Adapter card definition for TCP.

Related Information

In this book: “**net**” on page 1-29.

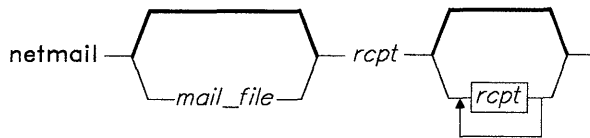
netmail

netmail

Purpose

Provides a simplified user interface to **smtp**.

Syntax



OLR20031

Description

The **netmail** command, used to send a *mail-file* to a user, is a simplified user interface to **smtp**. The *mail-file* parameter specifies an existing file to be sent. If it is entered without the *mail-file* parameter, **netmail** starts the editor specified in the environment variable, **EDITOR**. (If no editor is specified, **netmail** reads from standard input.) When the editing session is completed, **netmail** sends the newly created file to the recipient (*rcpt*). *rcpt* is a mailbox address in the **smtp** format, *user@host@host . . .* **netmail** will also accept redirected standard input.

The **netmail** command uses the mail relay service provided by the **maild** daemon. If **maild** is unable to deliver mail after 100 attempts, it notifies the sender that the mail remains undelivered and returns the mail to the sender.

Flags

<i>mail-file</i>	Name of the file that contains the message to be sent.
<i>rcpt</i>	Mailbox address of the recipient of the message in the form of <i>user@host@host . . .</i>

Examples

1. To send the existing file mailnote to user smith at host1:

```
$ netmail mailnote smith@host1
$ -
```

2. To send the file mailnote to users smith and tom, specifying a forwarding path for user tom:

```
$ netmail mailnote smith@host1 tom@host1@host2
$_
```

3. To create a new note and send it to users smith and tom:

```
$ netmail smith@host1 tom@host1@host2
.
.
.
(Editor defined by EDITOR started; note typed; editing session
ended.)
$ -
```

4. To create and send a note without the **EDITOR** environment variable set, enter the note on the command line, pressing **Enter** at the end of each line, and then press EOF (Ctrl-D) to end **netmail** and send the note:

```
$ netmail smith@host1
(Enter note on the command line.)
Ctrl-D
$ -
```

Files

/dev/net0	Network device
/etc/hosts	Name to address map table
/etc/mailed	Daemon that attempts to deliver mail
/maild/inbox	Unique message ID source (*)
/maild/smtp	Relay information for mail file
/maild/smtp/t*	Mail file
/maild/smtp/t*.reg	Directory for relaying mail

Related Information

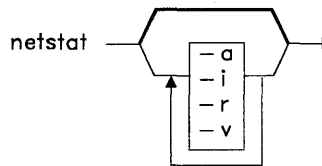
In this book: “Program Customization” on page 1-41, “**smtp**” on page 2-34 and “**maild**” on page 3-11.

netstat

Purpose

Shows network status and provides problem determination information.

Syntax



OLR20011

Description

The **netstat** command symbolically displays the contents of various network-related data structures. **netstat** does not accept any arguments and returns information about active connections only.

Flags

- a Shows the state of all connections. Connections used by server processes are not normally shown.
- i Shows the state of automatically configured connections (connections statically configured into a system, but not located at system start, are not shown).
- r Shows the routing tables.
- v Shows statistics about the VRM Baseband Adapter Device Driver and the VRM Token-Ring Adapter Device Driver.

netstat

Examples

1. To show the state of the connections that were automatically configured:

```
$ netstat -i
My Internet Address: 192.9.200.2
My Ethernet Address: 00:dd:00:94:62:00

          **** DEVICE DRIVER STATISTICS ****
Bytes in : 1162      Pkts in  : 15
Bytes out : 724      Pkts out  : 17
Pkts dropped : 0

Interrupts : 30
  Solicited inter : 15
    Timeout : 0      Init_fail : 0
  Unsolicited inter : 15
    Data avail : 15 SLIH underflow : 0  Device overflow : 0
$ _
```

2. To show the state of all connections:

```
$ netstat -a
Proto  Recv pkts  Send pkts  Local Address      Foreign Address
TCP    0           0           192.9.200.2..79    *.*
UDP    0           0           192.9.200.2..37    *.*
UDP    0           0           192.9.200.2..69    *.*
TCP    0           0           192.9.200.2..512   *.*
TCP    0           0           192.9.200.2..23    *.*
ICMP   0           0           192.9.200.2..*     *.*
GGP    0           0           192.9.200.2..*     *.*
TCP    0           0           192.9.200.2..25    *.*
TCP    0           0           192.9.200.2..21    *.*
UDP    0           0           192.9.200.2..42    *.*

          **** INTERNET ERROR STATISTICS ****
Bad pkt len: 0  Bad proto: 0  Bad IP hdr len: 0  Bad IP len: 0
Bad checksum: 0  Bad dest: 0  Fragments: 0      No conn: 0
Drops: 0
```

```
**** ADDRESS RESOLUTION PROTOCOL STATISTICS ****
Send: 1      Rcv req: 1      Rcv reply: 1      Not for me: 1
Bad pkts: 0  Bad len: 0      Unexpected: 0
$ -
```

3. To show the routing tables:

```
$ netstat -r
***** Host-Gateway Table *****
Host                Gateway              Metric

Default Gateway: 192.9.200.6
$ -
```

4. To show VRM Baseband Adapter Device Driver and Token-Ring Adapter statistics:

```
$ netstat -v
***** VRM STATISTICS *****

net0:
Receive interrupts: 35  Packets accepted: 37  Packets Rejected: 0
Packets transmitted: 37
Collisions: 0      16 Collisions: 0  Shorted: 0  Underflow: 0
Short packets: 0  Alignment errors: 0  CRC errors: 0  Overflow: 0

token0:
Receive interrupts: 45  Packets accepted: 47  Packets Rejected: 0
Transmit Interrupts: 47      Transmit Completes: 47
Packets transmitted: 47
```

File

/dev/net0 Network device

netstat

Related Information

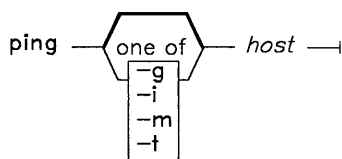
IBM RT PC Virtual Resource Manager Technical Reference

ping

Purpose

Sends an echo request to a network host.

Syntax



OLR20012

Description

With no flags or with the **-g** flag, **ping** sends a Gateway-to-Gateway Protocol (GGP) echo request to the specified network *host*. If the host is operational and on the network, it responds to the echo. If the **-i** flag is specified, an Internet Control Message Protocol (ICMP) echo request is sent instead. If the **-t** flag is specified, an ICMP timestamp request is sent. This is useful for determining round-trip delays to other hosts. The **-m** flag causes **ping** to return the subnet mask for the specified host. **ping** is useful for determining the status of the network and various foreign hosts.

Flags

- g** Sends a GGP echo request to the specified host.
- i** Sends an ICMP echo request to the specified host.
- m** Returns the subnet mask for the specified host.
- t** Sends an ICMP timestamp request.

ping

Examples

1. To send a GGP echo request to host host1:

```
$ ping -g host1
Ping host1: 192.9.200.2 responding
$ -
```

or

```
$ ping host1
Ping host1: 192.9.200.2 responding
$ -
```

2. To send an ICMP echo request to host host1:

```
$ ping -i host1
Ping host1: 192.9.200.2 responding
$ -
```

3. To determine the round trip time to host host1:

```
$ ping -t host1
Ping host1: Timestamp from 192.9.200.2
  Orig: 14:35:20:300 Rcvd: 14:35:20:300 Xmtd: 14:35:20:300
  Done: 14:35:20:316
$ -
```

4. To display the subnet mask for host1:

```
$ ping -m host1
Ping host1: Address mask from 192.9.200.2:  ffffffff00
$ -
```

Files

/dev/net0	Network device
/etc/locks/icmp	PID of icmpd
/etc/icmpd	icmp daemon

Related Information

In this book: “**icmpd**” on page 3-5.

rexec

rexec

Purpose

Executes commands one at a time on a foreign host in a secure environment.

Syntax

rexec *-d -n* *hostname* *command*

OLR20032

Description

The **rexec** command executes a command on the specified foreign host. The user is prompted for a valid user ID and password (if applicable) for the foreign host. **rexec** sends and receives data over a TCP connection.

The **rexec** command also allows you to run interactive commands remotely, provided that they do not require a full screen display. In interactive processing, **rexec** sends all characters typed at the local keyboard to the foreign host until the EOF character is sent. If the shell running on the foreign host is **cs**h, **rexec** sends **SIGINT** and **SIGQUIT** to the foreign host.

Flags

- d** An optional flag that provides a debug service. The **-d** flag causes **rexec** to print debug statements to the file **rexec.log** in the current directory.
For each user who concurrently runs commands on a host with **rexec**, that host must have three PTYs configured. That is, if three users on **host1** are using **rexec** to run commands on **host2** concurrently, **host2** must have nine PTYs configured. In the PTY definitions, the **auto enable** and **logger** keywords must be set to **false**.
- n** An optional flag that provides an automatic login feature. The **-n** flag causes **rexec** to search the **\$(HOME)/.netrc** file for the user's ID and password on the foreign host.

-
- hostname* A required parameter that specifies the name of the host where the command is to be executed. *hostname* can be in octal, dotted decimal, or name (alphanumeric) form.
- command* A required parameter that specifies the command to be executed on the foreign host. If *command* (that is, the command name, together with flags or parameters) contains embedded blanks, it may be enclosed in double quotes.

Examples

1. To execute the **date** command on foreign host **host1**:
\$ rexec host1 date
User: tom
Password:
Command execution has begun.
[*date command output*]
& -
2. To list the directory of user **tom** on the foreign host, **host1**:
\$ rexec host1 "li -l /u/tom"
User: tom
Password:
Command execution has begun.
[*listing of tom's directory on foreign host*]
\$ -
3. To list the **/tmp** directory of the foreign host, **host1**:
\$ rexec host1 li /tmp
User: tom
Password:
Command execution has begun.
[*listing of /tmp on foreign host*]

rexec

4. To start a shell (**cs**) on the foreign host, `host1`, enter:

```
$ rexec host1 csh
User: tom
Password:
Command execution has begun.
%
.
.
.
[EOF to end the foreign shell.]
```

5. To use the automatic login feature for the user ID **tom** on the foreign host, `host1`, enter:

```
$ rexec -n host1 date
User: tom
$ -
```

Files

<code>/etc/hosts</code>	Name to address map table
<code>/dev/net0</code>	Network device
<code>./rexec.log</code>	Debugging statements
<code>\$(HOME)/.netrc</code>	User IDs and passwords for remote login.

Related Information

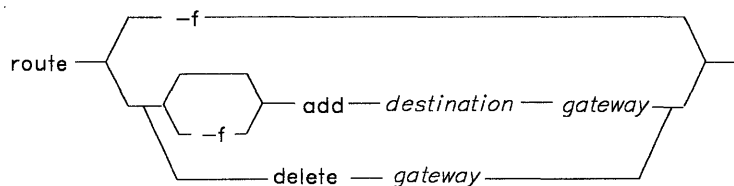
In this book: “**rexecd**” on page 3-15.

route

Purpose

Manually manipulates the routing tables.

Syntax



OLR20013

Description

The **route** command manually manipulates the network routing tables. It accepts two commands:

add Adds a route.

delete Removes a route.

The *destination* parameter names a host or network where the route is directed, and the *gateway* parameter names the gateway to which the packets should be addressed. Interpretation of the network address associated with *destination* distinguishes routes to a particular host from those to a network. If the local address is not specified (that is, only the network address is specified), the route is assumed to be the network. Otherwise, the route is assumed to be to a host. All symbolic names specified for a *destination* or *gateway* are resolved either through **/etc/hosts** or the network **nameserver**. The **netstat** command shows the information contained in the routing tables. You must have superuser authority or be a member of the system group to run the **route** command.

route

Flag

- f Provides a flush option. When run with the **-f** flag, **route** clears the host gateway table before changing the routing table.

Examples

1. To establish a route for sending a message to host address 192.9.201.7, which is not on the same network as the host sending the message:

```
$ route add 192.9.201.7 192.9.200.7
$ -
```

2. To establish a route that will enable you to send a message to any user on network 192.9.201:

```
$ route add 192.9.201.0 192.9.200.7
$ -
```

3. To establish a default gateway:

```
$ route add 0 192.9.200.7
$ -
```

The value 0 for the destination means that any packets sent to any destination on any other network go through the default gateway, 192.9.200.7.

4. To clear the host gateway table:

```
$ route -f
$ -
```

5. To clear the host gateway table and then add a destination gateway:

```
$ route -f add 192.9.201.0 192.9.200.7  
$ -
```

6. To delete a gateway:

```
$ route delete 192.9.200.7  
$ -
```

File

/dev/net0 Network device

Related Information

In this book: “**routed**” on page 3-17.

setclock

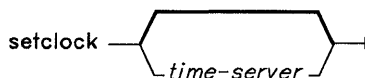
setclock

Purpose

Sets the time and date.

Syntax

setclock *time-server*



OLR20014

Description

The **setclock** command reads the network time service and sets the RT PC time and date accordingly. This command sends a request, using the standard Internet Protocol time service protocol to a *time-server*. The *time-server* parameter is either a character-string name or an address of a network host that provides an Internet Protocol time daemon. If the *time-server* name is omitted, **setclock** sends requests to the default time server, **timeserver**. **setclock** takes the first response, converts the calendar clock reading found therein to the local date and time and displays it. Finally, if **setclock** is run by a user with superuser authority, it calls the standard RT PC entry points to set the system date and time.

If no time server responds, or the network is not operational, **setclock** displays a message to that effect and leaves the current date and time settings of the system unchanged.

setclock is designed for use either as a stand-alone command or as a command that can be invoked in the */etc/rc* command file.

Examples

1. To display the date and time using the timeserver host specified in `/etc/hosts`:

```
$ setclock
Sat Mar 15:31:05 1986
$ -
```

2. To set the date and time from the `timeserver` in `host1`:

```
$ su
# setclock host1
Sat Mar 15:32:27 1986
```

File

`/dev/net0` Network device

Related Information

In this book: “`timed`” on page 3-26.

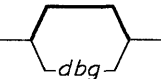
smtp

smtp

Purpose

Provides the Simple Mail Transfer Protocol.

Syntax

smtp —*mail-file* —*from-address* —*dest-host*—*recipient* —

OLR20015

Description

The **smtp** command implements a user-mode Simple Mail Transfer Protocol (SMTP). The **netmail** command uses **smtp** to send outgoing mail to other hosts in the network (see “**netmail**” on page 2-16 for additional information).

The **smtp** command accepts the following parameters:

- | | | | | | | | |
|---------------------|---|---|--------------------------------------|---|---|---|--------------------------------------|
| <i>mail-file</i> | Name of the file that contains the message text to be transmitted. | | | | | | |
| <i>from-address</i> | Address of the original sender of the message, in the user@host@host . . . format. | | | | | | |
| <i>dest-host</i> | Name of the next host in the path for this message. <i>dest-host</i> must be an entry in /etc/hosts . | | | | | | |
| <i>recipient</i> | Mailbox address of the ultimate recipient of the message, in the user@host@host . . . format, where host is the primary name. | | | | | | |
| <i>dbg</i> | Starts logging. The possible values for <i>dbg</i> are:
<table><tr><td>1</td><td>Gets smtp text type messages.</td></tr><tr><td>2</td><td>Performs tracing of input and output packets.</td></tr><tr><td>3</td><td>Performs the 1 and 2 tasks together.</td></tr></table> | 1 | Gets smtp text type messages. | 2 | Performs tracing of input and output packets. | 3 | Performs the 1 and 2 tasks together. |
| 1 | Gets smtp text type messages. | | | | | | |
| 2 | Performs tracing of input and output packets. | | | | | | |
| 3 | Performs the 1 and 2 tasks together. | | | | | | |

The **smtp** command displays a line on its standard output describing the results of the transfer. The line includes a numeric code, indicating success or failure, to be used by the mail daemon. In addition, the command returns a 0 exit status on success, and non-zero exit status on failure. Following is a list of the **smtp** numeric reply codes:

- 211 System status or system help reply.
- 214 Help message (information on how to use the receiver or the meaning of a particular non-standard command).
- 220 host-name Service ready.
- 221 host-name Service closing transmission channel.
- 250 Requested mail action okay, completed.
- 251 User not local; will forward to *forward-path*.
- 354 Start mail input; end with *CRLF.CRLF*.
- 421 host-name Service not available; closing transmission channel. (This may be a reply to any command if the service knows it must shut down.)
- 450 Requested mail action not taken; mailbox unavailable (for example, the mailbox is busy).
- 451 Requested action stopped; local error in processing.
- 452 Requested action not taken; insufficient system storage.
- 500 Syntax error, command unrecognized. (This may include errors such as a command line that is too long.)
- 501 Syntax error in parameters or arguments.
- 502 Command not implemented.
- 503 Bad sequence of commands.
- 504 Command parameter not implemented.
- 550 Requested action not taken; mailbox unavailable (for example, if the mailbox cannot be found or it does not allow appropriate access).
- 551 User not local; please try *forward-path*.
- 552 Requested mail action stopped; exceeded storage allocation.
- 553 Requested action not taken; mailbox name not allowed (for example, the mailbox syntax is incorrect).
- 554 Transaction failed.

smtp

Example

To send a copy of **/etc/hosts** to user jane at tcprt1:

```
$ smtp /etc/hosts smith@tcprt3 tcprt1 jane@tcprt1
250 ok
$ _
```

File

/dev/net0 Network device

Related Information

In this book: “**smtpd**” on page 3-21 and “**netmail**” on page 2-16.

tcom

Purpose

Controls the operation of **tftpd**.

Syntax

tcom **—** |

OLR20018

Description

The **tcom** command sends user commands to **tftpd** by writing them into a file (specified as standard output when **tftpd** starts) and signaling the daemon. **tcom** then monitors the **tftpd** log to obtain the results of the commands.

Subcommands

help	Displays a list of commands.
input_trace on off	Turns input packet tracing on or off.
output_trace on off	Turns output packet tracing on or off.
trace on off	Turns all packet tracing on or off.
time	Displays server parent and children process times.
uptime	Displays daemon start time.
exit	Forces daemon to shut down and exit.

tcom

Examples

1. To run interactively and trace all packets:

```
$ /etc/tftpd &
$ tcom trace on
$ tftp -w /etc/hosts smith /tmp/hosts netascii
.
.
.
(Display of all packets sent and received)
.
.
.
$ tcom trace off
$ -
```

2. To trace all packets and have the trace information redirected to a file:

```
$ /etc/tftpd > /tftpd/tftpd.log 2>&1 &
$ tcom trace on
$ tftp -w /etc/hosts smith /tmp/hosts netascii
$ tcom trace off
$ -
```

The **/tftpd/tftpd.log** file contains the trace data.

Files

<code>/etc/locks/tftp</code>	Lock file containing daemon PID
<code>/tftpd/command</code>	Command file to daemon
<code>/tftpd/slog</code>	Daemon log file

Related Information

In this book: “**tftpd**” on page 3-25.

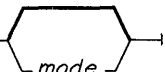
tftp

tftp

Purpose

Provides the Trivial File Transfer Protocol.

Syntax

tftp *action* *localname* *host* *foreignname*  *mode*

OLR20019

Description

The **tftp** command transfers files between hosts using minimal protocol. This command does not contain the features described under “**xftp**” on page 2-48. The transferred files are placed in the **/tftpd** directory or a file specified with a full path name. No command or utility is supplied to automatically process files in this directory.

To read input from a source other than a file (as in a pipe), use the **p** (put) action and a - (hyphen) for *localname*. To redirect output to standard output rather than to the **tftpd** directory, use the **g** (get) action and a - (hyphen) for *localname*.

If *action* is:

- w** or **p** Writes the local file, called *localname*, onto the file system of the foreign host as *foreignname*. Note that *foreignname* must be in quotations if it contains shell special characters.
- r** or **g** Reads file *foreignname* from the foreign host into the local file, *localname*. With *action* **r** or **g**, **tftp** prompts for verification before overwriting an existing local file. This characteristic can make it impractical to use **tftp g** or **tftp r** in a pipe. The **utftp** command performs the same **r** and **g** actions as **tftp**, but simply stops before overwriting a local file; thus, **utftp** can be more appropriate for use in a pipe.
- o** Supersedes or overwrites existing local files.

The *mode* parameter determines the form of the *foreignname* parameter. If *mode* is **netascii** or **image**, *foreignname* is a file name (the recipient’s mailbox). If *mode* is **mail**, *foreignname* is a user ID (that is, *user@host*).

The *mode* parameter can be **netascii**, **image**, or **mail**. These modes perform the following:

netascii Transfers the file as standard ASCII characters. This is the default.

image Transfers the file in binary, with no character conversion. **image** transfer is more efficient than **netascii** transfer.

mail Appends the transferred file onto the end of the user mailbox specified. After the file is appended, the user can retrieve it with the AIX **mail** command.

Examples

1. To transfer a binary file `/unix` to the directory `/tmp` on host `tcprt3`:

```
$ tftp -w /unix tcprt3 /tmp/unix image
Transfer successful
309295 bytes in 15 seconds, 164957 band
$ -
```

2. To copy the `/etc/hosts` file from host `tcprt3` and redirect the output to standard output of the local host:

```
$ tftp -g - tcprt3 /etc/hosts
192.9.200.3 nameserver
192.9.200.3 tcprt2
192.9.200.5 tcprt1
192.9.200.7 tcprt3
192.9.200.3 timeserver
Transfer successful
128 bytes in 1 second, 1024 band
$ -
```

tftp

File

`/dev/net0`

Network device

Related Information

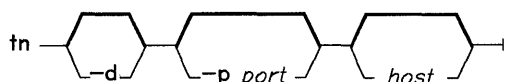
In this book: “**tftpd**” on page 3-25 and “**utftp**” on page 2-46.

tn

Purpose

Provides the Telnet interface for logging in to a foreign host.

Syntax



OLR20020

Description

The `tn` command implements the Telnet protocol, which allows remote login to other hosts. It uses the Transmission Control Protocol (TCP) to communicate with other hosts in the network.

PTYs for use with `tn` should be defined with the following characteristics:

auto enable **true**

terminal **hft** (for an RT PC-to-RT PC connection), **vt100**, or **dumb**.

logger **true**

When started with a `host` parameter, `tn` opens a Telnet connection to the specified host. If no `host` parameter is specified, `tn` prompts for a host name.

Once a connection is successfully opened, characters entered at the terminal are transmitted to the foreign host and acknowledged. The client and server communicate to determine whether to use a 3270 data stream. If the server supports the 3270 data stream, `tn` uses the keyboard mapping contained in `/etc/3270.keys` or, if it exists, `$(HOME)/.3270keys`.

If the value of the environment variable `EMULATE` is defined as `vt100`, `tn` emulates a DEC VT100¹ terminal. If `EMULATE` is not defined, or has a value other than `vt100`, `tn` operates normally. `EMULATE` is valid only for the console; if `EMULATE` is set for other than the RT PC console, results may be unpredictable. If `EMULATE` is set to `vt100`, the

¹ DEC VT100 is a registered trademark of Digital Equipment Corporation.

value of **TERM** = in the remote login connection should also be **vt100** (check this with the **env** command after the connection is open).

International Character Support Considerations

The Telnet protocol treats the decimal value 255 (hex FF) as a flag indicating that a control value follows. Therefore, the following characters can produce unpredictable results if sent across a Telnet connection:

- n acute (lowercase)
- ∴ (the *therefore* symbol).

Flags

- d Starts with debugging mode on.
- p *port* Opens the connection to the specified (decimal) foreign port number instead of the default Telnet port. The -p flag is useful for testing other commands; for example, you can specify the **smtpd** or **xftp** port, and use those daemons over the **tn** connection. For more information on ports, see “Port Numbers” on page 1-38.

Subcommands

To enter a character as a subcommand to **tn**, use **Ctrl-T** (the Telnet escape character). All **tn** subcommands are a single character:

- a Sends the Telnet **are you there** command. The foreign host should reply to indicate it is still connected.
- b Sends the Telnet **break** command in urgent mode. This is similar to typing the interrupt character to AIX.
- c Closes the Telnet connection and exits Telnet.
- d Toggles the debugging mode. The debugging mode is displayed on the screen when each packet is transmitted and received.
- e Transmits on every character typed. This is the default.
- l Sets local echo mode. Telnet attempts to negotiate the appropriate echo mode automatically. This subcommand allows you to override it.
- n Transmits after newline only. This reduces overhead for hosts, which only process data when a full line is typed.
- p Suspends **tn** and returns to the shell. To resume execution, press **Ctrl-D**.

-
- q** Quits Telnet immediately, without waiting for the connection to close. This is useful when the host you are talking to terminates the connection abnormally.
 - r** Requests remote echo. This is the default for hosts that support the **r** subcommand.
 - s** Displays the connection status on the terminal.
 - ?** Displays help for available Telnet commands.

Example

To log in to host1:

```
$ tn host1
Telnet escape character is ^T
Trying . . . Open
IBM RT PC Advanced Interactive Executive Operating System
55X8994 (C) COPYRIGHT IBM CORP. 1985, 1986
(/dev/pts0)
login: _
```

Files

/dev/net0	Network device
/etc/3270.keys	Default keyboard mapping (used when \$(HOME)/.3270keys does not exist)
\$(HOME)/.3270keys	Keyboard mapping for 3270 emulation

Related Information

In this book: “**telnetd**” on page 3-23.

pty in *AIX Operating System Technical Reference*.

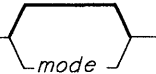
utftp

utftp

Purpose

Supports the Trivial File Transfer Protocol.

Syntax

utftp —*action* —*localname* —*host* —*foreignname* —

OLR20024

Description

The **utftp** command transfers files between hosts using minimal protocol. **utftp** is equivalent to **tftp** except that it does not overwrite a file; **tftp** can overwrite a file, but it prompts the user before doing so. Because it is not interactive, **utftp** can be more useful than **tftp** in a pipe. For more information, see “**tftp**” on page 2-40.

Example

The following command gets the file `/u/john/schedule` from the foreign host `host1` into the local file `newsched`, and pipes it to the `grep` command:

```
$ utftp -g newsched host1 /u/john/schedule | grep Jones
$ -
```

Related Information

In this book: “**tftp**” on page 2-40.

xftp

xftp

Purpose

Transfers files.

Syntax

xftp *-n* *host* *-dbg*

OLR20026

Description

The **xftp** command is the user interface to the File Transfer Protocol (FTP). This command allows a user to transfer files to and from a foreign network location. The **xftp** window size is 6K bytes; the packet size is 1576 bytes.

The name of the client *host* must be specified on the command line. The **xftp** command then prompts for the user's login ID and password. While **xftp** is waiting for the user to supply subcommands, the user sees the **xftp>** prompt. To end the **xftp** command, use the **quit** subcommand or the **xftp** break key sequence (**Ctrl-v**). Input to **xftp** can be redirected; that is, **xftp** can read input from a shell program rather than from the command line.

The default file transfer type is ASCII. However, binary file transfers can be more efficient. To set the file transfer type to binary, use the **binary** subcommand.

The signals **SIGHUP**, **SIGTERM**, and **SIGQUIT** cause **xftp** to close the control connection. The **SIGINT** signal causes **xftp** to stop any data transfer that is in progress. If no data transfer is in progress, **SIGINT** has no effect on **xftp**.

Flags

-n An optional flag that provides an automatic login feature. The **-n** flag causes **xftp** to search the **\$(HOME)/.netrc** file for the user's ID and password on the foreign host.

dbg An optional parameter that turns logging on or off, and specifies the type of logging to be done. The possible values for **dbg** are:

- 0** Gets trace messages.
- 1** Gets error messages.
- 2** Shows all received packets.
- 4** Shows all sent packets.

To perform more than one type of logging, add the appropriate **dbg** values together (for example, to show all received packets and all sent packets, use the **dbg** value **6**). **xftp** writes logging information to the **/usr/tmp/ftp.log** log file.

Subcommands

! (exclamation mark)

Invokes a shell on the local host.

acct Sends account information.

append *local-file* [*foreign-file*]

Appends a local file to a file on the foreign host. If *foreign-file* is not specified, the local file name is used in naming the foreign file. File transfer uses the current settings for *type*, *format*, *mode*, and *structure*.

ascii Sets the file transfer *type* to network ASCII. This is the default type. For more efficient file transfer, set the file transfer type to **binary**.

binary Sets the file transfer *type* to support binary image transfer.

cd *foreign-directory* Changes the working directory on the foreign host to *foreign-directory*.

cdup Changes to the parent directory.

delete *foreign-file* Deletes the file *foreign-file* on the foreign host.

dir [*foreign-directory*]

Displays a listing of the directory contents in the *foreign-directory* directory. If no directory is specified, **dir** lists the contents of the current foreign directory.

xftp

- get** [*local-file*] [*foreign-file*]
Retrieves the *foreign-file* and stores it on the local host. If you do not provide *local-file*, **xftp** gives it the same name it has on the foreign host. It uses the current settings for *type*, *form*, *mode*, and *structure* while transferring the file.
- help** [*command*]
Displays a message about the meaning of *command*. If you do not specify *command*, **xftp** displays a list of known commands.
- lcd** [*directory*]
Changes the working directory on the local host. If you do not specify a *directory*, **xftp** uses your home directory.
- ls** [*foreign-directory*]
Displays an abbreviated listing of the contents of a directory on the foreign host. If you do not specify *foreign-directory*, **xftp** uses the current directory.
- mdelete** [*foreign-files*]
Deletes multiple files.
- mget** [*foreign-files*]
Gets multiple files.
- mkdir** [*foreign-directory*]
Creates the directory *foreign-directory* on the foreign host.
- mode**
Sets file transfer mode. The only mode available is **stream**.
- mput** [*local-files*]
Puts multiple files.
- noop**
Sends **noop** command.
- pass** *password*
Sends password information.
- put** *local-file* [*foreign-file*]
Stores a local file on the foreign host. If you do not specify *foreign-file*, **xftp** uses the local file name to name the foreign file. File transfer uses the current settings for *type*, *format*, *mode*, and *structure*.
- pwd**
Displays the name of the current directory on the foreign host.
- quit**
Ends the FTP session with the foreign server and exits **xftp**.
- quote** [*line*]
Quotes the next line to be sent; that is, the next line is sent literally. Quoting commands that involve data transfer can produce unpredictable results.
- rename** *from-name* *to-name*
Renames a file on the foreign host.
- rmdir** [*foreign-directory*]
Removes the directory *foreign-directory* on the foreign host.

sendport	Toggles the use of port commands. By default, xftp uses a port command when establishing a connection for each data transfer. When the use of port commands is disabled, xftp does not use port commands for data transfer. This is useful for FTP implementations, which ignore port commands while incorrectly indicating they have been accepted.
status	Displays current status of the connection.
struct	Sets data transfer structure type. The only structure supported is file .
type [<i>type-name</i>]	Sets the file transfer type to <i>type-name</i> . If no type is specified, the current type is printed. The default type is network ASCII; the type binary can be more efficient.
user <i>user-name</i>	Allows a different user ID to be entered. If a user attempts to log in with an invalid ID, xftp starts and then displays a message indicating that the user is not recognized. At that point, the user can enter user <i>user-name</i> , where <i>user-name</i> is a user ID different from the one originally entered.

xftp

Examples

1. The user `smith` is logged in on `host1`. This example shows how `smith` can log in as the user `tcpip` (a password protected account) on the foreign host `host2`, and then transfer the file `/unix` to `/tmp/unix`:

```
$ xftp host2
connected to host2
220 host2 c05 FTP Server, Sat Mar 15 14:42:21 1986
Name (host2:smith) tcpip
331 Need password for user tcpip
Password (host2:tcpip) <enter password>
230 User tcpip logged in
xftp> binary
200 Command ok
xftp> put /unix /tmp/unix
200 Command ok
150 Opening data connection for /tmp/unix (192.9.200.1,1016)
308310 bytes sent in 3.58 seconds (85.71 Kbytes/s)
226 Closing data connection; requested file action successful

xftp> quit
221 Quit command received. Goodbye.
Connection to host2 closed
$ -
```

2. The user `smith` is logged in on `host1`. This example show how `smith` can log in as the user `smith` (not a password protected account) on the foreign host `host2`:

```
$ xftp host2
connected to host2
220 host2 c05 FTP Server, Sat Mar 15 14:49:01 1986
Name (host2:smith) <enter>
230 User smith logged in
xftp>
```

3. The user smith makes a typing error and tries to log in as the user miths:

```
$ xftp test
220 test C17 FTP Server, Fri May 9 08:26:25 1986
Name (test: root) miths
530 User miths unknown
ftp > user smith
230 User smith logged in
xftp>
```

Files

<code>/dev/net0</code>	Network device
<code>/usr/tmp/ftp.log</code>	Log file for protocol interpretation process of xftp

Related Information

In this book: “**xftpd**” on page 3-27.

xftp

Chapter 3. Server Commands

CONTENTS

About This Chapter	3-2
fingerd	3-3
icmpd	3-5
lpd	3-7
maild	3-11
named	3-13
rexecd	3-15
routed	3-17
smtpd	3-21
telnetd	3-23
tftpd	3-25
timed	3-26
xftpd	3-27

About This Chapter

This chapter describes the IBM RT PC Interface Program for use with TCP/IP server commands, or daemons. Server commands provide support for user commands (described in Chapter 2, "User Commands" on page 2-1). Server commands should be started at system start by the `/etc/rc.tcpip` file, and usually are not entered on the command line.

- File transfer
 - tftpd**
 - xftpd**
- Mail
 - maild**
 - smtpd**
- Remote login, command execution, and printing
 - lpd**
 - rexecd**
 - telnetd**
- Network management
 - fingerd**
 - icmpd**
 - named**
 - routed**
 - timed**

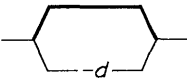
In this chapter, the server commands are organized alphabetically.

fingerd

Purpose

Provides the server function for the **finger** command.

Syntax

fingerd 

OLR20021

Description

The **fingerd** command is a server for the **finger** command. It returns information about a particular user at the host, or about all users logged in at the host.

Flag

-d Writes debugging messages to the **/usr/tmp/fingerd.log** file.

Files

/dev/net0	Network device
/etc/locks/finger	Interlock, PID storage
/etc/passwd	Password file
/etc/utmp	who file
\$HOME/.plan	Plans for requested user
\$HOME/.project	Projects for requested user
/usr/bin/whois	whois command
/bin/who	who command
/usr/tmp/fingerd.log	Debugging message log file

fingerd

Related Information

In this book: “**finger**” on page 2-3.

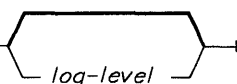
icmpd

Purpose

Provides the server function for Internet Control Message Protocol and the Gateway-to-Gateway Protocol.

Syntax

```
icmpd -{ log-level }
```



OLR20009

Description

The **icmpd** command is a daemon that deals with sending and processing Internet Control Message Protocol (ICMP) and Gateway-to-Gateway Protocol (GGP) packets. The ICMP and GGP provide network management, routing, and error notification functions. This daemon performs the following functions:

- Sends ICMP GGP echoes, timestamp packets, and subnet masks and receives the appropriate replies. These functions aid in network management and debugging. Sending of echoes, timestamping packets, and subnet mask packets is initiated by using the **ping** command.
- Displays received ICMP redirect packets.

Messages that the daemon was started or stopped are written to standard output. The **icmpd** file in the **/etc/locks** directory is used to prevent two daemons from being active simultaneously. It also contains the daemon process ID, which **ping** uses to wake up the daemon when it has work to do.

icmpd

icmpd accepts only one parameter, `log-level`, to indicate what logging is to be done. The following `log-level`s are defined:

- 0 Logs all ICMP and GGP errors.
- 1 Logs all locally originated ICMP and GGP transactions.
- 2 Logs all foreign originated ICMP and GGP transactions.
- 3 Logs all transmitted packets.
- 4 Logs all received packets.

All logging goes to standard output, but can be redirected at the time the **icmpd** daemon is started. If no `log-level` parameter is present, no logging is done. When the **icmpd** daemon is running, `log-level` may be changed dynamically with one of the following **kill** commands (where *pid* is the process ID number of the **icmpd** daemon):

- kill -20** *pid* Changes `log-level` to 0.
- kill -21** *pid* Changes `log-level` to 1.
- kill -22** *pid* Changes `log-level` to 2.
- kill -23** *pid* Changes `log-level` to 3.
- kill -24** *pid* Changes `log-level` to 4.

Files

<code>/etc/locks/icmp</code>	Interlock, PID storage
<code>/dev/net0</code>	Network device

Related Information

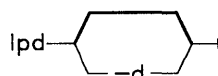
In this book: “**ping**” on page 2-23.

lpd

Purpose

Provides the server function for remote printing.

Syntax



OLR20034

Description

The **lpd** command (the remote print server) monitors port 515 for print requests. Each request is placed in the **/usr/spool/lpd** directory. A print request consists of three parts:

- Control file
- Data
- Status file (AIX controls).

A host that can create a TCP/IP data stream and handle the **lpd** protocol can print remotely or act as a print server. As a security feature, **lpd** accepts print requests only from hosts that are listed in the **/etc/hosts.equiv** file of the host it runs on.

The **lpd** daemon may run on any host in the network; its function is to accept print requests from foreign hosts (port 515). **lpd** handles each request by forking a child process. Remote requests are first checked against the **/etc/hosts.equiv** file for permission to print on the local host.

In the **lpd** protocol, the first byte of a message indicates which action is to be performed, and the remaining characters are taken as arguments. **lpd** recognizes the following four

types of requests:

Request	Meaning
<code>\2printer\n</code>	Receive and queue job
<code>\3printer [users . . .] [jobs . . .]\n</code>	Short list of queue
<code>\4printer [users . . .] [jobs . . .]\n</code>	Long list of queue
<code>\5printer person [users . . .] [jobs . . .]\n</code>	Remove jobs from queue

Figure 3-1. lpd Requests

Note: In Figure 3-1, `\x` is the standard C language notation of characters, the spaces are ASCII 32, and material enclosed in `[]` is optional.

The **lprbe** command creates three files in the spooling directory for each queued job. The names of these files are unique for each job, consisting of a special prefix for each type of file, a job sequence number, and the name of the originating host. The format for a spooling directory file name is:

spooling_directory/xfA#host

where:

x is one of the following prefixes:

- c** Control file
- d** Data file
- t** Temporary file

is the sequence number of the job.

host is the name of the originating host.

The file name in the following example corresponds to the control file, sequence number 124, from a job printed on the **lp** line printer queue on `host1`:

```
/usr/spool/lpd/cfA124host1
```

The three types of files created in the spool directory are:

Control A file that contains information about the job (for example, who the job is for, which host is to print the job, and what information should go on the title page). Each line of the control file is of the form `xstring`, where **x** is an element of the set shown in Figure 3-2 on page 3-9.

Data A file that contains a copy of the file to be printed or a symbolic link to that file.

Temporary A file used to create the control and data files. This file is discarded once the job is queued.

Figure 3-2 lists the control codes processed by AIX:

C **class name** on banner page
f **file name**, text file to print
H **host name** of host where **lprbe** was run
I **indent**, amount to indent output, for **pr**
J **job name** on banner page
L **literal** name of user to print on banner
N **name** of file (used by **lpq**)
P **person**, the login ID of the user
p **file name**, text file to print with **pr**
T **title** for **pr**
U **unlink**, name of file to remove after printing
W **width**, width to print output, for **pr**

Figure 3-2. lpd Control File Codes

If **-naix** is not specified, AIX keywords of the form **-xxx=sss** are appended to the control file.

Following is a sample control file created when user jim prints the `/etc/rc.tcpip` file on `host1`:

```
host1
Pjim
Jrc.tcpip
Chost1
Ljim
fdfA123host1
UdfA123host1
N/etc/rc.tcpip
```

The local print client opens the known printer port, 515, on the remote host and sends the `\2printer\n` message, indicating to the service that it should prepare to receive a job. If the remote daemon does not respond with the message `\0`, there is some problem with the request; otherwise, the local daemon sends the control file to the remote daemon. Communication between the daemons is accomplished with lines of text if the first byte indicates what action is to be taken:

`\1` Abandon the request; some problem prevents its completion.
`\2` Read the control file.
`\3` Read the data file.

lpd

The control file contains the names of the data files to be queued to print on the remote host. When the control file is sent, the local files are deleted. The remote **lpd** daemon starts a process to print the files. At this point, the local processing is finished.

Flag

-d Provides debugging information in the `/usr/tmp/lpd.log` file.

Examples

1. To start the **lpd** server daemon:

```
# /etc/lpd &  
[process ID number]  
# -
```

2. To start the **lpd** server daemon with the debugging option:

```
# /etc/lpd -d &  
[process ID number]  
# -
```

Files

<code>/dev/lp*</code>	Print devices
<code>/dev/net0</code>	Network device
<code>/etc/hosts.equiv</code>	Names of hosts allowed to print
<code>/etc/locks</code>	Interlock, PID storage
<code>/usr/spool/lpd</code>	Spool directory for control, status, and data files

maild

Purpose

Provides a mail relay server function for the Simple Mail Transfer Protocol.

Syntax

```
maild -d timeout
```

OLR20029

Description

The **maild** command is a relay program for SMTP. All mail to be relayed is stored in the **/maild/smtp** directory, and **maild** periodically scans **/maild/smtp** for ***.reg** files. **maild** then updates forward and reverse paths and invokes **smtp** to send the mail.

An optional parameter, *timeout*, sets the length of time (in seconds) that **maild** is to wait between scans of **/maild/smtp**. The default value of *timeout* is 600 (10 minutes). An optional flag, **-d**, causes **maild** to produce debugging messages and write them to standard output.

At each scan of **/maild/smtp**, **maild** attempts to deliver stored mail. After 100 unsuccessful attempts to deliver a particular mail file, **maild** notifies the sender that the file remains undelivered and returns the mail to the sender.

Flags

-d	Causes maild to write debugging messages to standard output
<i>timeout</i>	Sets the amount of time (in seconds) to wait between scans of the /maild/smtp directory.

maild

Examples

1. To start **maild** with the default *timeout* value:

```
# /etc/maild &  
# -
```

2. To start **maild** with a *timeout* value of 15 minutes (900 seconds) and debugging messages written to the file **/usr/tmp/maild.log**:

```
# /etc/maild -d 900 >>/usr/tmp/maild.log 2>&1 &  
[process ID number]  
# -
```

After 25 hours (100 delivery attempts at 15-minute intervals), mail that has not been delivered is returned to the sender.

Files

/etc/locks	Interlock, PID storage
/maild/smtp	Relay information for mail file
/maild/smtp/t*	Mail file
/maild/smtp/t*.reg	Directory for relaying mail

Related Information

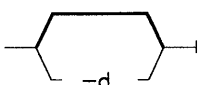
In this book: “**smtp**” on page 2-34.

named

Purpose

Provides the server function for the Name Server Protocol.

Syntax

named 

OLR20010

Description

The **named** command is the server for the Name Server Protocol. **named** uses a UDP connection port. It uses the **/etc/hosts** file to resolve name to address mapping. To put changes to the **/etc/hosts** file into effect without restarting the system, send **SIGINT (kill -2 PID)**; this causes the **/etc/hosts** data base to be reloaded.

Note: Generally, **named** should not run on every host in the network.

Flag

- d** Provides a debugging option. The **-d** flag causes **named** to write debugging information to standard output for each packet generated.

named

Files

/dev/net0	Network device
/etc/hosts	Name to address map table
/etc/hosts.dir	Database file built from /etc/hosts for name resolution
/etc/hosts.pag	Database file built from /etc/hosts for name resolution
/etc/locks/name	Interlock, PID storage

Related Information

In this book: “**host**” on page 2-6.

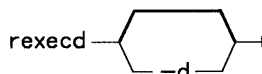
rexecd

Purpose

Provides the server function for the **rexec** command.

Syntax

rexecd -d



OLR20033

Description

The **rexecd** command is the server for the **rexec** command; it processes commands issued by a foreign host and returns the output of those commands to the foreign host. **rexecd** sends and receives data over a TCP connection.

Note: The host running **rexecd** must have available three PTYs defined without loggers; they are used for standard input, standard output, and standard error.

The **rexecd** daemon listens for requests at port 520. When **rexecd** receives a request, it initiates the following protocol:

1. The server reads characters from the socket up to a null (\0) byte and interprets the resulting string as an ASCII number (decimal).
2. If the number received is non-zero, **rexecd** interprets it as the port number of a secondary stream to be used for standard error output. **rexecd** then creates a second connection to the specified port on the client machine.
3. A null terminated user name of up to 16 characters is retrieved on the initial socket.
4. A null terminated, encrypted password of up to 16 characters is retrieved on the initial socket.
5. A null terminated command to be passed to a shell is retrieved on the initial socket. The length of the command is limited by the upper bound on the size of the system's argument list.
6. **rexecd** validates the user (as is done at log in) and, if successful, changes the user's home directory and establishes the user and group protections of the user. (If any part

rexecd

of this procedure fails, **rexecd** abandons its attempt to create the connection and returns a diagnostic message.)

7. A null byte is returned on the connection associated with standard error and the command line is passed to the normal login shell of the user. The shell inherits the network connections established by **rexecd**.

Flag

- d** An optional flag that provides a debug service. The **-d** flag causes **rexecd** to write debugging statements to standard output.

Examples

1. To start the **rexec** server:

```
# /etc/rexecd &  
[PID]  
# -
```

2. To start the server with the debug option and redirect the debugging statements to the log file, **/usr/tmp/rexecd.log**:

```
# /etc/rexecd -d 1>>/usr/tmp/rexecd.log 2>&1 &  
# -
```

Files

/dev/net0	Network device
/etc/locks	Interlock, PID storage
/dev/ptc*	PTY controller device
/dev/pts*	PTY server device

Related Information

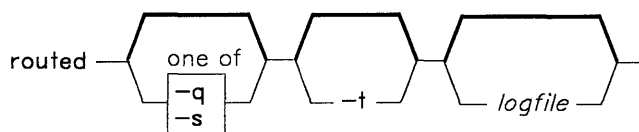
In this book: “**rexec**” on page 2-26.

routed

Purpose

Manages network routing tables.

Syntax



OLR20036

Description

The **routed** command is a daemon that manages the network routing tables. It is started at system start time. **routed** uses a variant of the Xerox NS Routing Information Protocol¹ to maintain current kernel routing table entries. Ordinarily, **routed** listens on UDP socket 520 (decimal) for routing information packets. If a host is an internetwork router, it periodically supplies copies of its routing tables to any directly connected hosts and networks.

When **routed** starts, it finds any interfaces to directly connected hosts and networks that are configured into the system and marked as **up**. If multiple interfaces are present, **routed** assumes that the host forwards packets between networks. Once it has determined which interfaces are configured, **routed** transmits a request packet on each interface (using a broadcast packet if the interface supports it) and then enters a loop, listening for request and response packets from other hosts.

When it receives a request packet, **routed** generates a reply (response packet) based on the information maintained in its internal tables. The response packet contains a list of known routes, each marked with a *hop count metric* (the number of host-to-host connections in the route). The metric for each route is relative to the sender. A metric of 16 or greater is considered to be *infinite*, or beyond reach.

If any one of the following conditions exists, **routed** uses the information contained in

¹ Defined in *Internet Transport Protocols*, XSIIS 028112, Xerox System Integration Standard.

routed

response packets to update the routing tables:

- No routing table entry exists for the destination network or host, and the metric associated with the route is finite (that is, the metric is less than 16).
- The source host of the packet is the same as the router in the existing routing table entry. That is, updated routing information is being received from the same internetwork router through which packets for the destination are being routed.
- The existing entry in the routing table has not been updated for some time and the route is at least as efficient as the current route.
- The new route is shorter than the one to the same destination that is currently stored in the routing tables. (**routed** determines relative route length by comparing the new metric with the one stored in the routing table.)

When **routed** updates its internal routing tables, it generates a response packet to all directly connected hosts and networks. Before updating the kernel routing tables, **routed** pauses for a brief period to allow any unstable conditions to stabilize.

Besides processing incoming packets, **routed** also checks the routing table entries periodically. The metric for any entry that has not been updated for 3 minutes is set to infinity and marked for deletion. The deletion is delayed for 60 seconds so that information about the invalidated route can be distributed throughout the network.

A host that acts as an internetwork router supplies its routing tables to all directly connected hosts and networks every 30 seconds.

Besides its ability to manage routes involving directly connected hosts and networks, **routed** also employs the concept of *distant* gateways, both *passive* and *active*. When it starts, **routed** reads the `/etc/gateways` file for information about gateways which may not be identified by querying. If a gateway specified in this way exchanges routing information (that is, it runs its own **routed** process), it should be marked active. A gateway that does not exchange routing information should be marked passive. Passive gateways are maintained in the routing tables indefinitely and information about them is included in any routing information transmitted. Active gateways are treated as if they were network interfaces; that is, routing information is distributed to the active gateway and, if no routing information is received from the gateway for a period of time, **routed** deletes the associated route. For information about `/etc/gateways`, see “**gateways**” on page 1-23.

Flags

- q** Prevents **routed** from supplying routing information whether it is functioning as an internetwork router or not. (**-q** is the opposite of **-s**; do not use these two flags together.)
 - s** Causes **routed** to supply routing information whether it is functioning as an internetwork router or not. (**-s** is the opposite of **-q**; do not use these two flags together.)
 - t** Causes all packets sent or received to be written to standard output. **routed** remains under control of the host that started it; therefore, an interrupt from the controlling host kills the **routed** process.
- logfile* Specifies the file in which **routed** writes the log information about its actions. The log contains information about any changes to the routing tables and a history of recent messages sent and received which are related to the changes route.

Examples

1. To start **routed** (normally done automatically at system start) and cause it to return routing information whether or not it is functioning as an internetwork router, enter:

```
# /etc/routed -s &  
[PID]  
# -
```
2. To start **routed** without causing it to return routing information, enter:

```
# /etc/routed -q &  
[PID]  
# -
```
3. To start **routed** and cause it to write all packets sent or received to standard output, enter:

```
# /etc/routed -t &  
[PID]  
# -
```
4. To start **routed** and cause it to write its logging information to the file `routlog`, enter:

```
# /etc/routed routlog &  
[PID]  
# -
```

routed

File

`/etc/gateways`

Distant gateways

Related Information

In this book: “**route**” on page 2-29.

smtpd

Purpose

Provides the server function for the Simple Mail Transfer Protocol.

Syntax

```
smtpd -{ log-level }
```

OLR20016

Description

The **smtpd** daemon runs the Simple Mail Transfer Protocol (SMTP) server. It is integrated into the AIX mail system and suitable for receiving mail from other hosts in the network. When mail arrives, the mail daemon queues it for processing. The **mail** command is used to process mail sent via **smtp**.

The daemon listens for incoming connection requests on a TCP connection port and performs a **fork** call to create a child process to handle each connection request. Incoming messages are received into temporary files in the mail daemon's directory. A mail daemon file is created and the mail daemon is awakened to process it.

The **smtpd** daemon uses the **/maild** directory. The **smtp** file in **/etc/locks** prevents two daemons from becoming active simultaneously.

The *log-level* parameter indicates the level of logging that **smtpd** is to perform. The *log-level* parameter can have one of the following values:

- 1 Logs data sent to the network and received from it.
- 2 Performs full packet tracing.
- 3 Performs the tasks of *log-level* 1 and 2 together.

The **smtpd** daemon supports a number of requests from the **smtp** command. Following is a list of those requests together with a brief explanation their actions:

- DATA** Treats following lines as mail data from sender.
- HELP** Sends helpful information from receiver to sender of **HELP**.
- MAIL** Initiates mail transaction.

smtpd

NOOP	Takes no action.
QUIT	Closes receiver of transmission channel.
RCPT	Identifies individual recipient of mail data.
RSET	Stops current mail transaction.
VRFY	Confirm that the received parameter identifies a user.
HELO	Identifies sender to receiver.
EXPN	Not supported.

Files

/bin/sig_maid	Dispatcher of mail received by smtpd
/dev/net0	Network device
/etc/locks/smtp	Interlock, PID storage

Related Information

In this book: “**netmail**” on page 2-16 and “**smtp**” on page 2-34.

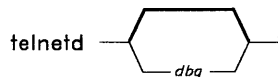
The **mail** command in *AIX Operating System Commands Reference*.

telnetd

Purpose

Provides the server function for the Telnet protocol.

Syntax



OLR20003

Description

The **telnetd** command is a server that supports the DARPA standard Telnet virtual terminal protocol. It operates at the assigned port 23. (For information about assigned ports, see “Port Numbers” on page 1-38.)

The **telnetd** server assumes that **getty** has opened the server side of a pseudo-terminal device. The server opens the client side, thus allowing **getty** to execute a login process.

When a Telnet session is started, **telnetd** sends a Telnet option to the client side indicating it will perform remote echo of characters. Aside from this initial setup, the only mode changes **telnetd** will carry out are those required for echoing characters at the client side of the connection. A host running the **telnetd** daemon must have pseudo teletype (PTY) devices defined as described under “**tn**” on page 2-43. PTY lines are configured with the **devices** command.

The optional parameter, *dbg*, turns logging on or off, and specifies the type of logging to be done. The possible values for *dbg* are:

- 1 Gets error messages.
- 2 Shows all received packets.
- 4 Shows all sent packets.

To perform more than one type of logging, add the appropriate *dbg* values together (for example, the *dbg* value **6** shows all received packets and all sent packets). The log file is:

/usr/tmp/telnetd.log

telnetd

The **telnetd** server supports the following options:

- Echo/no echo
- Timing marks
- Suppress go ahead.

telnetd also supports the following commands (available as subcommands of the **tn** command):

break
abort output
are you there?
erase character

Files

/dev/net0	Network device
/etc/ports	File describing configured ports
/usr/tmp/telnetd.log	Log file

Related Information

In this book: “**tn**” on page 2-43.

The **devices** command in *AIX Operating System Commands Reference*.

The **pty** device driver in *AIX Operating System Technical Reference*.

tftpd

Purpose

Provides the server function for the Trivial File Transfer Protocol.

Syntax

tftpd —l

OLR20017

Description

The **tftpd** daemon runs the Trivial File Transfer Protocol (TFTP) server. It listens for incoming connections and performs a **fork** call to create a child to perform each requested transfer. Files sent using TFTP can be found in the **/tftpd** directory or the file specified with a full path name. No command is provided to automatically extract files in this directory. Files in the **/etc/locks** directory prevent two daemons from becoming active simultaneously. It also contains the daemon process ID, which the **tcom** command uses to control the operation of the daemon.

Files

/dev/net0	Network device
/etc/locks/tftp	Interlock, PID storage

Related Information

In this book: “**tftp**” on page 2-40 and “**tcom**” on page 2-37.

timed

timed

Purpose

Provides a network time service.

Syntax

timed—l

OLR20023

Description

The **timed** command returns a 32-bit time value. The value represents time as the number of seconds since 12:01:01 a.m., January 1, 1970.

Files

/dev/net0	Network device
/etc/locks/timed	Interlock, PID storage

Related Information

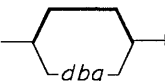
In this book: “**setclock**” on page 2-32.

xftpd

Purpose

Provides the server function for the **xftp** command.

Syntax

xftpd 

OLR20027

Description

The **xftpd** command is the server process for the File Transfer Protocol (FTP).

xftpd interprets file names according to the conventions the **sh** command uses. This allows using metacharacters like " (open double quotes), " (close double quotes), * (asterisk), ? (question mark), [(left bracket), and] (right bracket).

xftpd authenticates users according to these rules:

- The user name must be in the password data base, **/etc/passwd**. The client must provide the password (if the password is not null) before any file operations are performed.
- A non-active connection is dropped after 30 minutes.

The optional parameter, *dbg*, turns logging on or off, and specifies the type of logging to be done. The possible values for *dbg* are:

- 0** Gets trace messages.
- 1** Gets error messages.
- 2** Shows all received packets.
- 4** Shows all sent packets.

To perform more than one type of logging, add the appropriate *dbg* values together (for

xftpd

example, the *dbg* value **6** shows all received packets and all sent packets). The log file is:
`/usr/tmp/ftpd.log`

Requests

The **xftpd** server supports the following FTP requests:

ALLOC	No operation.
ABORT	Aborts transaction.
APPE	Appends to a file.
CDUP	Changes to the parent directory.
CWD	Changes current directory.
DELE	Deletes a file.
LIST	Gives list files in a directory (<code>ls -ls</code>).
MKD	Creates a directory on the foreign host
MODE	Specifies data transfer <i>mode</i> .
NLST	Gives name list of files in directory (<code>ls</code>).
NOOP	Does nothing.
PASS	Specifies password.
PORT	Specifies data connection port.
PWD	Prints the current working directory.
QUIT	Terminates session.
RETR	Retrieves a file.
RMD	Removes a directory on the foreign host.
RNFR	Specifies rename-from file name.
RNTO	Specifies rename-to file name.
STOR	Stores a file.
STRU	Specifies data transfer <i>structure</i> .
TYPE	Specifies data transfer <i>type</i> .
USER	Specifies user name.

The remaining **xftpd** requests specified in *File Transfer Protocol*, RFC 959, are recognized, but not implemented.

Files

<code>/dev/net0</code>	Network device
<code>/etc/locks/ftpd</code>	Interlock, PID storage
<code>/etc/passwd</code>	Password file.
<code>\$(HOME)/.netrc</code>	User IDs and passwords for remote login.
<code>/usr/tmp/ftpd.log</code>	Log file for protocol interpretation process

Related Information

In this book: “**xftp**” on page 2-48.

The **passwd** file in *AIX Operating System Technical Reference*.

Chapter 4. Protocol Library Routines

CONTENTS

About This Chapter	4-2
tcpm	4-3
Routines	4-3
User-Defined Routines	4-9
The Interface Program Tasking System	4-11
udp	4-14
Routines	4-15
Macros and Type Definitions	4-17
internet	4-19
Routines	4-20
Macros and Type Definitions	4-22
libS	4-24

About This Chapter

This chapter describes the libraries that contain the routines used by the Interface Program. The routines in these libraries also can be used by programmers developing their own applications; that is, the libraries provide the Application Programming Interface (API) to the Interface Program. For a description of the **tcp** library, which supports a single **tcp** connection, see Appendix C, “**tcp** Library Routines” on page C-1.

The following **include** files are in the **/usr/include** directory:

- in_extern.h**
- in_params.h**
- ip.h**
- netioctl.h**
- notice.h**
- udp.h**
- task.h**
- taskm.h**
- q.h**

Comment lines in these files explain their functions.

Note: In this chapter, the term *socket* or *foreign socket* refers to the 16-bit port number (not to the IP socket, which is the port number concatenated with the IP address).

tcpm

Purpose

Provides the layer that supports multiple TCP connections.

Library

`/usr/lib/libtcpm.a`

Syntax

```
#include <ip.h>
#include <taskm.h>
```

Description

The **tcpm** library provides support for multiple **tcp** connections, up to a maximum of 32 per process. The Interface Program supports a window size of 6K bytes and a packet size of up to 1576 bytes.

The **tcpm** library provides some support for the internal tasking system within a single AIX process. Task calls must be issued by the user. For more information about the tasking system, see “The Interface Program Tasking System” on page 4-11.

The routines in the `/usr/lib/libtcpm.a` library interface with the IP library and the AIX network driver. The `libtcpm.a` library contains the following routines:

Routines

```
int tcp_init (stacksize)
int stacksize;
```

Initializes TCP, timers, and the tasking system, and sends and receives tasks. *stacksize* is the size of the stack of the current task. `tcp_init ()` returns **TRUE** if initialization succeeds, **FALSE** if it fails.

tcpm

```
int tcp_open (fh, lh, fs, ls, win, us_procs)
    long fh;          /* foreign host to open to */
    long lh;          /* local host to open to */
    short fs;         /* foreign socket to open to */
    short ls;         /* local socket to open to */
    int win;          /* window size, see below */
    int (*us_procs[]) ( ); /* 6 user defined
procedures, see below */
```

Immediately tries to open a connection to foreign host (*fh*) on foreign socket (*fs*) from the local socket (*ls*). If the local socket is zero, **tcp_open ()** selects an unused socket number greater than 1000. If the value for local host is 0, **tcp_open** selects the optimal interface from the ones that are configured.

Note: This routine returns immediately. To determine whether the connection is open, your current task must block/yield until the **us_open ()** routine completes. (**us_open ()** is described under “User-Defined Routines” on page 4-9.)

A small, non-negative number (the connection ID) is returned if the routine opens an Internet connection with the specified hosts and sockets. If the connection is not opened, or if no more connection blocks are available, the routine returns -1.

```
int tcp_passive_open (fh, fs, ls, win, us_procs)
    long fh;          /* foreign host to open to */
    short fs;         /* foreign socket to open to */
    short ls;         /* local socket to open to */
    int win;          /* window size, see below */
    int (*us_procs[]) ( ); /* 6 user defined procedures, see below */
```

Performs a passive open on the specified foreign host, foreign socket, and local socket; that is, the process will accept incoming connection requests rather than attempting to initiate a connection. Both the foreign host and foreign socket may be zero.

Note: This routine returns immediately. To determine whether the connection is open, your current task must block/yield until the **us_open ()** routine completes. (**us_open ()** is described under “User-Defined Routines” on page 4-9.)

A small, non-negative number (the connection ID) is returned if the routine opens an Internet connection with the specified hosts and sockets. If the connection is not opened, or if no more connection blocks are available, the routine returns -1.

```
int tcp_listen(ls, win, us_procs)
short  ls;          /* local host to open to */
int    win;         /* window size, see below */
int    (*us_procs[])(); /* 6 user defined procedures, see below */
```

Listens for a server connection on the specified local socket with the foreign host and port unspecified. A small, non-negative number (the connection ID) is returned if the routine opens an Internet connection with the specified hosts and sockets. If the connection is not opened, or if no more connection blocks are available, the routine returns **-1**.

A child process runs the connection while the parent process returns to listen on the connection. A double fork makes the child process a child process of **/etc/init**; thus, clean up can be done after the process exits without the parent process having to wait for the exit. To determine whether the connection is open, the child process must block/yield until the **us_open** routine completes; **us_open** is described under “User-Defined Routines” on page 4-9.

```
int tcp_puts (con, str)
int  con;    /* connection id */
char *str;   /* string to output */
```

Functions like to the AIX **puts** routine except that the network connection must be specified. This routine returns **FALSE** if there is not enough room for the entire string in the TCP send buffer; the task that issues this call should either block or yield until room in the TCP send buffer becomes available.

Note: This routine does not awaken the send task (to send the character) unless the TCP send buffer is full. To force the character to be sent, use the **tcp_flush ()** routine.

```
int tcp_putc (con, c)
int  con;    /* connection id */
char c;      /* character to output */
```

Tries to write the character to the network. This routine returns **FALSE** if the TCP send buffer is full, **TRUE** otherwise.

Note: This routine does not awaken the send task (to send the character) unless the TCP send buffer is full. To force the character to be sent, use the **tcp_flush ()** routine.

tcpm

```
int tcp_write (con, buf, len)
    int  con;      /* connection id          */
    char *buf;     /* buffer to output      */
    int  len;     /* length of buffer     */
```

Functions like to the AIX `write ()` routine except that the network connection must be specified. This routine returns **FALSE** if there is not enough room for the entire string in the TCP send buffer.

Note: This routine does not awaken the send task (to send the character) unless the TCP send buffer is full. To force the character to be sent, use the `tcp_flush ()` routine.

```
int tcp_close (con)
    int  con;      /* connection id          */
```

Closes the local side of the current connection by sending **FIN** to the foreign host. This routine returns immediately, but the user must wait for the user defined procedure `us_close ()` to be called to indicate that the connection is completely closed.

```
int tcp_urgent (con)
    int  con;      /* connection id          */
```

Indicates that all data in the TCP send buffer is urgent. Subsequent `tcp_urgent ()` calls can be made, and they extend the length of the urgent data portion.

```
int tcp_clean (con)
    int  con;      /* connection id          */
```

Erases the connection. This routine can eliminate listening TCP connections or active TCP connections when the foreign host is down or does not respond.

```
int tcp_abort (con)
    int  con;      /* connection id          */
```

Sends **TCP RESET** to the foreign host and calls the procedure `us_close ()` with the value of `reason` set to **FALSE**. This routine returns immediately, so the user must wait until `us_close ()` is called before the reset is complete.

```
int tcp_flush (con)
    int  con;      /* connection id          */
```

Tries to send all outstanding data in the TCP send buffer to the foreign host.

```
int tcp_nput (con)
    int con;          /* connection id          */
```

Returns the amount of free space in the TCP send buffer. This number can be used later to determine how much data a user can put in the TCP send buffer.

```
int tcp_setu (con, ucb)
    int  con;          /* connection id          */
    char *ucb;         /* user control block     */
```

```
char *tcp_getu (con)
    int  con;          /* connection id          */
```

Two routines that make it possible to set and access a user-defined character pointer field for the connection. The value given to this field can be a pointer to the task that will send data through the network. The pointer is supplied to `tk_wake ()` when the `us_buf ()` call occurs.

```
tcp_new_window (con, window)
    int  con;          /* connection id */
    int  window;       /* window size   */
```

Changes the value of the local window. This routine should be used primarily to open a local window that has shrunk to zero; it is the only way to do so. If this routine is used at other times, poor performance and confusion from the foreign host may result.

```
conn_info (con, plhost, pfhost, plsock, pfsock)
    int  con;          /* connection id          */
    long *plhost, *pfhost; /* local host, foreign host */
    short *plsock, *pfsock; /* local socket, foreign socket */
```

Gets complete information about the connection: local host, foreign host, local port, and foreign port. If the user does not need information about a particular argument (for example, foreign host), he should pass NULL as the argument.

tcpm

```
tcp_debug (con, dbg)
    int  con;      /* connection id */
    int  dbg;      /* level of debug, valid are 0, 1, 2;
                   are additive */
extern char *logfname; /* log file name for trace output
                        0 = trace messages
                        1 = error messages
                        2 = shows received packets
                        3 = shows sent packets */
extern int  TCPDEBUG;
```

Turns on or off different levels of TCP debug tracing. In order to use this routine, the external variable *logfname* should be set to the name of the file into which all tracing will be written, and *TCPDEBUG* should be set to 1 before the call is made to *tcp_init* (). *TCPDEBUG* controls default tracing for all TCP connections. This default may be changed by using the *tcp_debug* () routine.

```
tcp_ioff (conn);      /* connection id */
```

Turns interrupts off; used while writing data to the display.

```
tcp_ion (conn);      /* connection id */
```

Turns network interrupts on.

```
tm_on ( );
```

Wakes up the timer task to process any events that went off while timer interrupts were disabled and to start a new alarm.

```
tm_off ( );
```

Turns off timer interrupts; useful when writing data to the display.

```
tk_block ( );
```

Blocks the current task and forces it to be rescheduled.

```
tk_cur
```

Points to the task control block of the currently running task; a global variable.

tk_yield ();

Yields the processor to any other task that can be run.

tk_wake (*tk*)

task *tk **/* task control block** ***/**

Wakes up a specified task. The *tk* parameter is a pointer to a control block, which is defined in **taskm.h**.

tk_setef (*tk, ef*)

task *tk **/* task control block** ***/**
task ushort ef **/* event flag** ***/**

Wakes up a task and sets a specified event flag for it. The *tk* parameter is a pointer to a control block, which is defined in **task.h**. The *ef* parameter specifies the event flag.

User-Defined Routines

Following are definitions of the six user-defined routines in the array that TCP calls. The names of these procedures are not fixed; that is, they can be renamed to meet your requirements.

User defined routines must be in the following order in the array definition:

```
us_procs        [0] = us_open
                 [1] = us_dispose
                 [2] = us_close
                 [3] = us_forclose
                 [4] = us_timeout
                 [5] = us_buff
```

us_open (*con, fh, fs*)

int con; **/* connection id** ***/**
long fh; **/* foreign host** ***/**
short fs; **/* foreign socket** ***/**

Called when the TCP connection reaches the **ESTABLISHED** state (that is, when both the local and foreign hosts can send data. Most applications should use this routine to wake up a task that writes data to the network.

```
int us_dispose (con, ptr, len, urg)
int  con;      /* connection id          */
char *ptr;     /* pointer to buffer to dispose */
int  len;     /* length of buffer          */
int  urg;     /* urgent pointer (see below) */
```

Called to send data to the user's program. This routine must return the size of the new local window, which will be made known to the foreign host

Warning: Do not return a window size of zero. If the local window size is set to zero, no more calls will occur.

A non-negative value for *urg* represents an offset in the buffer to the urgent data. If no urgent data is present, the value of *urg* is -1.

```
us_close (con, reason)
int  con;      /* connection id          */
int  reason;   /* 1 = normal close; 0 = abnormal close */
```

Called to indicate that both sides have closed and that the connection block is erased. The value of *reason* is **TRUE** if the connection closes normally, **FALSE** if the connection closes abnormally due to a **TCP RESET**.

```
us_forclose (con, reason)
int  con;      /* connection id          */
int  reason;   /* 1 = normal close; 0 = abnormal close */
```

Called if the foreign host sends **FIN** (that is, the foreign host closes its side of the connection) or **TCP RESET**. The **us_forclose** (foreign close) routine immediately closes the local side of the connection. The value of *reason* is **TRUE** if the connection closes normally, **FALSE** if the connection closes abnormally due to a **TCP RESET**.

```
us_timeout (con)
int  con;      /* connection id          */
```

Called when a timeout occurs.

```
us_buff (con, count)
int  con;      /* connection id          */
int  count;    /* amount of free space in TCP send buffer */
```

Called when the output buffer is filled and buffer space is not available. Usually a task which writes to the network should block itself when it determines that no

more space is available in the TCP send buffer. The main purpose of `us_buff` is to awaken the blocked task, which it always should do.

The Interface Program Tasking System

A **task** is a process together with the procedures that run in the process. The procedures are often from different protocol layers. **Tasking** is a way of organizing procedures and processes to form layers that:

- Preserve layer modularity
- Process the asynchronous data of layers efficiently.

One task cannot preempt another, but tasks can be preempted by routines (which are signal driven). A task stops running when it calls a blocking routine in the tasking module.

Tasks that are internal to the same protocol layer are said to share **states**. In task programming, one task can awaken another only if the two tasks share states. For example, if one task is to awaken another, it must have access to the task control block of the other task.

The tasking system is used by TCP and applications that use TCP. Each task has a stack and a task control block. A task control block is allocated by calling the `calloc()` structure. The pointer that `tk_fork()` returns is a pointer to this structure.

Figure 4-1 on page 4-12 compares the memory organization of a single AIX process running on the RT PC without tasking to the memory organization of single AIX process running with tasking. In the lower horizontal row (the one that represents memory organization with tasking), **TS** refers to a task stack and **TCB** refers to a task control block.

Process without Tasking →	3 F F F F F F F	Stack	2 F F F F F F F	Static Data	1 F F F F F F F	Program Code
Process with Tasking →	3 F F F F F F F	T T T S S S	2 F F F F F F F	T T T C C C B B B	1 F F F F F F F	Program Code

Figure 4-1. Relationship of Tasking to Memory Organization

These memory organizations are similar except that, for the process running with tasking, the stack is partitioned for multiple tasks and the static data area contains task control blocks. Each task stack is organized like the original program stack.

To use the tasking system, include the file `<taskm.h>`. `<taskm.h>` associates a task control block with every task; the task control block is a structure that provides a new task type. `<taskm.h>` also provides *events*, or binary semaphores, that allow communication between tasks and allow tasks to determine why they are started. The task control block is included in `<taskm.h>`.

A pointer to a task control block is analogous to a process ID. Task control blocks are chained together by the `tk_elt` member of the task control block structure to form a circular list. The tasking scheduler is a non-preemptive, round-robin scheduler that simply loops through the circular list of tasks until it finds one that can be run (that is, a task with a `tk_evf` that is set `TK_TRUE`) and then switches to that task. A combination of `setjmp` and `longjmp` routines performs the context switch. The global variable, `tk_cur` is a pointer to the task control block of the task that is currently running.

Generally, task wakeups should be treated as hints. When a task runs, it should try to determine why it was started and perform accordingly. Part of the function of a task should be to respond appropriately to nonspecific wakeups. Tasks are forked as runnable; therefore, the following structure is correct for a task:

```
task_definition(tk)
struct task      *tk;
{
/* all local declarations      */

        set_task(tk);      /* must be first executable statement */

        for (;;) {
                if (no_work) {
                        tk_block();
                        continue;
                }
                do_the_right_thing;
        }
}
```

udp

udp

Purpose

Provides the User Datagram Protocol (UDP) layer.

Libraries

`/usr/lib/libU.a`
`/usr/lib/libudp.a`

Syntax

```
#include <in_extern.h >  
#include <udp.h >
```

Description

The UDP provides a set of routines that allow commands to send messages (datagrams) to and receive messages from the other processes, possibly running on other hosts attached to other networks. It places no restriction on the format of data in the messages. The maximum length of a message is 512 bytes in the AIX implementation. The protocol is transaction oriented. Reliability and duplicate protection of datagrams are not guaranteed.

The routines in the `/usr/lib/libudp.a` and `/usr/lib/libU.a` libraries provide a user interface to the AIX network driver. (If you will perform packet tracing, link to `libudp.a`; otherwise, link to `libU.a`.) The `udp` library provides a basic set of routines to allocate, free, send, and receive UDP packets.

In order to use these routines, the user program must include a header file, containing type and structure definitions, and so on. To use the `udp` library, the `<udp.h >` and `<in_extern.h >` files must be included. The program must then be linked using the `-l udp` flag to include the `udp` library. In addition, the program must include the standard I/O library to use this library. This means that the `-IS` flag must also be used when the program is linked.

Routines

The `/lib/libudp` library contains the following routines:

```
udp_open(fhost, fsock, lsock)  
udp_name   fhost;  
ushort    fsock;  
ushort    lsock;
```

Opens a UDP connection to the specified foreign host with the specified local and foreign sockets. The foreign host address is a standard network address. The combination of foreign host, local socket, and foreign socket must be unique among all connections currently in progress on this host. The `udp_socket` routine may be used to obtain a unique local socket number. Note that any or all of the fields may be NULL to indicate **all**. It returns a file descriptor to be used in future read and write calls, or -1 for error. The external variable `errno` contains the error code.

```
ushort    udp_socket()
```

Returns a unique 16-bit value for the local UDP socket. It calls the operating system to obtain a unique value. The socket returned is > 1000 (decimal) to ensure that it is a unique socket.

```
udp_close(fd)  
int fd;
```

Closes the specified UDP connection. The parameter is an AIX file descriptor returned by the `udp_open` call.

```
caddr_t   udp_alloc(datalen, optlen)  
int       datalen;  
int       optlen;
```

Allocates the space for a UDP packet, with enough space for a data area of length **datalen** and enough space for **optlen** bytes of Internet options, plus room at the beginning for the local net, Internet, and UDP headers. It returns a pointer to the packet or NULL if unable to allocate. Also, it sets up the length fields of the Internet packet so that the `*in_head`, `udp_head`, and `udp_data` macros work properly (see “**internet**” on page 4-19). These macros return pointers to the Internet header, the UDP header, and the UDP data portions of the packet, respectively. They are described under “Macros and Type Definitions” on page 4-17.

udp

udp_free(*pkt*)
caddr_t *pkt*;

Frees the packet pointed to by **pkt**. The packet must have been allocated by **udp_alloc**. It is an error to attempt to free something not previously allocated by **udp_alloc**. Also, the user is responsible to remove any remaining references to the packet.

udp_read(*fd, buf, len*)
int *fd*;
caddr_t *buf*;
int *len*;

Attempts to read the next available UDP packet on the specified connection into the specified buffer of length *len*. It returns the length of the received packet in bytes. If no packet is available, it returns 0. Note that this routine is non-blocking. It also validates the UDP length and checksum of the received packet. If either are invalid, the packet is dropped and 0 is returned. The **udp_data** macro can be used to obtain a pointer to the start of the data area of the packet.

udp_bread(*fd, buf, len, timeout*)
int *fd*;
caddr_t *buf*;
int *len*;
int *timeout*;

Performs a blocking read for UDP packets, with a timeout on the read. It waits *timeout* seconds for a UDP packet to arrive on the specified UDP connection, then reads it into the specified buffer. If a packet is presently available, the routine reads it and returns immediately. It returns the length of the received packet in bytes, or 0 if the timeout expires before a valid UDP packet is received. It also returns 0 on network error, or if interrupted by a signal. In these cases **errno** contains the error code.

udp_time ()

Attempts to find the current time by requesting it from other UDP time servers. **udp_time** builds a time server request for each of the known time servers, then sends the request and waits for a response. If the request is answered, **udp_time** returns the time in seconds since midnight, 1-Jan-70 GMT. If the request is not answered (that is, no time server is running on the network), **udp_** returns 0.

```

udp_write(fd, buf, datalen)
udp_writex(fd, buf, datalen, lh )
int fd;
caddr_t buf;
int datalen;
int lh;

```

Writes the specified packet out to the specified net connection. The **buf** parameter is a pointer to a packet buffer as allocated by **udp_alloc**. The write routine assumes that the foreign address, Internet packet ID, protocol, and type of service fields (in the Internet header), and the local and foreign sockets (in the UDP header) have been supplied by the caller. If the Internet ID field is 0, the system assigns a unique ID. The system provides the rest of the UDP header, including checksum, then writes it to the net connection. The system returns the number of bytes written, if successful. If not successful, the system returns -1, and the external variable **errno** contains the error code. If the value of **lh** is zero, the optimal interface of those configured is selected.

```

long resolve_name (name)
register char *name;

```

Resolves a host name into an Internet address. Three name formats are accepted:

- A character string host name.
- An octal or decimal host number in the form *net, subnet, rsd, host*. Values for *net, subnet,* and *rsd* can be left blank or left out entirely; they default to the local net or subnet.
- A 32-bit hex number, preceded by a #, which is used without interpretation as the host number.

If a character string name is supplied, it is first looked up in a local host table. If it is not found there, the routine goes off to Internet name servers to try to resolve the name.

Macros and Type Definitions

In addition to the routines previously described, several data type definitions and macros are supplied in the **<udp.h>** header file to simplify writing network programs.

The following data types are defined:

```

ushort   An unsigned 16-bit integer, used many places in the UDP header.
caddr_t  An address in memory. On the RT PC, this is a character pointer. Packet
           pointers and other pointers to untyped data are of this type.

```

udp

The UDP level provides the following macros to obtain pointers to various useful portions of UDP packets and other useful data:

udp_head(*pip*)

Returns a pointer (**struct ip * pip**) to the UDP packet when given a pointer to the start of the ***internet** header.

udp_data(*pup*)

Returns a pointer (**struct udp * pup**) to the start of the data portion of a UDP packet, given a pointer to the UDP header. The pointer is a memory address, and can be cast to the appropriate type.

internet

Purpose

Provides the Internet protocol layer.

Libraries

`/usr/lib/libI.a`
`/usr/lib/libinternet.a`

Syntax

```
#include <ip.h >  
#include <in_extern.h >
```

Description

The Internet Protocol (IP) provides a set of routines for sending raw datagrams between hosts. It is used by a number of higher-level protocols, like User Datagram Protocol (UDP) and Transmission Control Protocol (TCP). Most users never need to access the Internet Protocol layer, but instead must be able to use it indirectly by using one of the higher-level protocols.

The routines in the `/usr/lib/libinternet.a` and `/usr/lib/libI.a` libraries provide a user interface to the AIX network driver. (If you perform packet tracing, link to `libinternet.a`; otherwise, link to `libI.a`.) The **internet** library contains routines to allocate, free, send, and receive the lower-level Internet Protocol packets, which are used by higher-level protocols. Routines in the **internet** library support packet fragmentation but not packet reassembly.

In order to use these routines, the user program must include a header file, containing type and structure definitions, and other necessary information. To use the **internet** library, the `<ip.h >` and `<in_extern.h >` files must be included. The program must then be linked with either the `-I internet` or `-II` flag to include the **internet** library, and with the `-IS` flag. In addition, using this library requires that the program include the standard I/O library. This means that the `-IS` flag must also be used when the program is linked. For additional information about Internet header fields, see “Tasks of Kernel and User Processes” on page 5-14.

Routines

The **internet** library provides the following routines:

```
in_open(prot, fhost, fsock, lsock)  
ushort    prot;  
in_name    fhost;  
ushort    fsock;  
ushort    lsock;
```

Opens an Internet connection on the specified protocol, to the specified foreign host, with the specified local and foreign sockets. Note that any or all of the parameters may be NULL to indicate that the field is to be ignored in packet demultiplexing. If this routine is called directly, rather than by **udp** or **tcp**, the value for foreign host, foreign socket and local socket should be 0. It also initializes a number of network parameters, like the local net header and trailer sizes for this connection. Note that this routine must be called before attempting to allocate a packet for this connection. It returns a file descriptor suitable for use in future read and write system calls on this connection, or -1 for error. In the event of an error, the external variable **errno** contains the error.

Note: The **in_open** call opens with NINTRUP mode enabled for the process issuing the call. This request can generate an EACCES error that indicates NINTRUP enabled by another process.

For related information, see “Tasks of Kernel and User Processes” on page 5-14.

```
in_logpkt (ppkt, len, dir)  
unsigned char *ppacket  
int len  
int dir
```

Logs specified packets to standard output. *ppkt* is a pointer to the packet. *len* specifies the length of the data portion of the packet in bytes. *dir* specifies the direction of the packet; its value can be either **INPKT** (for an input packet) or **OUTPKT** (for an output packet).

```
in_close(fd)  
int fd;
```

Closes the network connection specified by **fd**. The connection should have been opened by the **in_open** call.

```
caddr_t    in_alloc(datalen, optlen)  
int       datalen;  
int       optlen;
```

Allocates the storage for an Internet packet with data area of size *datalen*, and including space for an Internet option string of length *optlen*, including space for the local net header and trailer and for the Internet header. The length will be increased to an even number of bytes if needed. It also sets up the Internet header length fields so that the various macros like **in_data** (see below) will work properly. It returns a pointer to the allocated packet, or NULL if unable to allocate one.

```
in_free(pkt)  
caddr_t    pkt;
```

Frees the packet pointed to by **pkt**. The packet must have been allocated by **in_alloc**. It is an error to attempt to free something not allocated by **in_alloc**.

```
in_read(fd, buf, len)  
int fd;  
caddr_t  buf;  
int     len;
```

Reads the next available Internet packet for the specified connection from the net into the specified buffer of length *len*. It returns the length of the received packet in bytes, or 0 if no packet is available. Note that this routine is non-blocking. The **in_data** macro may be called to obtain a pointer to the start of the data area of the packet.

```
in_bread(fd, buf, len, timeout)  
int fd;  
caddr_t  buf;  
int     len;  
int     timeout;
```

Performs a blocking read for an Internet packet, with a timeout on the read. It waits up to *timeout* seconds for an Internet packet to arrive on the specified Internet connection, then reads it into the specified buffer. If a packet is presently available, the routine then reads the packet and returns immediately. It returns the length of the received packet in bytes, or 0 if the timeout expires before a valid packet arrives. It also returns 0 if a net error occurs or if the call is interrupted by the arrival of a signal. In these cases, **errno** contains the error code.

internet

```
in_write(fd, buf, datalen)  
in_writex(fd, buf, datalen, lh)  
int fd;  
caddr_t buf;  
int datalen;  
int lh;
```

Writes the specified packet to the net. The **buf** parameter is a pointer to the start of a packet buffer, such as one allocated by **in_alloc**. The **datalen** parameter is the length of the data portion of the packet in bytes. The packet buffer is assumed to have space at the beginning for local net and Internet headers. It is also assumed that there is enough space in the packet buffer to pad the packet up to an even number of bytes, because all transmitted packets must be an even number of bytes in length. (If the packet was obtained by **in_alloc** or by **in_read**, this is the case.) The caller is assumed to have provided the foreign host address, Internet packet ID, protocol, and the type of service fields in the Internet header, as well as any option fields. If the Internet ID field is 0, the system assigns a unique ID. The Internet header length field is assumed to have been provided by the allocator of the packet (for example, in **in_alloc**). This routine provides the remainder of the Internet header before writing the packet out to the network. It returns the number of bytes written if successful. If not, it returns -1 and the external variable **errno** contains the system error code. If the value of **lh** is zero, the optimal interface of those configured is selected.

Macros and Type Definitions

In addition to the routines previously described, several macros and data type definitions are supplied in the **<ip.h>** header file to simplify writing network programs.

The following data types are defined:

- ushort** An unsigned 16-bit integer, used in many places in the Internet header.
- caddr_t** An address in memory. On the RT PC, this a character pointer. Packet pointers and other pointers to untyped data are of this type.
- in_name** An Internet address. This is a 4-byte value appearing in the Internet header. Routines are available to deal with Internet addresses.

Several macros are provided at the Internet level to aid in obtaining pointers to various useful portions of Internet packets and other useful data. The following macros are presently available:

in_head(*ppkt*)

Returns a pointer (**caddr_t p pkt**) to the start of the Internet header of the specified packet. The packet is assumed to have been allocated by one of the packet allocation routines described in this manual, such as **in_alloc**, in order to ensure that the Internet header length is set up correctly.

in_data(*pip*)

Returns a pointer (**struct ip * pip**) to the data portion of an Internet packet given a pointer to the Internet header. The pointer is a memory address and may be converted into the appropriate data type.

in_options(*pip*)

Returns a pointer (**struct ip * pip**) to the start of the Internet option string in an Internet packet given a pointer to the Internet header. Although the **internet** library on AIX does not process Internet options, this macro and the **in_optlen** macro (and the **optlen** parameters to **udp_alloc** and **in_alloc**) permit user programs to deal with Internet options if they wish. The pointer returned is a character pointer to permit the user program to parse the option string.

in_optlen(*pip*)

Returns the length of the Internet option string in an Internet packet in bytes given a pointer to the Internet header (**struct ip * pip**). Note that the string is always a multiple of 4 bytes in length, but can include padding (NULL bytes).

libS

libS

Purpose

Provides miscellaneous routines for use with the TCP, UDP, and IP layers.

Library

`/usr/lib/libS.a`

Description

The `/usr/lib/libS` is a collection of routines that are used with the TCP, UDP, and IP layers but are not included in the other Interface Program libraries. The two routines in `libS` that may be of interest to the application programmer are **cksum** and **netopen**:

cksum (*buf, lenhw, len*)

```
char * buf;          /*address of buffer to checksum*/
int lenhw;           /*length of buffer in half words*/
int intval;         /*initial checksum value, normally 0*/
```

The checksum is a ones complement of the buffer. Data to be check summed is on a half- or full-word boundary. This routine returns the checksum value.

netopen (*devname, netstruct*)

```
char * devname      /*device name, "/dev/net0"*/
struct netdf netstruct
#include netioctl.h
```

Opens the `/dev/net0` device. To close the `/dev/net0` device, use the system call `close ()`. This routine returns a file descriptor.

Chapter 5. /dev/net0 Device Driver

CONTENTS

About This Chapter	5-2
/dev/net0	5-3
/dev/net0 Calls	5-4
/dev/net0 IOCTLs	5-5
Tasks of Kernel and User Processes	5-14

About This Chapter

This chapter describes the Application Programming Interface (API) to the Interface Program `/dev/net0` device driver.

Note: Generally, you do not use the `/dev/net0` device driver directly except to issue IOCTLs. Rather, you should use the libraries described in Chapter 4, “Protocol Library Routines.”

/dev/net0

Purpose

Provides kernel services for the Interface Program.

Synopsis

```
#include <netioctl.h>
```

Description

The **/dev/net0** device driver is the kernel-level support for Interface Program. It is a multiplexed device driver that supports up to 32 simultaneous open calls, providing one connection per call. As physical media, **/dev/net0** supports both the IBM RT PC Baseband Adapter for use with Ethernet and the IBM Token-Ring Network RT PC Adapter. The principal functions of **/dev/net0** are to:

- Reading Internet packets from the network.
- Writing Internet packets to the network.
- Performing address translation (using ARP) between Internet and its physical media addresses.
- Providing gateway support between Baseband Adapter networks, Token-Ring networks, or both, and bridge support between Token-Ring networks.

The interface to the **/dev/net0** device driver is through the Internet library calls.

The **/dev/net0** device driver supports the following types of incoming packets:

- IP
- ARP or broadcast
- VAX

The **/dev/net0** device driver can process VAX trailer encapsulation protocol (VAX trailers) on incoming packets. **/dev/net0** supports three types of VAX trailers: 0x1001, 0x1002, and 0x1003. If VAX trailers are to be transmitted, all hosts sharing the network environment must accept them uniformly. While the Interface Program accepts and processes VAX trailers, it does not transmit them.

net

The following IOCTLs must be issued by root; otherwise, `/dev/net0` returns an error:

IOCCONFIG
NIOCGATE
NIOCGDEL
NIOCHGADD
NIOCHGCLR
NIOCSHOST

Following is a list of the IOCTLs that `/dev/net0` supports.

`/dev/net0` Calls

open

`/dev/net0` is a multiplexed device driver that can have a maximum of 32 concurrent opens. The **open** call initializes the device driver structures, initializes ARP, opens the block I/O device manager, and starts devices for ARP and IP. To complete the open, the user must issue the **NIOCNODS** IOCTL. The **open** call returns file descriptors.

Note: The **netopen** subroutine in **libS.a** opens the device driver and issues the IOCTL.

The structure of the **open** call is:

```
int open (path,oflag[],mode)
char *path;                               /* path = /dev/net0 */
int oflag,mode;
```

close

Closes `/dev/net0` when it has been opened with either the **open** system call or the **libS.a** subroutine **netopen**. If this is the last close of the device driver, **close** issues the **halt device** for IP and ARP and closes the block I/O manager. **close** also cleans up the structures of the device driver, freeing any packets that were pending for it. The structure of the **close** call is:

```
int close (fildes)
int fildes;                               /* file descriptor returned by open */
```

read

Reads a packet from the network. The read may be blocking or non-blocking. Blocking status is controlled by the *timeout* field in the structures of the device driver. If the *timeout* value is not 0, then a blocking read is performed until timeout; otherwise, a non-blocking read is performed. The **NIOCSTMO** IOCTL controls the *timeout* value. When **read** completes, it clears the *timeout* value. If incoming data is fragmented, it is reassembled; the user receives a complete

packet. The structure of the **read** call is:

```
int read (fildes, buf, nbyte)
int fildes;          /* file descriptor returned by open */
char *buf;          /* buffer to place data in */
unsigned int nbyte; /* length of buffer */
```

write

Writes a packet to the network. **write** fills in the ID, checksum, and source address fields of the IP header, and also fills in the local header. The structure of the **write** call is:

```
int write (fildes, buf, nbyte)
int fildes;          /* file descriptor returned by open */
char *buf;          /* packet to send */
unsigned int nbyte; /* size of buffer, including local */
                  /* header and IP header */
```

/dev/net0 IOCTLS

IOCCONFIG

Defines the association between the **/dev/net0** device driver and multiple VRM device drivers (for the Baseband Adapter and the Token-Ring Adapter). *arg* is a pointer to an **IOCCONFIG** structure defined in **<sys/iocfg.h>**. The option field pointer in the **<sys/iocfg.h>** structure points to information specific for **/dev/net0** for each interface to be defined. The structure is:

```
# define IFNAMSIZ 16
struct ifreq {
    char    ifr_name [IFNAMSIZ]; /* name of interface */
    unsigned long   ifr_mymach; /* local IP address */
    unsigned long   ifr_mask; /* subnet mask */
    int    ifr_mtu; /* maximum transmission unit */
    char    ifr_type; /* type of interface */
```

To issue **IOCCONFIG**:

```
int ioctl (fildes, IOCCONFIG, arg);
int fildes;
int IOCCONFIG;
struct iocfg *arg;
```

IOCINFO

Returns the following structure, defined in `<sys/devinfo.h>`:

```
struct devinfo {
    char devtype;           /* DD-BIO */
    char flags;            /* 0 */
    union {
        struct {
            struct {
                char type;      /* hardware type: ethernet or
                                token ring */
                char haddr [ ]; /* hardware ethernet/token
                                ring address */
                long mymach;    /* local IP address */
                long subnet_mask; /* subnet mask */
                int mtu;        /* maximum transmission unit */
                char if_flags;  /* up/down: 1=ATTACHED
                                2=RUNNING
                                3=PRIMARY INTERFACE */
                char if_name [IFNAMSIZ]; /* interface name */
            } lan[8];
        } bio;

        .
        .
        .
                                /* for other devices */
    } un;
};
```

The structure from `char if_name` through `char if_flags` is repeated an indefinite number of times.

To issue **IOCINFO**:

```
int ioctl (fildev, IOCINFO, 0);
int fildev;
int IOCINFO
```

IOCTYPE

Returns the device type **Block I/O, DD-BIO**, defined

in `<sys/devinfo.h>`. The parameter is ignored. To issue **IOCTYPE**:

```
int ioctl (fildev, IOCTYPE, 0);
int fildev;
int IOCTYPE;
```

NIOCGATE

Defines the default gateway. If the defined gateway replaces an existing one, **NIOCGATE** updates connections as necessary for use of the new gateway. The *arg* parameter points to an integer that contains the local net address of the gateway to which packets (without a destination host in the host gate table) should be sent. To issue **NIOCGATE**:

```
int ioctl (fildev, NIOCGATE, arg);
int fildev;
int NIOCGATE
char *arg;
```

NIOCGDEL

Deletes a gateway. *arg* points to an integer containing the address of the gateway to be deleted. All entries in the host gate table for that gateway are deleted. The default gateway is set to NULL if it is the same as the gateway being deleted. To issue **NIOCGDEL**:

```
int ioctl (fildev, NIOCGDEL, arg);
int fildev;
int NIOCGDEL;
char *arg;
```

NIOCGETHOST

Gets the Internet address of this host. *arg* is a pointer to a four-byte array. Upon return, *arg* contains the Internet address of the host. To issue **NIOCGETHOST**:

```
int ioctl (fildev, NIOCGETHOST, arg);
int fildev;
int NIOCGETHOST;
char *arg;
```

NIOCGETHHT

Gets the local net header and trailer sizes. *htsize* is a two-word array. Upon return from this request, *htsize [0]* contains the number of bytes to allocate for

the packet local net header and *htsize [1]* contains the number of bytes to allocate for the local net trailer. To issue **NIOCGETHHT**:

```
int ioctl (fildes, NIOCGETHHT, htsize);
int fildes;
int NIOCGETHHT;
int htsize [2];
```

NIOCGETL

Gets the net mode for the net connection. The possible modes are:

NINTRUP 01

Indicates that interrupts are enabled and that no packets have been missed. With interrupts enabled, the interrupt SIGAIO is sent to the process when a packet for this connection is read into the net handler read buffer. Only one process per connection can enable this mode.

NINTMISS 02

Indicates that interrupts are not enabled and that packets have been missed (that is, a packet arrived when no process had enabled SIGAIO).

00

Indicates that interrupts are not enabled and that no packets have been missed.

03

Indicates that interrupts are enabled and that packets have been missed.

To issue **NIOCGETL**:

```
int ioctl (fildes, NIOCGETL, arg);
int fildes;
int NIOCGETL;
char *arg;                               /* mode returned */
```

NIOCGETMAXPKT

Gets the maximum Internet packet size (in bytes) that may be received. Upon return, the integer pointed to by *arg* contains that value. To issue **NIOCGETMAXPKT**:

```
int ioctl (fildes, NIOGETMAXPKT, arg);
int fildes;
int NIOCGETMAXPKT
char *arg;
```

NIOCGIF

Returns the address of the interface used to send data to a particular foreign host. *temp* is a two-word array. When the address of a foreign host is supplied for *temp[0]*, **NIOCGIF** returns the address of the interface for sending to the host in *temp[1]*. To issue **NIOCGIF**:

```
ioctl (fd, NIOCGIF, temp);
unsigned long  temp[2]
```

NIOCGSKT

Gets a unique socket ID. *arg* is a pointer to an integer that indicates where the returned 16-bit unique socket ID (or port number) is placed. To issue **NIOCGSKT**:

```
int ioctl (fildev, NIOCGSKT, arg);
int fildev;
int NIOCGSKT;
char *arg;
```

NIOCHGADD

Adds a host to the host gateway table. For input, *gt_host* should contain an Internet host address and *gate* should contain the local net address of the first hop gateway to which packets destined for the host should be sent. If the host name is already in the table, this option replaces its current entry. If the local address is 0, the address for the entire network is used. If more than 32 entries are specified, an attempt is made to drop a host that currently has no net connection. To issue **NIOCHGADD**:

```
int ioctl (fildev, NIOCHGADD, arg);
int fildev;
int NIOCHGADD;
struct gt_host *arg;

struct gt_host {
    long gt_fhost;
    long gt_gate;
}
```

NIOCHGCLR

Clears the host gateway table. The *arg* is NULL. To issue **NIOCHGCLR**:

```
int ioctl (fildev, NIOCHGCLR, 0);
int fildev;
int NIOCHGCLR
```

NIOCNODS

Issued after the open system call to **/dev/net0** to define the connection at the time it is opened. *arg* is a pointer to the following **netdf** structure:

```
struct netdf {
    ushort nd_prot1;           /*net protocol id=1*/
    ushort nd_prot2;           /*Internet protocol id*/
    in_name nd_fhost;         /*foreign host*/
    ushort nd_fsock;          /*foreign socket*/
    ushort nd_lsock;          /*local socket*/
}
```

To issue **NIOCNODS**:

```
int ioctl (fildes, NIOCNODS, arg);
int fildes;
int NIOCNODS;
struct netdf *arg;
```

NIOCQUERY

Gets VRM device driver statistics for Baseband Adapter and Token-Ring devices. The *arg* structures are:

```
union {
    struct {
        int dummy;           /*statistics for Ethernet*/
        int intr_rcv;        /*iodn of VRM device */
        int pkt_accept;     /*Receive interrupts */
        int bytes_rcv;      /*Packets accepted */
        int pkt_reject;     /*Receive byte count */
        int ring_queue_full; /*Packets rejected */
        int rcv_pkt_ovflow; /*Ring queue full count */
        int slih_ring_empty; /*Receive pkts overflow */
        int pkt_xmt;        /*SLIH ring empty count */
        int bytes_xmt;      /*Transmit interrupts */
        int queues_saved;  /*Bytes transmitted */
        int wr_queue_proc; /*Number of queue saved */
        unsigned short collision; /*Write queue processed */
        unsigned short collision16; /*Collision counter */
        unsigned short shorted; /*Collision 16 counter */
        unsigned short underflow; /*Shorted counter */
        unsigned short short_pkt; /*Underflow counter */
        unsigned short align_error; /*Short pkt counter */
    };
};
```

```

    unsigned short  crc_error;      /*CRC error counter    */
    unsigned short  overflow;      /*Overflow counter     */
} et_stat;

struct {
    int             dummy;         /*statistics for Token */
    int             intr_rcv;      /*iodn of VRM device   */
    int             pkt_reject;    /*Receive interrupts   */
    int             pkt_accept;    /*Packets rejected     */
    int             bytes_rcv;     /*Packets accepted     */
    int             bytes_rcv;     /*Receive byte count   */
    int             ring_queue_full; /*Ring queue full count */
    int             slih_ring_empty; /*SLIH ring empty count */
    int             xmt_intr;      /*Transmit interrupts   */
    int             xmt_completes; /*Transmit completes   */
    int             bytes_xmt;     /*Bytes transmitted    */
    int             snd_qcnt;      /*Send Queues Counter  */
    int             sio_qcnt;      /*SIO Queues Counter   */
    int             pkt_xmt;       /*Packets Transmitted   */
    int             queues_saved;  /*Number of queue saved */
} tk_stat;

```

```

} query_vrm;

```

To issue **NIOCGQUERY**:

```

int ioctl (fildev, NIOCQUERY, arg);
int fildev;
int NIOCQUERY;
struct query_vrm *arg;

```

NIOCSETL

Sets the net mode for the net connection:

NINTRUP 01

Indicates that interrupts are enabled and that no packets have been missed. With interrupts enabled, the interrupt SIGAIO is sent to the process when a packet for this connection is read into the net handler read buffer. Only one process per connection can enable this mode.

00

Indicates that interrupts are not enabled and that no packets have been missed.

To issue **NIOCSETL**:

```
int ioctl (fildev, NIOCSETL, arg);
int fildev;
int NIOCSETL;
char *arg;                               /* mode to set*/
```

NIOCShost

Sets a local IP address. *arg* is a pointer to an unsigned integer that contains the local IP address. The IP address of the host is marked as the primary interface.

To issue **NIOCShost**:

```
int ioctl (fildev, NIOCShost, arg);
int fildev;
int NIOCShost;
char *arg;
```

NIOCSNode

Sets the node name (host name) in the **uname** structure. *arg* is a pointer to a NULL terminated string containing the node name. To issue **NIOCSNode**:

```
int ioctl (fildev, NIOCSNode, arg);
int fildev;
int NIOCSNode;
char *arg;
```

NIOCSTMO

Sets the timeout value, in seconds, to be used by **udp_bread** and **in_bread** in setting the timeout value for return from a read. *arg* is a pointer to an integer containing the timeout value. To issue **NIOCSTMO**:

```
int ioctl (fildev, NIOCSTMO, arg);
int fildev;
int NIOCSTMO;
char *arg;
```

NIONREAD

Gets a read count. Upon return, *arg* is a pointer to an integer that contains the number of bytes in the first packet in the **/dev/net0** read buffer for this connection. To issue **NIONREAD**:

```
int ioctl (fildev, NIONREAD, arg);
int fildev;
int NIONREAD;
char *arg;
```

NIONWRITE

Gets write count. The *arg* structure is:

```
struct nwrite {
    int    in_name;
    int    maxfrag;
}
```

Upon return, *maxfrag* contains the maximum fragment size that may be sent between this host and the foreign host specified in **in_name**. To issue **NIONWRITE**:

```
int ioctl (fildev, NIONWRITE, arg);
int fildev;
int NIONWRITE;
struct nwrite *arg;
```

Tasks of Kernel and User Processes

The following table shows the Internet header fields used or modified by the kernel and user processes.

Internet Header Field	Output Filled By	Kernel Checks or Uses Output?	Input Checked or Used By
Header length	user	yes	kernel/user
Version	user		
Type of service	user		
Packet length	user	yes	kernel/user
ID	kernel/user		kernel
Fragment offset	user		kernel
Flags	user		kernel
Time to live	user		
Protocol	user		kernel
Checksum	kernel		kernel
Source	kernel		kernel/user
Destination	user	yes	kernel
Options	user		user

Figure 5-1. Tasks Performed, Internet Header

Notes:

1. If the Internet ID field is 0 in an outgoing packet, the kernel fills it. If it is not 0, the kernel does not fill it.
2. The user process does the fragmenting.

The following table shows the local header fields used or modified by the kernel processes:

Local Header Field	Output Filled By	Kernel Checks or Uses Output?	Input Checked or Used By
Dummy	not used		
Source	kernel		
Destination	kernel	yes	kernel
Type	kernel		kernel

Figure 5-2. Tasks Performed, Local Header

The following table shows the TCP header fields used or modified by user processes.

TCP Header Field	Output Filled By	Kernel Checks or Uses Output?	Input Checked or Used By
Source port	user	yes	user
Destination port	user	yes	user
Sequence number	user		user
Acknowledgement number	user		user
Offset to data	user		user
Flags	user		user
Window Size	user		user
Checksum	user		user
Urgent pointer	user		user

Figure 5-3. Tasks Performed, TCP Header

The following table shows the UDP header fields used or modified by user processes.

UDP Header Field	Output Filled By	Kernel Checks or Uses Output?	Input Checked or Used By
Source port	user		user
Destination port	user	yes	
Length	user		user
Checksum	user		user

Figure 5-4. Tasks Performed, UDP Header

Appendix A. Customizing the Program

About This Appendix

Before you can use the Interface Program, you must install the necessary hardware and software, and then customize the software to meet your requirements. For information about the software and hardware prerequisites, see “Before You Begin” on page iv. Once you have installed the Interface Program, return to this section for information about establishing a network.

How to Customize an Interface Program Network

Following are the general steps in customizing an Interface Program network. You must be able to supply the specific information pertaining to your system and network.

1. Set the jumpers on the IBM RT PC Baseband Adapter for use with Ethernet, the IBM Token-Ring Network RT PC Adapter, or both, according to the instructions in *Options Installation*.
 - If more than one Baseband Adapter is installed, each one must be set at a different interrupt level and address space. Interrupt levels cannot be shared among Baseband Adapter cards or other devices.
 - Multiple Token-Ring Adapters can share the same interrupt level.
2. Use the **devices** command to add **adapters**. You can add Baseband Adapters and Token-Ring Adapters to run the Interface Program. The keyword values must match the adapter switch settings.
3. Use the **devices** command to add PTYs if your system is to be available to other users for remote login (as in Telnet sessions.). Select the appropriate terminal type for each PTY (just as you would for a TTY). If that PTY is to be used for Telnet (remote login to this host), set the values of **logger** and **automatic enable** to **true**. (**automatic enable** causes the **penable** command to be run for that device at system start, making it available for use by Telnet.) For more information about customizing PTYs for Telnet, see “**tn**” on page 2-43. For information about customizing PTYs for **rexec**, see “**rexecd**” on page 3-15.
4. Customize the **/etc/hosts** file according to the information under “**hosts**” on page 1-25.
5. Customize the **/etc/rc.tcpip** file according to the information under “**rc.tcpip**” on page 1-33.
6. Customize the **/etc/hosts.equiv** file according to the information under “**hosts.equiv**” on page 1-28.
7. Customize the **/etc/net** file according to the information under “**net**” on page 1-29.
8. Customize the **/etc/networks** file according to the information under “**networks**” on page 1-31.

Note: The name of your host must be used as the parameter to **/bin/hostname** in **/etc/rc.tcpip**, and it must correspond with the appropriate address in the **/etc/net** stanza.
9. Customize the **/etc/gateways** file according to the information under “**gateways**” on page 1-23.

-
10. Customize the `/etc/3270.keys` file if necessary. For related information, see “.3270keys” on page 1-36.
 11. Customize the `$(HOME)/.3270keys` file according to the information under “.3270keys” on page 1-36.
 12. Customize the `$(HOME)/.netrc` file according to the information under “.netrc” on page 1-34.
 13. In `/etc/rc`, remove the comment symbols from the lines that start the Interface Program daemon processes. (This causes the network to be configured and the daemons started when the system is started.)
 14. Run `shutdown` and then restart the system.

Appendix B. Samples

About This Appendix

This appendix contains samples of how to use the routines in the **tcp** and **tcpm** libraries, and the tasking system. “Running the test Programs” on page B-2 explains how to run the **tcptestm.c** and **tcptest.c** programs. “tcptestm.c” on page B-3 is a listing of the **tcptestm** program for multiple TCP connections (**libtcpm.a**). “tcptest.c” on page B-12 is a listing of the **tcptest** program for a single TCP connection (**libtcp.a**).

To understand these samples, you must be familiar with the C programming language.

Running the test Programs

“Running `tcptestm`” shows how to run `tcptestm` (using the library that supports multiple TCP connections `libtcpm.a`). “Running `tcptest`” explains how to run `tcptest` (using the library that supports a single TCP connection, `libtcp.a`). Both programs support the same commands, which are listed in “`tcptestm` and `tcptest` commands” on page B-3.

Note: `tcptestm` and `tcptest` should be run on two hosts.

To compile the programs and place them in the `/usr/bin` directory, enter the following command:

```
make install -f make.tcptest
```

Running `tcptestm`

1. Start the program on the first host with the following command:

```
tcptestm -l 1001
```

The `-l` flag sets this host to listen; the `1001` parameter specifies a port.

2. Start the program on the second host with the following command:

```
tcptestm hostname 1001
```

The *hostname* parameter is the name of the host that is listening.

Running `tcptest`

1. Start the program on the first host with the following command:

```
tcptest -l 1001
```

The `-l` flag sets this host to listen; the `1001` parameter specifies a port.

2. Start the program on the second host with the following command:

```
tcptest hostname 1001
```

The *hostname* parameter is the name of the host that is listening.

tcptestm and tcptest commands

After the connection is open, enter commands by:

1. Pressing the **ESC** key
2. Entering the command name.

The commands are:

- c** Close connection.
- e** Echo local.
- r** Echo remote.
- q** Quit.
- v** Quote next character. This command allows you to send a character as an octal number (for example, **<ESC>v007** produces a beep on the foreign host).
- d** Toggle debug state.

tcptestm.c

```
#
/* tcptestm.c */

/* Copyright 1984 by the Massachusetts Institute of Technology */
/* See permission and disclaimer notice in the file "notice.h" */
#include <notice.h>

/* EMACS_MODES: c !fill */

typedef unsigned char uchar_t;
#include <stdio.h>
#include <signal.h>
#include <sys/ioctl.h>
#include <termio.h>
#include <errno.h>
#include <ip.h>
#include <netioctl.h>
#include <taskm.h>
```

```

#define ESCHAR 033

#define WINDOW 1024 * 2

extern int    displa ();
extern int    us_opna (), us_opnl (), us_cls (),
us_tmo (), us_space();
extern int    done ();
extern int    his_cls();

extern char   *ip2ascii ();

event  open_done = 0;      /* set to 1 when open completed */
event  close_done = 0;    /* set to 1 when close completed */

task   *main_tk;          /* main task */

struct termio  stermio;

int     con;

main (argc, argv)
int     argc;
char    **argv;
{
    long                fhost, lhost;
    register unshort    fsock, lsock;
    register char       c;
    register int        i;
    register int        lstn = FALSE;
    register int        dbg = FALSE;
    struct termio       termio;
    char                forname [17], locname [17]
    register int        escseen = FALSE;
    int                 (*pblock[6])();

```

```

extern char      *logfname;
extern int      TCPDEBUG;

if (argc < 3) {
    printf ("usage: %s [-l] [-d] host port\n", argv[0]);
    exit (1);
}

for (i = 1; i < argc && argv[i][0] == '-'; i++)
    switch (argv[i][1]) {
        case 'l':      /* listen */
            lstn = TRUE;
            break;
        case 'd':      /* debug */
            dbg = TRUE;
            logfname = "tcptest.trace";
            TCPDEBUG = 0x3f;
            break;
        default:
            printf ("usage: %s [-l] [-d] host port\n", argv[0]);
            exit (1);
    }

setbuf(stdout, 0);

ioctl(0,TCGETA,&termio);
stermio = termio;

if (!lstn) {          /* active open */
    termio.c_lflag = 070;
    termio.c_cc[4] = 1;
    termio.c_cc[5] = 1;
    termio.c_oflag = ONLCR | OPOST;
    termio.c_cflag &= ~PARENB;
    termio.c_cflag |= CS8;
    termio.c_iflag = 0 | ICRNL;
    ioctl (0, TCSETAF, &termio);
}

```

```

signal (SIGINT, done);
signal (SIGTERM, done);

tcp_init(6000);

main_tk = tk_cur;      /* save current task */

if (lstn)
    pblock[0] = us_opn1;
else
    pblock[0] = us_opna;

pblock[1] = displa;
pblock[2] = us_cls;
pblock[3] = his_cls;
pblock[4] = us_tmo;
pblock[5] = us_space;
if (lstn) {
    lsock = atoi(argv[i]);
    if ((con = tcp_listen (lsock, WINDOW, pblock)) == -1) {
        printf("tcp_lstn failed: lsock %d\n", lsock);
        exit(1);
    }
}
else {

    if ((fhost = resolve_name (argv[i])) == 0L) {
        printf ("%s: unknown host\n", argv[i]);
        exit (1);
    }

    fsock = atoi(argv[i+1]);

    printf ("Trying...");

    if ((con = tcp_open (fhost, 0L, fsock, (short) 0, WINDOW,
pblock)) == -1) {

```

```

        printf("tcp_open failed: fhost = %x, fsock = %d\n",
               fhost, fsock);
        exit(1);
    }
}

for (;;) {
    if (lstn) {
        tk_block ( );
        continue;
    }
    c = getkey ( );
    if (escseen) {
        esceen = FALSE;
        switch (c) {
case 'c':
            /* close connection */
            tcp_close (con);
            while (!close_done)
                tk_block;
            done ( );
            break;
case 'e':
            /* echo local */
            ioctl (0,TCGETA,&termio);
            termio.c_lflag != ECHO;
            ioctl (0,TCSETA,&termio);
            break;
case 'r':
            /* echo remote */
            ioctl(0,TCGETA,&termio);
            termio.c_lflag &= ~ECHO;
            ioctl(0,TCSETA,&termio);
            break;
case 'a':
            /* abort          */
            tcp_abort(con);
            while (!close_done)
                tk_block ( );
            done ( );
            break;
case 'i':
            /* get connection info */

```

```

                                conn_info (con, &lhost, &fhost,
                                                &lsock, &fsock);
                                strcpy (forname, ip2ascii (fhost));
                                strcpy (locname, ip2ascii (lhost));
                                printf ("Connection information:
                                        local host %s, foreign host %s,
                                        local port %d, remote port
                                        %d\n",
                                        locname, forname, lsock, fsock);
                                break;

case 'q':
                                /* quit */
                                tcp_close (con);
                                done ( );
                                break;

case 'v':
                                /* quote next character */
                                c = 0;
                                for (i = 0; i < 3; i++)
                                        c = (c << 3) + (getkey ( ) - '0');
                                tcp_putc (con, c);
                                tcp_flush(con);
                                break;

case 'd':
                                /* toggle debug state */
                                dbg = !dbg;
                                break;

default:
                                tputc (con, c);
                                tcp_flush(con);
                                break:
                                }
                                continue:
                                }

else
if (c == ESCHAR) {
        escseen = TRUE;
        continue;
}

```

```

        }
        tcp_putc (con, c);
        tcp_flush(con);
    }
}

displa (con, buf, len, urg)
register int    con;
register char   *buf;
register int    len;
register int    urg;
{

    register char *bp;

    char *endp = &buf[len];
    for (bp=buf;bp < endp;bp++)
        if (*bp & 0200) {
            write (1, buf, bp-buf-1);
            buf = bp + 1;
            printf("\\\\%o",(*bp & 0377));
        }
    write (1, buf, bp-buf);

    return(WINDOW);
}

us_opna (con, fhost, fsock)
register int    con;
register long   fhost;
register short  fsock;
{
    printf ("\nActive connection opened foreign host %s, fsock %d.\n",
            fhost, fsock);
}

```

```

us_opn1 (con, fhost, fsock)
register int    con;
register long   fhost;
register short  fsock;
{
    printf ("\nServer connection opened foreign host %s, foreign port %d.\n",
            ip2ascii (fhost), (fsock);
    tk_setef (main_tk, &open_done); /*wake up main task */
}
us_opn1 (con, fhost, fsock)
register int    con;
register long   fhost;
register short  fsock;
{
    printf ("\nServer connection opened foreign host %s, foreign port %d. \n",
            ip2ascii (fhost), fsock);
    tk_setef (main_tk, &open_done); /*wake up main task */
}

us_space (con, count)
register int    con;
register int    count;
{
}

us_cls (con, reason)
register int    con;
int            reason;
{
    printf ("\nClosed %d\n", reason);
    tk_setef (main_tk, &close_done);
}

his_cls(con, reason)
int    con;

```

```

int    reason;
{
    printf("\nForeign closed %d\n", reason);
    if (!reason) {                /* received reset        */
        done( );
    }
    tcp_close(con);
}

us_tmo (con)
int    con;
{
    printf ("Host not responding\n");
    done ( );
}

done ( )
{
    ioctl(0,TCSETAWk,&sternio);
    exit (0);
}

getkey ( )
{
    char    c;
    int    nch;

    for (;;) {
        ioctl(0, TIONREAD, &nch);
        if (nch == 0) {
            tk_yield ( );
            continue;
        }
        if (read (0, &c, sizeof(c)) == sizeof(c))
            return (c & 0377);
    }
}
/*

```

```

* Convert IP address into char string xxx.xxx.xxx.xxx
*/
char *ip2ascii (ipaddr)
register long  ipaddr;
{
    union {
        long l;
        char c[4];
        { foo;
    static char    tmp[17];

    foo.l = ipaddr;
    sprintf (tmp, "%d.%d.%d.%d", foo.c[0] & 0xff, foo.c[1] & 0xff,
            foo.c[2] & 0xff, foo.c [3] & 0xff);

    return (tmp);
}

```

tcptest.c

```

/* tcptest.c */

/* Copyright 1984 by the Massachusetts Institute of Technology */
/* See permission and disclaimer notice in file "notice.h" */
#include    <notice.h>

/* EMACS_MODES: c !fill */

typedef unsigned char uchar_t;
#include    <stdio.h>
#include    <signal.h>
#include    <sys/ioctl.h>
#include    <termio.h>
#include    <sys/errno.h>
#include    <ip.h>

```

```

#include      <netioctl.h>
#include      <task.h>

#define ESCHAR 033

extern int    displa ();
extern int    us_opna (), us_opnl (), us_cls (), us_tmo (), us_space();
extern int    done ();
extern int    his_cls();

struct termio  stermio;

main (argc, argv)
int    argc;
char   **argv;
{
    in_name          fhost;
    register unshort fsock, lsock;
    register char    c;
    register int     i;
    register int     lstn = FALSE;
    register int     dbg = FALSE;
    struct termio    termio;
    register int     escseen = FALSE;

    if (argc < 3) {
        printf ("usage: %s [-l] [-d] host port\n", argv[0]);
        exit (1);
    }

    for (i = 1; i < argc && argv[i][0] == '-'; i++)

        switch (argv[i][1]) {
            case 'l':      /* listen */
                lstn = TRUE;
                break;

```

```

    case 'd':          /* debug */
        dbg = TRUE;
        tcpdebug (dbg);
        break;
    default:
        printf ("usage: %s [-l] [-d] host port\n", argv[0]);
        exit (1);
}

setbuf(stdout, 0);

ioctl(0,TCGETA,&termio);

stermio = termio;
termio.c_lflag = 070;
termio.c_cc[4] = 1;
termio.c_cc[5] = 1;
termio.c_oflag = ONLCR | OPOST;
termio.c_cflag &= &similarPARENB;
termio.c_cflag |= CS8;
termio.c_iflag = 0 | ICRNL;

ioctl(0, TCSETAF, &termio);

signal (SIGINT, done);
signal (SIGTERM, done);

if (!tcp_init (3000, 0, (lstn ? us_opnl : us_opna), displa, us_cls,
               his_cls, us_tmo, us_space)) {
    printf("tcp_init failed\n");
    exit (1);
}

if (lstn) {
    lsock = atoi(argv[i]);
    if (tcp_lstn (lsock, 1000) == FALSE) {
        printf("tcp_lstn failed: lsock %d\n", lsock);
        exit(1);
    }
}

```

```

        }
    }
else {

    if ((fhost = resolve_name (argv[i])) == 0L) {
        printf ("%s: unknown host\n", argv[i]);
        exit (1);
    }

    fsock = atoi(argv[i+1]);

    printf ("Trying...");

    if (tcp_open (fhost, fsock, 1000) == FALSE) {
        printf("tcp_open failed: fhost = %X, fsock = %d\n", fhost, fsock);
        exit(1);
    }
}

for (;;) {
    if (lstn) {
        tk_block ();
        continue;
    }
    c = getkey ();
    if (escseen) {
        escseen = FALSE;
        switch (c) {
case 'c':
                /* close connection */
                tcp_close ();
                break;
case 'e':
                /* echo local */
                ioctl(0,TCGETA,&termio);
                termio.c_lflag |= ECHO;
                ioctl(0,TCSETA,&termio);

```

```

                                break;
case 'r':                        /* echo remote */
                                ioctl(0,TCGETA,&termio);
                                termio.c_lflag &= &similarECHO;
                                ioctl(0,TCSETA,&termio);
                                break;
case 'q':                        /* quit */
                                tcp_close ();
                                done ();
                                break;
case 'v':                        /* quote next character */
                                c = 0;
                                for (i = 0; i < 3; i++)
                                    c = (c << 3) + (getkey () - '0');
                                tc_fput (c);
                                break;
case 'd':                        /* toggle debug state */
                                dbg = !dbg;
                                tcpdebug (dbg);
                                break;

default:
                                tc_fput (c);
                                break;

                                }
                                continue;
                                }
else
if (c == ESCHAR) {
    escseen = TRUE;
    continue;
}
tc_fput (c);
}
}

```

```
displa (buf, len, urg).
register char *buf;
register int len;
register int urg;
{

    register char *bp;

    char *endp = &buf[len];
    for (bp=buf;bp < endp;bp++)
        if (*bp & 0200) {
            write (1, buf, bp-buf-1);
            buf = bp + 1;
            printf("\\%o",(*bp & 0377));
        }
    write (1, buf, bp-buf);
}
```

```
us_opna ()
{
    printf ("\nActive connection opened.\n");
}
```

```
us_opn1 ()
{

    printf ("\nServer connection opened.\n");

}
```

```
us_space ()
{
}
```

```
us_cls ()
{
    printf ("Closed\n");
    done ();
}

his_cls()
{
    printf("Foreign closed\n");
    tcp_close();
}

us_tmo ()
{
    printf ("Host not responding\n");
    done ();
}

done ()
{
    ioctl(0,TCSETAW,&sternio);
    exit (0);
}

getkey ()
{
    char    c;
    int     nch;

    for (;;) {
        ioctl(0, TIONREAD, &nch);
        if (nch == 0) {
            tk_yield ();
            continue;
        }
        if (read (0, &c, sizeof(c)) == sizeof(c))
```

```
        return (c & 0377);  
    }  
}
```

Appendix C. tcp Library Routines

About This Appendix

This appendix describes a version of the **tcp** library routines that support a single **tcp** connection. It is provided only for compatibility with existing programs. The **tcpm** library, described under “**tcpm**” on page 4-3 should be used in the development of new programs.

tcp

tcp

Purpose

Provides the Transmission Control Protocol (TCP) layer.

Library

`/usr/lib/libtcp.a`

Syntax

```
#include <ip.h >  
#include <task.h >
```

Description

The TCP provides a set of routines suitable for reliable data transmission. It assumes that the receiver can keep pace with the sender and that it will never run out of receive window. Also it does not handle out of sequence packets received. Any packets that are out of sequence are ignored.

Note: The Interface Program supports a window size of 6K bytes and a packet size of 1576 bytes.

The TCP library provides some support for the internal tasking system within a single AIX process. Task calls must be issued by the user.

The routines in the `/usr/lib/libtcp.a` library interface with the IP library and the AIX

network driver. The **libtcp.a** library contains the following routines:

```

tcp_init (stksiz, 0, ofcn, infcn, cfcn, fcfcn, tmofcn, sfcn) int   stksiz;
int   extsiz;
int   (*ofcn) ();
int   (*infcn) (prt, len, urgent);
int   (*cfcn) (TRUE!FALSE);
int   (*fcfcn) ();
int   (*tmofcn) ();
int   (*sfcn) ();

```

Initializes the TCP layer. This routine starts the internal tasking system, initiates the timer and sets the pointers to the user routines that can be called. It does not attempt to open the connection. That function is performed by **tcp_open**. When it returns, the caller is running as the first task on the primary process stack.

tcp_init sets the following pointers:

- stksiz** Value for the depth that the thread or task will require.
- ofcn** Called once TCP reaches the ESTABLISHED state (both the local and foreign hosts are able to send data). Most commands use this routine to awaken a task that writes data to the network.
- infcn** Called to send some data to the user program. TCP does not acknowledge data until this routine returns. **infcn** sets the following pointers:
 - char * prt** Sets pointer to the beginning of the data.
 - int len** Returns the number of bytes of data sent.
 - ushort urgent** The normal value is -1. If the value is greater than 0, it implies urgent data in the buffer and points to it.
- cfcn** Called to signal that both sides have closed, and the connection block is erased. The returned status is **TRUE** if the connection closed normally, or **FALSE** if the connection closed abnormally due to a TCP RESET.
- fcfcn** Called if the foreign host sends a FIN (the foreign host has closed its half of the connection) before a close in the local host. This initiates an immediate close in the local host.
- tmofcn** Called when a timeout has occurred.
- sfcn** Called when the previously full output buffer has buffer space available. TCP immediately blocks the writing process, which prevents writing to a connection before the connection is open. Also, if TCP determines the foreign window is full, or when the current output packet is full, any attempt to write to the network blocks the writing process. In these

tcp

three cases, this call signals that the cause of the block has disappeared. The user routine should always awaken those tasks that are blocked.

tcp_open (*fh, fs, win*)
in_name *fh*;
ushort *fs*;
int *win*;

Opens an active **tcp** connection to foreign host *fh*, on foreign socket *fs*, with a receive window size of *win* bytes. Gets a unique local socket on which to open the connection. Returns FALSE if unable to open an Internet connection with the specified hosts and sockets. Otherwise it returns the local socket on which the connection is opened. This routine forks a child process to handle the connection; it does not wait until the connection is actually opened before it returns. Instead, the *ofcn* (user open routine) specified in the call to **tcp_init** is called when the connection is successfully opened. (The **tcp_init** call must precede the **tcp_open** call.) **open** allows a single AIX process on a single port; the tasking system allows multiple threads within each process.

tcp_close ()

Initiates the TCP closing sequence. This routine returns immediately. When the close is complete, the *cfcn* (user close routine) is called.

tcp_lstn (*ls, win*)
ushort *ls*;
int *win*;

Listens for a server connection on the specified local socket with the foreign host and port unspecified. Returns FALSE if unable to open an Internet connection with the specified hosts and sockets, or TRUE otherwise. This routine returns immediately. It does not wait until the connection is actually opened before returning. When a connection is successfully opened, a process performs a **fork** call while in the **tcp_rcv()** routine, the child process proceeds to open a specified Internet connection with the appropriate parameters, and the *ofcn* (user open routine) is in the child process. The parent returns to listen on the connection. **lstn** supports one TCP process on one or more ports; the tasking system allows multiple threads within each process.

tc_put (*c*)
char *c*;

Inserts a character into the send buffer for transmission, does not wake up the TCP sending task. It assumes that more data immediately follows. It returns TRUE if there is more room to store data in the buffer. Otherwise, it returns FALSE.

tc_fput (*c*)
char c;

Inserts a character to be transmitted into the send buffer, and wakes up the TCP sending task to send it. It returns TRUE if there is more space in the buffer to store data. Otherwise, it returns FALSE.

tcpurgent ()

Indicates that urgent data is present. Sets the urgent pointer to the current data length and awakens TCP to send it.

tcp_passive_open (*fh, fs, ls, win*)
in_name fh;
ushort fs;
ushort ls;
int win;

Indicates that the process will accept incoming connection requests rather than attempt to initiate a connection. Often the process requesting a passive OPEN will accept a connection request from any caller. In this case, a foreign socket of all zeros is used to denote an unspecified socket. Unspecified foreign sockets are allowed only on passive OPENS. In this case, *fh* specifies the foreign host, *fs* specifies the foreign socket, *ls* specifies the local socket, *win* specifies the receive window size. **passive_open** allows a single AIX process on a single port; the tasking system allows multiple threads within each process.

tcp_debug (*onoff*)
int onoff;

Turns TCP debugging (packet-level tracing) on or off.

tcp_abort ()

Makes a user-level request to abort connection.

tcp_flush ()

Wakes up a TCP send task so that all outstanding data will be sent over the TCP connection.

tcp

tc_write (*buf*, *count*)
char *buf;
int count;

Writes a block of data to a TCP connection. *buf* is a counter and *count* is an integer.

tm_on ()

Wakes up the timer task to process any events that went off while timer interrupts were disabled and to start a new alarm.

tm_off ()

Turns off timer interrupts; useful when writing data to the display.

tk_block

Blocks the current task and forces it to be rescheduled.

tk_yield

Yields the processor to any other task that can be run.

tk_wake (*tk*)
task *tk

Wakes up a specified task. The *tk* parameter is a pointer to a control block, which is defined in **task.h**.

tk_setef (*tk*, *ef*)
task *tk
task ushort ef

Wakes up a task and sets a specified event flag for it. The *tk* parameter is a pointer to a control block, which is defined in **task.h**. The *ef* parameter specifies the event flag.

Figures

1-1.	Interface Program for use with TCP/IP Commands, Protocols, and APIs	1-5
1-2.	Network and Gateway Routing	1-11
1-3.	Class A Address	1-16
1-4.	Class B Address	1-16
1-5.	Class C Address	1-16
1-6.	IP and ARP Type Numbers	1-17
1-7.	Local Header, IP or ARP Packet, Baseband Adapter	1-17
1-8.	Local Header, IP or ARP Packet, Token-Ring Adapter	1-17
1-9.	Medium Access Control (MAC) Header, Token-Ring Adapter Local Address	1-17
1-10.	MAC Header Routing Information, Token-Ring Adapter Local Address	1-18
1-11.	Logical Link Control (LLC) Header, Token-Ring Adapter Local Address	1-18
1-12.	Class B Address with Subnet	1-21
3-1.	lpd Requests	3-8
3-2.	lpd Control File Codes	3-9
4-1.	Relationship of Tasking to Memory Organization	4-12
5-1.	Tasks Performed, Internet Header	5-14
5-2.	Tasks Performed, Local Header	5-15
5-3.	Tasks Performed, TCP Header	5-15
5-4.	Tasks Performed, UDP Header	5-16

Glossary

This glossary contains a list of some of the common terms that you may read or hear when working with data communications. The terms are described only as they relate to data communications.

access. The manner in which files or data sets are referred to by the computer.

adapter. See *communications adapter*.

address field. The part of a packet containing addressing information.

addressing. (1) The way that the sending or control station selects the station to which it is sending data. (2) A means of identifying storage locations.

American National Standard Code for Information Interchange (ASCII). The code developed by ANSI for information interchange among data processing systems, data communications systems, and associated equipment. The ASCII character set consists of 7-bit control characters and symbolic characters.

American National Standards Institute (ANSI). An organization sponsored by the Computer and Business Equipment Manufacturers Association for establishing voluntary industry standards.

bandwidth. The difference, in hertz, between the two limiting frequencies of a band.

baseband system. A system whereby information is encoded, modulated, and impressed on the transmission medium. At any point on the medium, only one information signal at a time is present.

bit rate. The speed at which serialized data is transmitted, usually expressed in bits per second.

block. A group of records that is recorded, processed, or sent as a unit.

bridge. In the connection of local loops, channels, or rings, the equipment and techniques used to match circuits and facilitate accurate data transmission.

broadband. Transmission media and techniques that use a broad frequency range, divided into sub-bands of narrower frequency.

cable. The physical media for transmitting signals; includes copper conductors and optical fibers.

cancel. To end a task before it is completed.

carrier. A continuous frequency that can be modulated with a second (information-carrying) signal.

carrier sense multiple access with collision detection (CSMA/CD). The generic term for a class of medium access procedures that (1) allows multiple stations to access the medium at will without explicit prior coordination, (2) avoids contention by way of carrier sense and deference, and (3) resolves contention by way of collision detection and transmission.

channel. A path along which data passes.

character. A letter, digit, or other symbol.

client. On a network, the computer requesting services or data from another computer.

clock. A device that generates periodic signals used for synchronization.

coaxial cable. A cable consisting of one conductor, usually a small copper tube or wire, within and insulated from another conductor of larger diameter, usually copper tubing or copper braid.

collision. A condition caused by multiple overlapping transmissions on the medium, which results in garbled data.

communication channel. An electrical path that facilitates transmission of information from one location to another.

communications. See *data communications*.

communications adapter. A hardware feature that enables a computer or device to become a part of a data communications network.

communications line. The line over which data communications takes place; for example, a telephone line.

configuration. The group of machines, devices, and programs that make up a data processing system.

confirmation. A transmission by a receiver that permits a sender to continue.

console. A part of a computer used for communications between the operator or maintenance engineer and the computer.

console display. A display that can be requested only at the system console. From a console display an operator can display, send, and reply to messages and use all control commands.

contention. A condition on a communications channel when two stations attempt to use the same channel simultaneously.

contention resolution. The process of resolving contention (medium access control conflicts) according to a defined algorithm.

control block. A storage area used by a program to hold control information.

control character. A character, occurring in a particular context, that initiates, modifies, or stops any operation that affects the recording, processing, transmission, or interpretation of data (such as carriage return, font change, and end of transmission).

CSMA/CD. See *carrier sense multiple access with collision detection (CSMA/CD)*.

current host. See *local host*.

daemon. A background process that is usually started at system start, runs continuously, and performs a function required by other processes.

data circuit. Associated transmit and receive lines that provide a means of two-way data communications.

data communications. The transmission of data between computers and/or remote devices (usually over a long distance).

data link. The equipment and rules (protocols) used for sending and receiving data.

data stream. All information (data and control information) transmitted over a data link.

digital data. Data represented by on and off conditions called bits.

display station. A device that includes a keyboard from which an operator can send information to the system and a display screen on which an operator can see the information sent to or received from the computer.

distortion. An undesirable change in a data communications signal.

dotted decimal. A common notation for Internet host addresses, which divides the 32-bit address into four 8-bit fields. The value of each field is specified as a decimal number and the

fields are separated by periods (for example, 010.002.000.052, or 10.2.0.52).

echo. A reflected signal on a communications channel.

emulation. Imitation; for example, the imitation of a computer or device.

enable. In interactive communications, to load and start a subsystem.

foreign host. Any host on the network except the one at which a particular operator is working; sometimes called *remote host*.

gateway. An entity operating above the link layer, which translates, when required, the interface and protocol used by one network into those used by another distinct network.

host. (1) The primary or controlling computer in the communications network. (2) A computer attached to a network.

interface. A common boundary, but not of internal connections.

interrupt. To take an action at a receiving station that causes the sending station to end a transmission.

LAN. See *local area network*.

local. Pertaining to a device, file, or system that is accessed directly from your system, without the use of a communications line. Contrast with *remote*.

local area network. A network in which communications are limited to a moderate-sized geographic area (1 to 10 km) such as a single office building, warehouse, or campus. A local network depends upon a communications medium capable of moderate to high data rate, and normally operates with a consistently low error rate.

local host. The host on the network at which a particular operator is working; sometimes called *current host*.

medium (media). The material in or on which data may be represented (for example, twisted pairs, coaxial cables, and optical fibers).

network. A collection of data processing products connected by communication lines for information exchange between stations.

network adapter. Circuitry that allows devices using a directly attached network to communicate with the system.

network management. The conceptual control element of a data station that interfaces with all of the layers of that data station and is responsible for the resetting and setting of control parameters, obtaining reports of error conditions, and determining if the station should be connected to or disconnected from the medium.

node. See *host*.

packet. The data of one transaction between a host and its network. A packet usually contains a network header, followed by one or more headers used by high level protocols, followed by data blocks.

physical layer (or level). The lowest layer of network design as specified by the ISO Open System Interconnection (OSI) reference model. This layer is responsible for interfacing with the medium, detecting and generating signals on the medium, and converting and processing signals received from the medium and from the data link layer.

port. (1) An access point for data entry or exit. (2) An entrance to or exit from a network.

protocol. Rules for transferring data.

process. A program in execution.

remote. Pertaining to a device, file, or system that is accessed by your system through a communications line. Contrast with *local*.

remote host. See *foreign host*.

retransmit. To repeat the transmission of a message or segment of a message.

retry. To resend a transmission that did not achieve the desired or intended result; usually follows a timeout.

route. A path defined for sending data across a network.

route table. A structure in memory that describes, for the computer, all of the routes that are currently defined.

server. On a network, the computer that contains the data to be accessed.

session. The logical connection by which a host program or device can communicate with a program or device at a remote location.

socket. (1) A unique host identifier created by the concatenation of a port identifier with an IP address. (2) A port identifier.

status. The state of affairs or the condition of a station that determines its ability to enter into exchanges of control or information.

telecommunications. Transmitting signals over long distance.

teleprocessing. Processing data that is received from or transmitted to a remote location via communication channels.

terminal. A device, usually equipped with a keyboard and a display device, capable of sending and receiving information over a communications line.

timeout. Measurement of time interval allotted for certain events to occur (such as a

response to polling or other controls) before corrective (recovery) action is taken.

transfer. To send data to one place and to receive data at another place.

transmission control characters. Special characters that are included in a message to control communication over a data link. For example, the sending station and the receiving station use transmission control characters to exchange information; the receiving station uses transmission control characters to indicate errors in data it receives.

transparent. In communications, pertaining to transmissions that have no possibility of interference with data link control, regardless of format or content. Transparent transmissions are unrecognized by data link controls.

well-known host name. A conventional name associated with an IP address on a particular network (for example, **nameserver** and **timeserver**).

well-known port. A conventional port assignment used by hosts that support the same protocols, whether or not the hosts are on the same network.

wide area network. A network that provides data communication capability in geographic areas larger than those serviced by local area networks.

work station. A device that lets people transmit information to or receive information from a computer; for example, a display station or printer.

A

additional copies of this book vii
additional information vi
address field
 definition of X-3
 IP 1-15
Address Resolution Protocol
 See ARP
addressing
 See also naming
 address types
 class A 1-16
 class B 1-16
 class C 1-16
 class D 1-16
 definition of X-3
 IP 1-15
 IP address field 1-15
 notation
 dotted decimal 1-19
 octal 1-19
 sockets 1-20
 TCP 1-20
API
 for `/dev/net0` 5-3
 illustration 1-5
 to libraries
 internet 4-19
 libS 4-24
 tcp C-2
 tcpm 4-3
 udp 4-14
Application Programming Interfaces
 See API
applications
 See commands
ARP
 addresses 1-16

 overview 1-7
assigned numbers
 ports 1-38
 protocols 1-39
 versions 1-39

B

Baseband Adapter 1-4
broadcast messages 1-22

C

calls
 See `/dev/net0`
chparm
 host name consistency 1-19
 warning 1-19
client
 definition of X-3
 term usage 1-3
commands
 See also server commands, user commands
 file transfer
 tcom 2-37
 tftp 1-13, 2-40
 tftpd 3-25
 utftp 1-13, 2-46
 xftp 1-13, 2-48
 xftpd 3-27
 illustration 1-5
mail
 maild 3-11
 netmail 1-14, 2-16
 smtp 1-14
 smtpd 3-21

network management
finger 2-3
fingerd 3-3
host 1-15, 2-6
hostname 1-15, 2-8
icmpd 3-5
named 3-13
netconfig 1-15
netstat 1-15, 2-19
ping 1-15, 2-23
route 1-15, 2-29
setclock 1-15, 2-32
timed 3-26
remote command execution
 /etc/hosts.equiv 1-28
 lpd 3-7
 lprbe 2-10
 rexec 2-26
 rexecd 3-15
remote login
 telnetd 3-23
 tn 1-14, 2-43
routed 3-17
configuration
 See also customizing
 /etc/hosts 1-25, 1-41
 /etc/hosts.equiv 1-41
 /etc/net 1-29, 1-41
 /etc/networks 1-41
 /etc/rc.tcpip 1-42
 definition of X-4
 EDITOR 1-41
 finger 1-42
 variables 1-41
configuring multiple adapters 2-13
contention
 definition of X-4
 resolution
 definition of X-4
customizing
 /etc/hosts 1-25
 /etc/net 1-29
 steps A-2

D

daemons
 fingerd 3-3
 icmpd 3-5
 named 3-13
 smtpd 3-21
 telnetd 3-23
 timed 3-26
 xftpd 3-27
DARPA vi
data security 1-41
Defense Advanced Research Projects
 Agency vi
 Defense Communications Agency vi
determining host addresses 2-6
/dev/net0
 calls
 close 5-4
 open 5-4
 read 5-4
 write 5-5
 incoming packets supported 5-3
 IOCTLs 5-5
 tasks
 kernel 5-14, 5-15
 user process 5-14, 5-15, 5-16
device driver
 See **/dev/net0**
displaying local host name 2-8
dotted decimal
 definition of X-4
 notation 1-19

E

echo request 2-23
emulation
 definition of X-5
 terminal 1-14
/etc/gateways 1-23
examples 1-24

- format 1-23
- /etc/hosts**
 - examples 1-26
 - format 1-25
 - host name consistency 1-19
- /etc/hosts.equiv**
 - format 1-28
- /etc/master**
 - host name consistency 1-19
- /etc/net**
 - format 1-29
- /etc/networks** 1-31
 - format 1-31
- /etc/rc.tcpip**
 - purpose 1-33
 - relation to **/etc/rc** 1-33
- /etc/system**
 - host name consistency 1-19
- executing commands remotely 2-26, 3-15

F

- file formats
 - .netrc** 1-34
 - .3270keys** 1-36
 - gateways** 1-23
 - hosts** 1-25
 - hosts.equiv** 1-28
 - net** 1-29
 - networks** 1-31
 - overview of 1-22
 - rc.tcpip** 1-33
- file transfer
 - commands 2-2, 3-2
 - type 2-48
 - ASCII 2-48
 - binary 2-48
- files
 - See file formats
- finger** command 2-3
- fingerd** command 3-3

G

- Gateway-to-Gateway Protocol
 - See GGP
- gateways
 - definition of X-5
 - description 1-6
 - using hosts as 1-10
 - illustration 1-11
- GGP
 - in routing 1-8

H

- hardware prerequisites iv
- header
 - local 1-17
- \$(HOME)/.netrc** 1-34
 - format 1-34
- home 3270 keys 1-36
- \$(HOME)/.3270keys**
 - format 1-36
- host** command 2-6
- hostname** command 2-8
- hosts
 - as gateways 1-10
 - definition of X-5
 - foreign 1-3
 - definition of X-5
 - local 1-3
 - definition of X-5
 - names 1-19

I

- IBM RT PC Baseband Adapter for use with Ethernet 1-4
- IBM RT PC Interface Program for use with TCP/IP
 - customizing A-2

installing iv, A-1
icmpd command 3-5
ICMP overview 1-8
information about users 2-3
information protection 1-41
installing the Interface Program A-1
interface
 definition of X-5
international character support considerations
 tn command 2-44
Internet Control Message Protocol 1-8
Internet environment 1-4
internet library 4-19
Internet Protocol
 overview 1-6
Internet Router 1-10

L

libraries
 internet 4-19
 libS 4-24
 tcp B-1, C-2
 tcpm 4-3, B-1
 usage sample B-1
 udp 4-14
libS library 4-24
lpd command 3-7
lprbe command 2-10

M

mail commands 2-2, 3-2
maild command 3-11
manipulating route tables 2-29
messages
 broadcast 1-22

N

named command 3-13
nameserver 1-25
naming
 See also addressing
 host
 conventions 1-19
netconfig command 2-13
netmail command 1-14, 2-16
netstat command 2-19
network
 customizing A-2
 definition of X-5
 problem determination 2-19
 status 2-19
network adapter
 definition of X-5
Network Information Center vi
network management
 commands 2-2, 3-2
 definition of X-5
 sub-networks 1-21
networks
 &I2@nets.
 networks 1-31
 IBM RT PC Baseband Adapter for use with
 Ethernet 1-4
nodes
 See hosts
numbers
 See assigned numbers

O

octal notation 1-19
ordering this book vii

P

packets
 definition of X-5
 description 1-3
ping command 2-23
pipes 2-46
plan file 2-3
ports
 definition of X-5
 identification 1-4
 numbers 1-38
 well-known 1-38
printing remotely 2-10, 3-7
process
 definition of X-5
 description 1-3
programming interfaces
 /**dev/net0** 5-3
 illustration 1-5
 internet 4-19
 libS 4-24
 tcp C-2
 tcpm 4-3
 udp 4-14
project file 2-3
protocols
 Address Resolution Protocol 1-7
 ARP 1-7
 definition of X-5
 illustration 1-5
 Internet 1-6
 IP
 addressing 1-15
 lpd 1-9
 minimal 2-40
 numbers 1-39
 other network
 GGP 1-8
 ICMP 1-8
 overview 1-8
 overview 1-4
 remote command execution 1-9
 remote printing 1-9

rexecd 1-9
RIP 1-9
TCP
 addressing 1-20
 overview 1-7
 sockets 1-20
Telnet 1-14
UDP
 overview 1-7
VAX trailer 5-3
VAX trailer encapsulation protocol 1-9

R

rc.tcpip file 1-33
related information vi
related publications vi
retry
 definition of X-6
 definition X-6
rexec command 2-26
rexecd command 3-15
route command 2-29
routed command 3-17
routes
 broadcast to all 1-22
 definition of X-6
 Internet Router 1-10
 illustration 1-11
 route table
 description 1-20
 gateways 1-23
 routed 3-17
 Routing Information Protocol 1-9
 sub-networks 1-21
 types 1-20
routines 4-2
 internet layer 4-19
 libS layer 4-24
 tcp layer C-2
 tcpm layer 4-3
 udp layer 4-14
routing 3-17
Routing Information Protocol 1-9

S

samples
 tcptest.c B-12
 tcptestm.c B-3
security
 considerations 1-41
 features
 lprbe 1-40
 rexec 1-40
 Telnet 1-40
 xftp 1-40
 information protection 1-41
 sending mail 2-16, 2-34, 3-11
server
 definition of X-6
 term usage 1-4
server commands
 fingerd 3-3
 icmpd 3-5
 lpd 3-7
 maild 3-11
 named 3-13
 rexecd 3-15
 routed 3-17
 smtpd 3-21
 telnetd 3-23
 tftpd 3-25
 timed 3-26
 xftpd 3-27
session
 definition of X-6
setclock command 2-32
setting time 2-32
Simple Mail Transfer Protocol 1-14
sntp command 2-34
 reply codes 2-34
smtpd command 3-21
sockets
 definition of X-6
 in routines 4-2

 in TCP addressing 1-20
 software prerequisites iv
 special characters
 n acute (lowercase) 2-44
 therefore 2-44
 sub networks 1-21
 subnets 1-21
 system calls
 See **/dev/net0**

T

tasking system
 description 4-11
 usage sample B-1
tcom command 2-37
 subcommands 2-37
TCP
 overview 1-7
tcp library C-2
tcpm library 4-3
Telnet 1-14
telnetd command 3-23
tftp command 2-40
 minimal
 tftp 2-40
tftpd command 3-25
time
 service 3-26
 setting 2-32
timed command 3-26
timeserver 1-25
tn command 2-43
 subcommands 2-43
trailer encapsulation 1-9, 5-3
Transmission Control Protocol
 See TCP
type numbers
 ARP 1-17
 IP 1-17
typography in this book v

U

UDP

overview 1-7

udp library 4-14

user commands

finger 2-3**host** 2-6**hostname** 2-8**lprbe** 2-10**netconfig** 2-13**netmail** 2-16**netstat** 2-19**ping** 2-23**rexec** 2-26**route** 2-29**setclock** 2-32**smtp** 2-34**tcom** 2-37**tftp** 2-40**tn** 2-43**utftp** 2-46**xftp** 2-48

User Datagram Protocol

See UDP

utftp command 2-46**V**

VAX trailers 1-9, 5-3

version numbers 1-39

W

warnings

host name consistency 1-20, 2-8

window size 2-48

X**xftp** command 2-48

packet size 2-48

subcommands 2-48

window size 2-48

xftpd command 3-27



The IBM RT Personal
Computer
Family

Reader's Comment Form

IBM RT PC Interface Program
for use with TCP/IP

SC23-0812-0

Your comments assist us in improving our products. IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

For prompt resolution to questions regarding set up, operation, program support, and new program literature, contact the authorized IBM RT PC dealer in your area.

Comments:

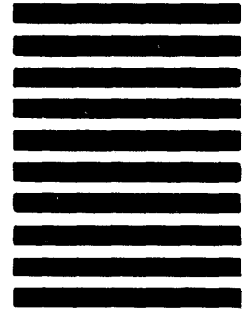


NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department 997, Building 998
11400 Burnet Rd.
Austin, Texas 78758



Fold and tape

Fold and tape

Cut or Fold Along Line

Tape

Please Do Not Staple

1ape

Book Title**Order No.****Book Evaluation Form**

Your comments can help us produce better books. You may use this form to communicate your comments about this book, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you. Please take a few minutes to evaluate this book as soon as you become familiar with it. Circle Y (Yes) or N (No) for each question that applies and give us any information that may improve this book.

Y N Is the purpose of this book clear?

Y N Is the table of contents helpful?

Y N Is the index complete?

Y N Are the chapter titles and other headings meaningful?

Y N Is the information organized appropriately?

Y N Is the information accurate?

Y N Is the information complete?

Y N Is only necessary information included?

Y N Does the book refer you to the appropriate places for more information?

Y N Are terms defined clearly?

Y N Are terms used consistently?

Y N Are the abbreviations and acronyms understandable?

Y N Are the examples clear?

Y N Are examples provided where they are needed?

Y N Are the illustrations clear?

Y N Is the format of the book (shape, size, color) effective?

Other Comments

What could we do to make this book or the entire set of books for this system easier to use?

Optional Information

Your name _____

Company name _____

Street address _____

City, State, ZIP _____

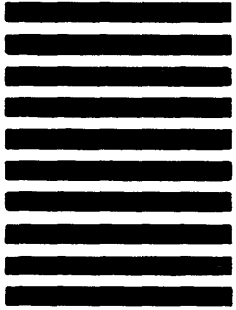


NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department 997, Building 998
11400 Burnet Rd.
Austin, Texas 78758



Fold and tape

Fold and tape

— Cut or Fold Along Line —

Tape

Please Do Not Staple

Tape

© IBM Corp. 1987
All rights reserved.

International Business
Machines Corporation
Department 997, Building 998
11400 Burnet Rd.
Austin, Texas 78758

Printed in the
United States of America

SC23-0812-0



SC23-0812-00



92X1291