The RadiSys logo is a blue rectangular box with the word "RadiSys." in white serif font. A thin black line extends from the right side of the box, ending in a small circle, and then continues vertically down the page.

RadiSys.

# **iRMX<sup>®</sup>** **TCP/IP for the iRMX OS**

RadiSys Corporation  
5445 NE Dawson Creek Drive  
Hillsboro, OR 97124  
(503) 615-1100  
FAX: (503) 615-1150  
[www.radisys.com](http://www.radisys.com)  
07-0629-01  
December 1999

EPC, iRMX, INtime, Inside Advantage, and RadiSys are registered trademarks of RadiSys Corporation. Spirit, DAI, DAQ, ASM, Brahma, and SAIB are trademarks of RadiSys Corporation.

Microsoft and MS-DOS are registered trademarks of Microsoft Corporation and Windows 95 is a trademark of Microsoft Corporation.

IBM and PC/AT are registered trademarks of International Business Machines Corporation.

Microsoft Windows and MS-DOS are registered trademarks of Microsoft Corporation.

Intel is a registered trademark of Intel Corporation.

All other trademarks, registered trademarks, service marks, and trade names are property of their respective owners.

December 1999

Copyright © 1999 by RadiSys Corporation

All rights reserved.

# Quick Contents

---

## Getting Started and User's Guide

- Chapter 1. Overview of TCP/IP
- Chapter 2. Installing and Starting TCP/IP
- Chapter 3. Using TELNET
- Chapter 4. Using File Transfer Protocols

## Network Administration

- Chapter 5. Network Services and Daemons
- Chapter 6. Configuring and Administering Network Files
- Chapter 7. Commands for the Network Administrator
- Chapter 8. Tunable Parameters

## Reference

- Chapter 9. Files
- Chapter 10. TCP/IP Components
- Chapter 11. Library Functions
- Appendix A. Recommended Reading
- Glossary
- Index

# Notational Conventions

This manual uses these conventions:

- All numbers are decimal unless otherwise stated. Hexadecimal numbers include the H radix character (for example, 0FFH) or a leading 0x (for example, 0x0FF).
- Bit 0 is the low-order bit unless otherwise stated.
- Syntax is printed like this.
- In interactive sessions, computer output is printed like this and user input appears like this.
- **System call names, command names, and processes like jobs or daemons appear in bold.**

Directory names and filenames are shown as seen from the iRMX prompt. To access files from the DOS prompt on an DOSRMX system, use a backslash (\) in pathnames rather than the forward slash (/) shown here.

Filenames are shown as they would appear on the iRMX or UNIX operating systems. On an DOSRMX system using the EDOS file driver, some filenames are truncated to match the DOS 8.3 character limits. For example, the *arpbypass* utility is installed as *arpbypas*, and its associated help file is *arpbypas.hlp*. From the iRMX prompt you can use either the truncated name or the full name to view such filenames or to invoke utilities.

This manual uses this to indicate command syntax; do not enter these characters as shown:

- [ ] Surrounds optional items
- | Separates one or more items, from which you choose one
- italic* A variable name. Do not enter as shown; substitute the appropriate item, such as a command, value, or filename.



## Note

Notes indicate important information.



## CAUTION

Cautions indicate situations that may damage hardware or data.

# Contents

---

---

## 1 Overview of TCP/IP

Connecting to Network Resources.....	1
Using TCP/IP Programs and Utilities .....	3
Administering TCP/IP .....	5
Programming with TCP/IP .....	5
Understanding Internet Addresses .....	6
Subnet Addresses.....	7
Special Addresses.....	7
Obtaining an Internet Address .....	8
Specifying Domain Names.....	9
Request For Comment (RFC) Reports.....	10

---

## 2 Installing and Starting TCP/IP

Before You Begin.....	11
Software Required.....	12
Hardware Required.....	13
Overview of the Setup.....	13
TCP/IP Configuration.....	13
Editing the Hosts File .....	14
Using DNS .....	14
Not Using DNS .....	14
Configuring TCP/IP as a Loadable Job .....	14
Editing the tcpstart.csd File .....	15
Starting and Stopping TCP/IP .....	15
Testing the TCP/IP Setup .....	16
Troubleshooting.....	16
General TCP/IP Debugging.....	16
Setting Up a Remote Unix Host for Telnet.....	17
Creating a Terminal Definition for the PC Console .....	17
Setting Terminal Characteristics for User Sessions.....	18

---

<b>3</b>	<b>Using Telnet</b>	
	Before You Begin.....	21
	Telnet Modes.....	21
	Starting TELNET.....	22
	Starting in Input Mode.....	22
	Starting in Command Mode.....	22
	Switching Telnet Modes.....	23
	Using TELNET for a Remote Session.....	23
	Connecting to the Remote Host.....	24
	Setting the Terminal Type on a Unix System.....	25
	Terminal Type Strings.....	25
	Disabling Local Echo on Berkeley Unix Hosts.....	26
	Entering Commands During the Session.....	26
	Closing the Remote Connection.....	27
	Using Telnet for a Local Session.....	28
	Entering Commands in a Local Session.....	28
	Ending the Local Session.....	28

---

<b>4</b>	<b>File Transfer Protocols</b>	
	Before You Begin.....	29
	File Transfer Protocol (FTP).....	30
	FTP Help Information.....	30
	FTP File Transfer Session.....	31
	Connecting to the Remote Host.....	31
	Using FTP Commands.....	32
	Ending the FTP Session.....	35
	FTP Initialization File.....	35

---

<b>5</b>	<b>Network Services and Daemons</b>	
	Ftpd Server.....	38
	Telnetd Server.....	39
	Configuring Pseudo-terminals for Telnetd.....	39

---

<b>6</b>	<b>Configuring and Administering Network Files</b>	
	Restricting and Updating Network Databases and Files .....	41

---

<b>7</b>	<b>Commands for the Network Administrator</b>	
	Administrative Commands .....	43
	Performing Network Tests.....	43
	Verifying Network Services .....	44
	Network Status Test .....	44
	Verifying Network Configuration .....	45
	Interface Status Test .....	45
	Verifying Interface Functionality .....	46

---

<b>8</b>	<b>Tunable Parameters</b>	
	Determining When to Tune Parameters.....	49
	TCP/IP Parameters .....	50
	[TCP].....	50
	[UDP].....	50
	[RIP].....	50
	[IP] .....	50
	[ETH0] .....	51
	[LO0].....	51
	TCP Job Parameters .....	51
	UDP Job Parameters.....	53
	Raw IP Job Parameters.....	54
	IP Job Parameters .....	55
	DNS Configuration Parameters .....	56
	Network Interface Parameters .....	56
	Loopback Pseudo-driver Interface Parameters .....	57

---

<b>9</b>	<b>Files</b>	
	hosts.....	60
	protocols .....	62
	netrc.....	64
	services .....	66

---

## 10 TCP/IP Components

Protocol Jobs .....	70
ip.job .....	71
rip.job .....	72
tcp.job.....	73
udp.job.....	74
Network Interface Controller (NIC) Jobs .....	69
loopback.job .....	70
edl.job.....	71
eepr100.job.....	72
ne.job.....	73
Tulip.job.....	74

---

## 11 Library Functions

Using Sockets .....	78
Calling Sequence for Connection-oriented Applications.....	78
Calling Sequence for Connectionless Applications .....	79
Internet Socket Addresses .....	80
Network and Host Byte Order .....	81
Changes From the Standard Socket Interface.....	82
Task Priority.....	82
Multitasking Considerations.....	82
Include Files .....	83
Example Programs .....	83
Compiling.....	84
Handling Errors .....	84
Errno Values for Network Functions.....	84
Function Reference.....	87
accept .....	88
bind.....	90
bstring.....	92
byteorder .....	93
connect .....	94
ffs .....	96
gethostent .....	97
gethostid .....	100
gethostname.....	101
getnetent .....	102
getpeername .....	104
getprotoent.....	106
getservent .....	108

getsockname .....	110
getsockopt.....	112
inet.....	115
listen .....	117
recv.....	119
send .....	122
shutdown .....	124
socket.....	126
socktout .....	75

---

## A Recommended Reading

TCP/IP.....	77
NFS.....	77
Networks .....	78

---

<b>Glossary</b> .....	79
-----------------------	----

---

<b>Index</b> .....	87
--------------------	----

---

## Tables

Table 11-1. Functions in the Socket Library .....	76
Table 11-2. Functions in the Network Library (continued).....	77

---

## Figures

Figure 1-1. Hosts Connected on a Network .....	2
Figure 2-1. How TCP/IP Works with iNA 960 Software.....	12
Figure 3-1. TELNET Modes.....	22



---

TCP/IP programs are based on a set of protocols called Transmission Control Protocol/Internet Protocol (TCP/IP). The TCP/IP suite of networking protocols makes it possible for different brands of computers, running different operating systems, to supply resources to network users.

This manual describes how to install, use, and maintain TCP/IP networking software on your iRMX<sup>®</sup> Operating System (OS). This software allows you to communicate across a network with any other computer running TCP/IP software, regardless of its operating system.

## Connecting to Network Resources

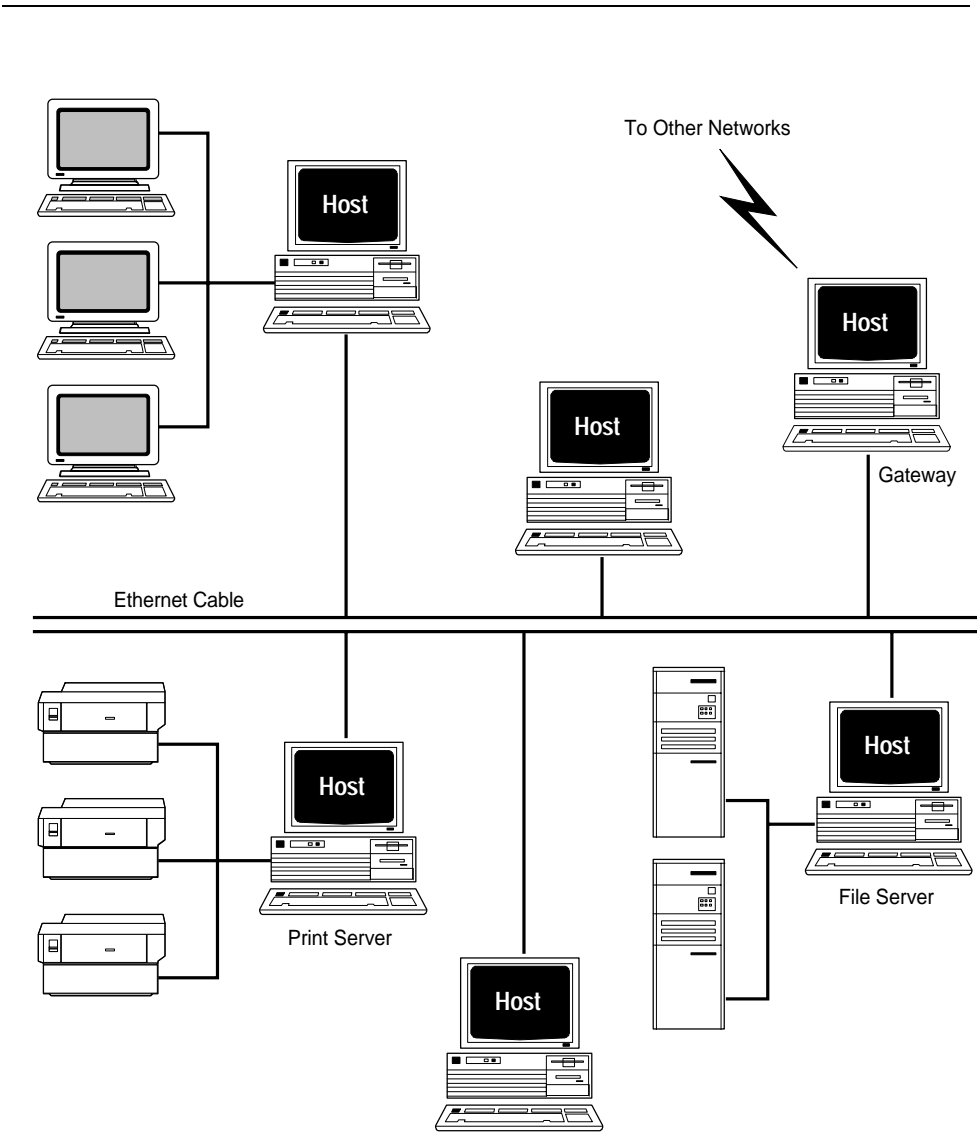
Individual computers on a computer network are called *hosts*. TCP/IP software lets you connect to various hosts on a network so that you can use their resources. The computer you use to make your original connection to the network is the *local host*. Any other computer on the network, regardless of its location, is a *remote host*.

Each host on a network is identified by a number, called an Internet address or IP address, and an official name. Hosts can also have symbolic names, called *aliases*. To access a remote host, you must specify either its Internet address, official name, or a valid alias to network software.

The computer and software that originate a network command are the *client*, because they request a network service. The computer and software responding to the request are the *server*, because they provide the network service. Servers provide *sharable resources*; the network gives shared access to many users.

Host configurations and sharable resources vary with individual networks. Check with your network administrator to determine the layout of your network and the resources available to you.

Figure 1-1 on page 2 illustrates network connections and possible resources. Print Servers and File Servers have special responsibilities: they provide network printer and file storage resources. iRMX systems cannot function as print servers. The host labeled Gateway acts as a connection, or *router*, to other networks, whose resources can also be accessed. iRMX systems that function as Gateways must be Multibus II systems.



W-3402

**Figure 1-1. Hosts Connected on a Network**

---

## Using TCP/IP Programs and Utilities

To use TCP/IP programs and utilities, you enter network commands at the iRMX command line. After you enter a command, TCP/IP software running on the local host cooperates with TCP/IP software running on the remote host to handle your transaction.

You can use iRMX TCP/IP programs and utilities in these ways:

- Network File System (NFS) support allows you to access remote devices on hosts who use iRMX or non-iRMX operating systems.
- The **telnet** program connects to a remote host that runs a TELNET server.
- File Transfer Protocol (FTP) connects to a host that runs an FTP server and transfers files between hosts.

The TELNET service, sometimes called the Virtual Terminal Program, provides access to remote hosts on your network and allows you to use them as if your terminal is directly connected to the remote computers. While TELNET is running, you can submit commands to control the remote session and get information about it. To connect to a remote host, you must have the appropriate authorization and know how to use its OS.

See also: *Chapter 3, Using TELNET;*  
**telnet** command, *Command Reference*

FTP transfers files between any two accessible network hosts supporting TCP/IP, regardless of their OSs. FTP accepts user commands to control the transfer process and perform additional operations. You don't need to know the OS on the remote host in order to use FTP. However, you must know the pathnames, filenames, and names of hosts involved in the transfer.

See also: *Chapter 4, Using File Transfer Protocols;*  
**ftp** command, *Command Reference*

TCP/IP includes query commands such as **hostid**, **hostname**, **netstat**, and **showmount**. The **hostid** command displays or sets the Internet address of the local system. The **hostname** command displays or sets the system name. The **netstat** command symbolically displays the contents of network-related data structures to show the status of active connections (default), configured interfaces, routing tables, network statistics, STREAMS buffer allocation failures, and packet traffic. The **showmount** command reports information on NFS-shared file systems.

See also: **hostid**, **hostname**, **netstat** and **showmount** commands, *Command Reference*

---

Trivial File Transfer Protocol (TFTP) transfers publicly readable files to and from a remote host. It requires no remote login. However, you must know the pathnames, filenames, and names of the hosts involved in the transfer.

See also: *Chapter 4, Using File Transfer Protocols;*  
**tftp** command, *Command Reference*

The Remote Copy, or **rcp**, command copies files between TCP/IP hosts that support the Berkeley R-series commands. Files can be copied between your local host and a remote host, or between two remote hosts in the network. **Rcp** will copy multiple files from directories on the source host to equivalent directories on the destination host. **Rcp** differs from FTP in that it uses an automatic login validation procedure, can be used only to copy files, and does not provide interactive session control. To use **rcp** you must have authorization to use the remote host.

See also: *Chapter 5, Network Services and Daemons;*  
**rcp** command, *Command Reference*

The Remote Shell, or **rsh**, command executes a command or program on remote hosts that support the Berkeley R-series commands. **Rsh** differs from **rlogin** in that it executes only one command at a time. When a foreground command finishes processing, or when a background command has been initiated, **rsh** terminates and returns control to your local host. To use **rsh** you must have authorization to access the remote host.

See also: *Chapter 5, Network Services and Daemons;*  
**rsh** command, *Command Reference*

The Remote Login, or **rlogin**, command establishes a connection to a remote host that supports the Berkeley R-series commands. You can then use the remote host as if your terminal were directly cabled to it. Sessions with **rlogin** are similar to those with TELNET, except that **rlogin** uses an automatic login validation procedure and does not accept user commands for controlling the remote session.

---

To use **rlogin**, you must have authorization to access the remote host and know how to use its operating system.

See also: *Chapter 5, Network Services and Daemons;*  
**rlogin** command, *Command Reference*

The **ruptime** command is a Berkeley R-series command that prints a table containing status information for each host on the network that runs the **rwhod** daemon. The command obtains its information from a report supplied by each host on the local area network (LAN) every three minutes.

See also: *Chapter 5, Network Services and Daemons;*  
**ruptime** command, *Command Reference*

The **rwho** command is a Berkeley R-series command that prints the host name, login name, terminal number, and login time for each user on the LAN. The output from this command is similar to that from the Unix system **who** command, but includes users on other network hosts.

See also: *Chapter 5, Network Services and Daemons;*  
**rwho** command, *Command Reference*

## Administering TCP/IP

Chapter 2 describes a minimal configuration needed to start using TCP/IP. There are other files you can configure and special commands to control and test the configuration. If you are the network administrator or are configuring your own host machine, you should understand how to use these files and commands.

For example, one item you can configure is an FTP server (called the **ftpd** daemon). You can set it up so remote hosts can use FTP to transfer files to and from your local host.

See also: Chapter 2, Installing and Starting TCP/IP;  
Chapters 6 through 9

## Programming with TCP/IP

There are two general types of programmatic interface to TCP/IP software: the TLI calls defined by Unix System V operating systems and socket calls made popular by Berkeley Unix. The iRMX implementation provides a socket interface. You can write applications that make both iRMX system calls and socket calls, or you can port existing socket applications to this interface.

See also: *Chapter 10, TCP/IP Components*

---

# Understanding Internet Addresses

To make entries in the configuration files you need to understand the format of Internet addresses. You will also use either Internet addresses or host names (and aliases) that represent addresses when communicating with remote systems. If you already know the format of Internet addresses and names that represent them, proceed with the installation and configuration instructions in Chapter 2.

The Defense Advanced Research Projects Agency (DARPA) Internet protocol family is a collection of protocols that utilize the Internet address format. This family includes the Transmission Control Protocol (TCP), Internet Protocol (IP), Internet Control Message Protocol (ICMP), and User Datagram Protocol (UDP). A raw interface is also provided to IP and ICMP.

Internet addresses are also called IP addresses; they use the IP routing protocol. An IP address is a 4-byte quantity. It is a (*net,host*) pair, where *net* identifies a network and *host* identifies a host on that network. There are three basic classes of address, as distinguished by the high-order bits of the address. Class A addresses use an 8-bit *net* and a 24-bit *host*; the high-order bit is 0. Class B addresses use a 16-bit *net* and a 16-bit *host*; the high-order bits are 10. Class C addresses use a 24-bit *net* and an 8-bit *host*; the high-order bits are 110.

Because of the size of the host part of an address, the different classes of address correspond to networks of varying size. The format of the addresses is shown below, along with the number of hosts possible in each class:

Class	Byte 1	Byte 2	Byte 3	Byte 4	Number of Hosts		
A	0	Net			Host	16,777,214	
B	10		Net		Host	65534	
C	110			Net		Host	254

↑  
High-order bits

OM03651

The dot notation form of an Internet address consists of one to four numbers separated by dots (.). Each number can be expressed in decimal, octal (leading 0), or hexadecimal (leading 0x).

The most common format is a four-part address (*a.b.c.d*), consisting of four 8-bit decimal numbers in the range 0-255. This is called dotted-decimal notation. The four parts are assigned, in order, to the four bytes in the Internet address.

See also: **inet** function, Chapter 11, for more information about dot notation

---

You can distinguish between the classes of address by the first number of a dotted-decimal address. Class A addresses begin with numbers in the range 1-126. (Value 127 is a special case used for the loopback device, described later in this manual.) Class B addresses begin with numbers in the range 128-191. Class C addresses begin with numbers in the range 192-223 (there are other special classes of network in the range 224-255).

Once you know the class of an address, you can tell which part of the address specifies the network and which specifies the host. For example, in a Class A address, the first byte is the network number and the last three bytes specify the host. In the address 89.3.240.9, the network is number 89, and the host is number 3.240.9 on that network. The host address is 89.3.240.9, because it must be specified in terms of its network. The network address is 89.0.0.0, which can be interpreted as any host on that network.

See also: *hosts* and *networks* files, Chapter 9

⇒ **Note**  
iRMX TCP/IP does not support IP multicast addressing (or IP multicasting or multicast addressing).

## Subnet Addresses

Sites may implement subnet addressing to accommodate a cluster of local networks. Subnet addressing further divides the local host portion of the address into a subnet part and a host part. Within the local cluster, each subnet appears to be an individual network; externally, the entire cluster appears to be a single network. In the example address 89.3.240.9, you might choose to use one byte of the host part to designate subnets. In that case, you would interpret the host to be number 240.9 on subnet 3 of network 89.

You enable subnet addressing by specifying a subnet mask for a network interface and by using the subnet mask when setting up the routes to each subnet.

See also: **ifconfig** and **route** commands, Chapter 9

## Special Addresses

Addresses of all 0s or all 1s are special cases and are not assigned to hosts. The address 0.0.0.0 means the local host. The address 255.255.255.255 broadcasts to all hosts on the network to which you are directly connected. An address with the host part set to all 1s broadcasts to all hosts on a specific network.

---

In a program, use the local address `INADDR_ANY` to do wildcard matching on incoming messages and to mean the local host on outgoing messages. Use the distinguished address `INADDR_BROADCAST` to broadcast on the primary network interface if it supports broadcast. These and other Internet-specific data types are defined in the include (header) file `<netinet/in.h>`. This file is installed in the `/intel/include` directory.

## Obtaining an Internet Address

On any network that is connected to the Internet, you must obtain a block of Internet addresses from a central authority, the InterNIC Registration Services. This prevents you from using the same Internet address as someone else on the Internet.

The InterNIC issues a network number whose class is appropriate for the size of your local network. Your organization is responsible for issuing and maintaining the host ID part of the address on local systems.



### CAUTION

Do not use any of the example Internet addresses shown in this manual as your host address. Always use addresses in the network range assigned to you.

Because of the growing popularity of the Internet, available addresses are becoming scarce. It may be necessary for you to have a sponsoring organization that is already on the Internet. The sponsor would assign your organization a subnet address within the sponsor's network address.

You can use TCP/IP software without an official Internet address on a network that is not connected to the Internet. However, it is a good precaution to proceed as if you intend to connect to the Internet. These are the numbers and addresses for the InterNIC Registration Services:

User Assistance Service

1-703-742-4777

On-line Mailboxes

HOSTMASTER@RS.INTERNIC.NET

Host, domain, network changes, and updates

ACTION@RS.INTERNIC.NET

Computer operations

MAILSERV@RS.INTERNIC.NET

Automatic mail service

Anonymous FTP

RS.INTERNIC.NET IP address 198.41.0.7

log in as ANONYMOUS, with password GUEST

---

Postal address            Network Solutions  
                                  Attn: InterNIC Registration Services  
                                  505 Huntmar Park Drive  
                                  Herndon, VA 22070

## Specifying Domain Names

You often use an alias to specify a host, not an IP address. The */etc/hosts* file is one method used to translate between names and addresses. The iRMX TCP/IP software does not include a Domain Name Service (DNS) server, which is another method used to translate the names. However, it does include a DNS client. The client contacts any DNS servers running on other hosts on the network and uses their name translation services. This section briefly describes the format of domain names, which is the naming convention generally used for TCP/IP.

The Internet authorities maintain several domains, including:

<i>arpa</i>	used by ARPANET
<i>com</i>	commercial organizations
<i>edu</i>	educational institutions
<i>mil</i>	military groups

Within the major domains, Internet authorities assign subdomains for use by organizations. Local authorities in the organizations then assign machine names and possibly further subdomains.

You specify domain names with dotted notation; *myhost.mydept.mycompany.com* is an example. In this name, *myhost* is the name of the host computer, *mydept* is a subdomain assigned by a company, *mycompany* is a subdomain assigned to that company, and it is in the *com* domain because it is a commercial organization. This is an example of a fully-qualified name, beginning with the host name and ending with the Internet domain. The name *myhost* is qualified by its domain *mydept.mycompany.com*. Each name must be unique within its domain; there cannot be two *mydept* names (of either a host or subdomain) within *mycompany*.

In a local network you need only a host name to communicate between systems. However, to communicate by name with hosts on the Internet, you may want to specify the complete domain names as their official names in your */etc/hosts* file.

## Request For Comment (RFC) Reports

The Internet community uses RFCs to discuss and define TCP/IP. This manual refers to certain RFCs by number for protocol definitions and details. You can get RFCs in electronic format from the InterNIC Information Services.

---

Anonymous FTP      NIS.NSF.NET      IP address 35.1.1.48  
                                 RFC index */internet/documents/rfc/INDEX.rfc*  
NISC.JVNC.NET      IP address 128.121.50.7  
VENERA.ISI.EDU      IP address 128.9.0.32  
WUARCHIVE.WUSTL.EDU  
                                 IP address 128.252.135.4  
NIC.DDN.MIL      IP address 192.112.36.5

On all hosts, log in as ANONYMOUS, with password GUEST or, if requested, your net address.

On-line Mailboxes      rfc-info@ISI.EDU      Send the message  
                                 "help: ways\_to\_get\_rfcs". For example:  
                                 To: rfc-info@ISI.EDU  
                                 Subject: getting rfcs  
                                 help: ways\_to\_get\_rfcs

Info@is.internic.net

□□□

To use TCP/IP software, install it on your system's hard disk along with the iRMX OS, using the general installation instructions for the OS. Edit text files to configure the system for your network, following the TCP/IP software requirements and configuration instructions in this chapter, and then start the network jobs. This chapter provides additional software requirements and configuration instructions for TCP/IP. This chapter does not describe the hardware installation or setup.

See also: *Installation and Startup* for installation instructions

## Before You Begin

During the installation, some new files replace existing files of the same name. The old files are saved in a different directory. If you install over a previous version of TCP/IP software, there may be old versions of configuration files that you want to merge with the new files.



### CAUTION

If you install over a previous version, unload the TCP/IP jobs first. Do not allow the installation to overwrite the files in `/rmx386/jobs` while TCP/IP is running. If this happens, restore the files from the installation disks.

Existing configuration files are preserved during installation, but it is a good precaution to back up your entire hard drive to tape before beginning the installation.

---

## Software Required

Figure 2-1 on page 12 shows the relationship between TCP/IP software and the iNA 960 software. The two separate stacks are the two sets of network protocols that can operate simultaneously when you run iNA 960 software, an iRMX network job, and TCP/IP software. In the middle of the figure, note that the EDL NIC driver provides the direct interface between iNA 960 software and the TCP/IP NIC driver. Although the figure shows multiple NICs, in most systems you only use one.

See also: *Network User's Guide and Reference* for more information about the layers and multiple subnets in iNA 960 software

See also: *Configuring and Administering Network Files*, Chapter 6

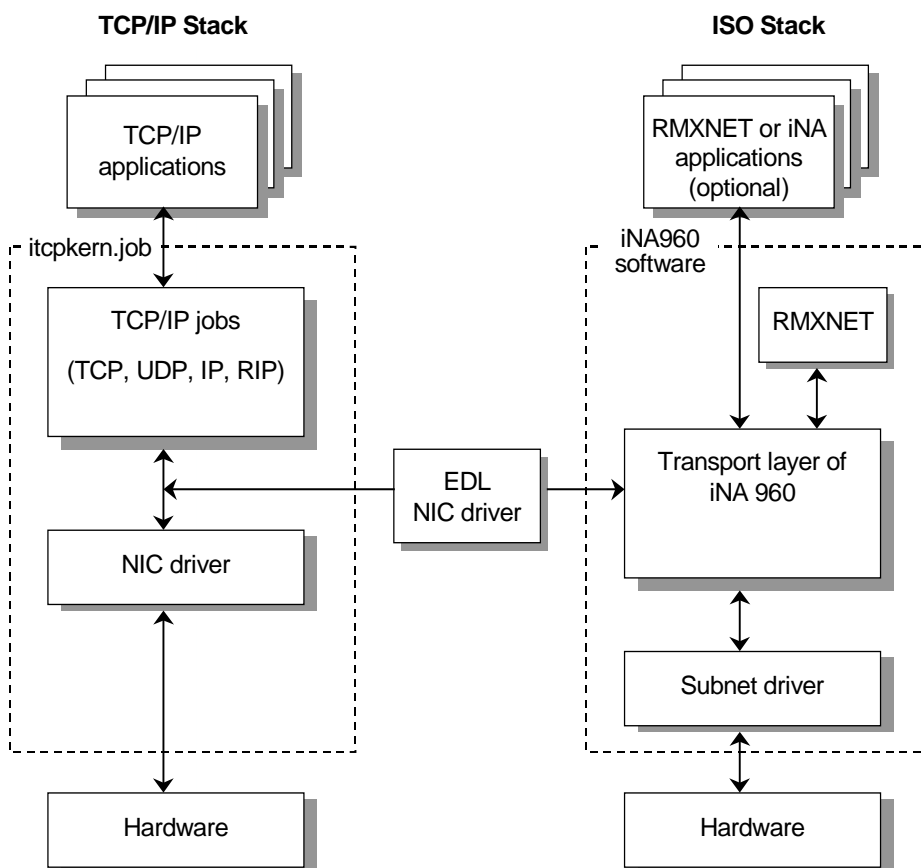


Figure 2-1. How TCP/IP Works with iNA 960 Software

---

## Hardware Required

TCP/IP can run on any system supported by the required iRMX software.

The NIC must be one supported by the NIC driver software.

See also: Tunable Parameters, Chapter 8;  
*i\*.job* and *clib.job*, *System Configuration and Administration*;  
Hardware Environments, *Network User's Guide and Reference*

## Overview of the Setup

To begin using the TCP/IP software:

1. Install the iRMX OS software.
2. Configure the TCP/IP software by editing the *tcp.ini* configuration file.
3. Load the TCP/IP jobs with the **sysload** command.
4. For servers, optionally start the daemons required to support TCP/IP commands: **ftpd** and **telnetd**.
5. If users will run **telnet** from a PC console to a UNIX host, set up the remote UNIX host to support the RMXPC terminal type.

## TCP/IP Configuration

TCP/IP can run as a job loaded with the **sysload** command. Configuring TCP/IP involves editing one or more of these ASCII text files:

- Hosts file
- tcpstart.csd
- 

The purpose of each file is explained in more detail later in this manual, but the instructions here will get you started using TCP/IP.

The files are installed in :CONFIG: directory. Edit the files while logged in as the Super user. On a multiuser machine, access to these files should be restricted to a network administrator.

The network administrator for your organization should assign the name and address values described here. If you are the network administrator, you must assign and keep track of unique values within the network and domain given you by the InterNIC Registration Services.

See also: Understanding Internet Addresses, Chapter 1

---

## Editing the Hosts File

For any TCP/IP communications you can specify an IP address for a remote host or obtain the address from one of two places: the `:CONFIG:hosts` database or the Domain Name Service (DNS).

⇒ **Note**

The iRMX TCP/IP software does not include **named**, the DNS server. However, it does include a DNS client. Another system running an OS like Unix must provide the DNS server.

The client contacts any DNS name server running on the network and uses its name translation services to get the IP address. Regardless of whether you use DNS or not, you must edit the `:CONFIG:hosts` file.

See also: **gethostent**, Chapter 11 for more information on DNS;  
`:CONFIG:hosts`, Chapter 9

## Using DNS

TCP/IP applications may use a DNS client to get an IP address associated with a name from the DNS server on the network. If you choose to use the DNS server, you need to specify only the local host name in the `:config:hosts` file. You also need to edit the DNS section of the tunable parameters file to configure the DNS client.

See also: Tunable Parameters, Chapter 8;  
**gethostent**, Chapter 11

## Not Using DNS

If you don't use DNS, add one line to `:config:hosts` for each system on your network, including the local host. Each line must have at least these two entries:

```
IP_address official_name
```

Specify the official name of the host machine, using a fully-qualified domain name if you have one. You can add alias names on the same line after the official name.

## Configuring TCP/IP as a Loadable Job

To configure TCP/IP as loadable jobs (loaded by the **sysload** command), you need check the contents of the `:config:tcpstart.csd` and `:config:inetinit.cf` files.

---

## Editing the `tcpstart.csd` File

For TCP/IP jobs loaded with the **sysload** command, edit the `:config:tcpstart.csd` file. This file is an esubmit file that sets values and starts jobs needed to run TCP/IP software. The next few subsections describe the changes you need to make in `:config:tcpstart.csd`.

## Configuring the Interfaces

To configure the interfaces you use, edit the `:config:tcp.ini` file. You must change the address and mask values to be appropriate for your host and network.

See also: Tunable Parameters, Chapter 9

## Starting and Stopping TCP/IP

To load TCP/IP with the **sysload** command, you need to submit a file to start the jobs. (This assumes that you have already installed a NIC.) You can submit the file yourself at the iRMX prompt while logged in as Super. Or, you can place the commands in startup files so that the file is submitted automatically every time you boot the system.

The submit command is:

```
esubmit :config:tcpstart
```

In addition to the entries described in the earlier Configuration section, the `tcpstart.csd` file also starts the TCP/IP kernel as a set of loadable jobs. By default, this command appears as a comment in the `:config:r?init2` startup file. To automatically submit the file every time the system boots, remove the semicolon character at the start of the line.

### ⇒ Note

Do not place commands that prompt for keyboard input in any of the configuration files `:config:loadinfo`, `:config:r?init`, or `:config:r?init2`. Running commands from the `:config:r?init2` file can make booting a little slower.

To stop TCP/IP without shutting down the system, you can submit the `:config:tcpstop` file. Before unloading TCP/IP, make sure that there are no TCP/IP servers or clients running.

```
submit :config:tcpstop
```

The `:config:tcpstop.csd` file provides an orderly shutdown of the TCP/IP jobs and applications. It unloads `tcp.job`, `udp.job`, `rip.job`, `ip.job`, and NIC driver jobs.

---

## Testing the TCP/IP Setup

Test the TCP/IP software and its connection to the network by issuing this command:

```
ping loopback 56 3
```

This command sends packets on the network to the local machine. It tests both TCP/IP and the network hardware; TCP/IP must be able to send and receive packets to display a message similar to this:

```
PING loopback: 56 data bytes
64 bytes from IP_address: icmp_seq=0. time=0 100th of sec
64 bytes from IP_address: icmp_seq=1. time=0 100th of sec
64 bytes from IP_address: icmp_seq=2. time=0 100th of sec
----loopback PING Statistics----
3 packets transmitted, 3 packets received, 0% packet loss
round-trip (100th of sec)   min/avg/max = 0/0/0
```

Notice the next-to-last line, indicating that all the packets sent were received.

If this command succeeds, test the connection to other hosts on the network. Repeat the **ping** command, specifying the remote host's name or address instead of `loopback`. To use names, you must configure the name-to-address translation in the `:config:hosts` file or from the DNS server as described earlier.

If you enter the **ping** command without the numeric values, it continues sending packets until you interrupt it with a <Ctrl-C>.

See also: Network Tests, Chapter 9, for other tests you can perform  
**ping** command, *Command Reference*

## Troubleshooting

Problems can occur at several different levels. For example, TCP/IP may have failed to install correctly. This in turn causes jobs dependent on TCP/IP to not load correctly. This section provides some general troubleshooting guidelines and explains some specific error conditions.

## General TCP/IP Debugging

Follow these ordered steps to try and isolate TCP/IP problems:

1. Try to execute some of these commands:

```
netstat -i
netstat -a
```

If you get errors then perform steps a through c.

---

If these commands execute correctly, you can assume that TCP/IP is loaded and running. Steps a through c do not apply.

- a. Check the messages in `:config:r?init2.log`, the log file of the `:config:r?init2` file. Be sure that `r?init2` submitted the file `:config:tcpstart.csd` and that all the commands in the submit file ran properly.
  - b. Check the `hostid` and `hostname` settings in `:config:tcp.ini` to be sure they are correct.
  - c. Check the `:config:hosts` file to be sure your `hostname` is there with the correct IP address.
2. If you are having trouble with **telnet** try the following:
    - a. Check the `ptydrvnn.log`, where `nn` is the slot number of the client board.
    - b. Enter **initstatus** and see if any `ttyp_*` are available and not locked.

## Setting Up a Remote Unix Host for Telnet

Once the **ping** command succeeds, TCP/IP is set up and ready to support file transfer via the File Transfer Protocol (FTP) or the Trivial File Transfer Protocol (TFTP). Before you can begin remote login through the **telnet** command, however, you may need to do additional setup on the remote Unix host.

## Creating a Terminal Definition for the PC Console

To run any Unix program that supports cursor movement (any program using the `curses` library, such as the `vi` editor) you must set a `TERM` environment variable that matches your iRMX terminal. If you make a connection through the **telnet** or **rlogin** commands from any standard terminal, the Unix host should already have a matching terminal type definition. However, to use **telnet** or **rlogin** from a PC console (`:d_cons:`) you need to define a new terminal name, `RMXPC`. This procedure modifies system files on the remote Unix host, which requires root privileges. If necessary, contact your Unix system administrator for assistance.

To set up the `RMXPC` terminal definition:

1. Copy `:config:termcap.rmx` and `:config:terminfo.rmx` from the iRMX host to `/etc` on the Unix host. Use FTP or some other method.
2. Log into the Unix host as root.
3. Edit the existing `termcap` file to add the contents of `termcap.rmx`. (If you are sure that users will be running only applications that use `terminfo`, instead of `termcap`, you can skip this step. But if there is any doubt, perform this step.)

- 
4. Run this command:

```
tic terminfo.rmx
```

If your system does not have a **tic** command, skip this step.

## Setting Terminal Characteristics for User Sessions

When iRMX users remotely log into a Unix host, the **telnet** or **rlogin** commands change the Unix terminal type to the name of the iRMX terminal. If the Unix host is set up to support that terminal type, and the Unix account does not reset the terminal type, nothing more is necessary.

However, Unix accounts that are also used for local logins need to set the terminal type during initialization. This overwrites the **telnet** or **rlogin** terminal setting. Because the remote iRMX terminals and local Unix terminals are likely to be different, the best way to handle this is to prompt for the terminal type.

Use this procedure to set up Unix user accounts for users who use the **telnet** or **rlogin** commands. You may need to experiment; the specifics vary for different shells and versions of Unix. If you need help, ask your Unix system administrator.

1. Check that the Unix host is set up to support the required terminal types and that the terminal names are exactly the same on the Unix and iRMX OS. The possible names include:
  - Standard terminals, like *wyse50*
  - RMXPC for the PC console, as discussed earlier
2. Edit the initialization file in each user's home directory. For those who use the Bourne shell, *bash*, or Korn shell, the file should be *\$HOME/.profile*, and for C shell users, *\$HOME/.login*.
  - Set up handling of the terminal type. If the account will only be used for logins via **telnet** or **rlogin**, comment out any reference to terminal type, such as:

```
setenv TERM
TERM =
export TERM
```

Or, if the account will be used for both remote and local logins, set up a prompt for the terminal type. This simple example for the Bourne shell *.profile* produces a prompt:

```
echo "TERM=\c"
read TERM
```

- 
- Define the interrupt sequence, erase sequence, and tab settings for the potential terminal types. This Bourne shell example for the RMXPC terminal sets the interrupt to <Ctrl-C> and erase to <Del>, and sets tab expansion.

```
if[$TERM = "RMXPC"]
then
    stty intr ^C erase ^? -tabs
fi
```

See also: Documentation for your Unix system





With TELNET you can log in to a remote host as if your terminal were directly cabled to it. TELNET provides reliable, virtual terminal communication with any network host that supports the TCP standard, regardless of the host's OS. The remote host must implement a TELNET server.

## Before You Begin

Before you begin a TELNET session on a remote host, you must know:

- A user login name and password on the remote host
- One of the valid names for the remote host: its Internet address, its official host name, or its alias

You can get valid host names and addresses from your `:config:hosts` file.

The remote host must have a TELNET server process, **telnetd**, and be listening for TELNET requests. If you need additional information or help setting up a remote host login, see your network administrator.

## Telnet Modes

TELNET operates in two modes: input mode and command mode. In input mode, you log in and enter OS commands, which are processed by the OS on the remote host. In command mode, you enter TELNET commands, which are processed by the TELNET program on the local host.

You can start the TELNET program in either mode and switch between modes during a TELNET session.

---

Figure 3-1 shows how commands are processed in input mode and in command mode.

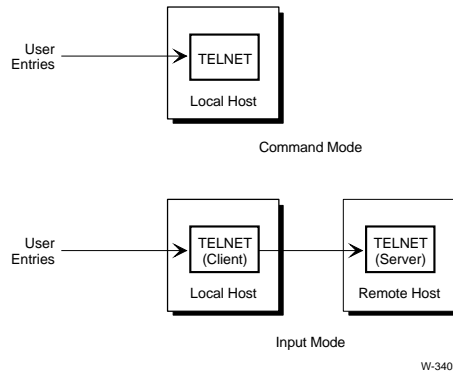


Figure 3-1. TELNET Modes

## Starting TELNET

You can use this command to start TELNET and connect to any other remote host:

- **telnet**

### Starting in Input Mode

To start TELNET in input mode enter `telnet hostname` at the iRMX prompt, specifying the name of the remote host to which you want to connect. If TELNET connects to the host, you are prompted to log in. After you log in, any commands you enter are processed by the remote host. The input mode prompt is the remote host's OS prompt. When you exit the remote session, the TELNET program terminates, and you are returned to the OS prompt on the local host.

### Starting in Command Mode

To start TELNET in command mode, enter **telnet** at the iRMX prompt. The TELNET program starts and displays the command mode prompt, `telnet>`. It does not attempt to connect you to a remote host; in command mode you enter TELNET commands that are processed by the local host. From the `telnet>` prompt, you can use the **open** command to connect to a remote host in input mode. If you open a remote session in this way, you will be returned to command mode when you close the session.

---

## Switching Telnet Modes

To switch from input mode to command mode, enter the current TELNET escape character, followed by a carriage-return. The default escape character is ^] (control J). You can change the escape character with the TELNET **escape** command. The `telnet>` prompt confirms that you have entered command mode. You can specify several options on the **telnet** command line.

To switch back from command mode to input mode, enter a <CR> at the command mode prompt. At this point you can resume what you were doing before you entered command mode.

## Using TELNET for a Remote Session

When you use TELNET for a remote session, you establish a virtual terminal connection to the remote host. The remote host gives you the same privileges and capabilities as it does for users with terminals directly cabled to it. While you are working on the remote host, your session with the local host is maintained.

The procedure for conducting a remote TELNET session consists of three general steps:

1. Connecting to the remote host
2. Entering commands during the session
3. Closing the remote connection

---

## Connecting to the Remote Host

You can begin a remote session at the iRMX system prompt or at the TELNET command mode prompt. In either case, the TELNET client process in your local host activates a TELNET server process in the remote host to service your session.

You specify a remote host by its Internet address, its official name, or an alias name. To connect to a remote host named *host2* at Internet address 128.215.12.21, you could use either of the command methods shown below to open the connection. You could use either form of the name in either command:

### From the iRMX Prompt

```
- telnet 128.215.12.21  
- telnet
```

### From the TELNET Prompt

```
telnet> open host2
```

If the attempt succeeds, your screen displays a connection message and the remote host login prompt. The connection message includes information about the TELNET session, including the current escape character. The output from the above commands is similar to this:

```
Trying 128.215.12.21 ...  
Connected to 128.215.12.21.  
Character mode is enabled.  
Escape character is ^].  
UNIX System V Release 3.2 (host2.intel.com)
```

```
login:
```

If all pty devices are in use and a TELNET request comes in, **telnetd** will send the following error back to the client:

```
No pty devices available at this time.
```

Regardless of the reason, if the connection attempt does not succeed, you are returned to the `telnet>` prompt and are requested to log in. This cycle repeats until you successfully log in or until you close the TELNET session with the `^]quit` command, where `^]` is the current escape character.

---

## Setting the Terminal Type on a Unix System

When you log in to a Unix host, TELNET changes the Unix terminal type to the name of your iRMX terminal. If the Unix host is set up to support that terminal type, and your Unix account does not reset the terminal type, you do not need to do anything more.

Some Unix accounts, however, reset the terminal type during initialization. This overwrites TELNET's terminal setting. If a terminal prompt appears when you log in, respond with the name of your iRMX terminal. For example, if you are working at a Wyse 50 terminal, specify:

```
TERM = wyse50
```

If you are using the PC console as your iRMX terminal, the terminal type is `RMXPC`. For ICU-configurable iRMX OS with Multibus II and the iSBX 279 Graphics Module, the terminal type is `i279`. If the system does not recognize your terminal type, see your Unix system administrator.

Sometimes the initialization file automatically sets a predetermined terminal type. TELNET will not work properly with this setup, unless the terminal type happens to be the same as your iRMX terminal. If you suspect this has happened, check your current terminal type:

```
echo $TERM
```

If you need to reset the terminal type, it is best to do it in your initialization file. Otherwise, the problem will happen again every time you log in. If you need help, ask your Unix system administrator.

See also: [Setting Up a Remote Unix Host for Telnet and Rlogin, Chapter 2](#), for initialization file setup

## Terminal Type Strings

iRMX supports terminal type lengths of six characters or less. When a TELNET session begins, the client passes a string representing the terminal type to the server. If a client with a terminal type of more than six characters tries to connect to a **telnetd/rlogind** server running on iRMX, the following warning displays at the client end:

```
Terminal type too long for iRMX, try another
```

---

## Disabling Local Echo on Berkeley Unix Hosts

When you connect to a TELNET server on a Berkeley Unix host, before any other commands, you need to enter the TELNET **localecho** command. This is a toggle that turns local echo off. Use the instructions for entering TELNET commands in this section.

## Entering Commands During the Session

During the remote session, you can enter input mode commands at the remote host's OS prompt or command mode commands at the TELNET prompt.

At the remote OS prompt, enter any command that is appropriate for that environment. The local host will pass your commands to the remote host for processing without interpreting them.

To enter TELNET commands, switch to command mode by entering the escape character, followed by a space. The system displays the `telnet>` prompt. At the prompt, type your command, then press <Enter>. You can enter any of the TELNET commands in this manner. TELNET processes your command, then returns to input mode so you can continue your remote session.

This example uses the escape character `^]` and the TELNET **status** command during a remote host session. The `§` is the remote OS's prompt. Unlike the way it is shown here, the escape character does not appear on your screen when you enter it.

```
§ ^]
```

```
telnet> status
```

```
Connected to host2.intel.com.
```

```
Character mode is enabled.
```

```
Escape character is ^].
```

```
§
```

---

There are several other TELNET commands that let you control options for the TELNET session. Use the TELNET ? command to list all the commands and their descriptions:

```
telnet> ?

Commands may be abbreviated.  Commands are:

close      close current connection
logout     forcibly logout remote user and close the
           connection
display    display operating parameters
mode       try to enter line or character mode ('mode ?' for
           more)
telnet     connect to a site
open       connect to a site
quit       exit telnet
send       transmit special characters ('send ?' for more)
set        set operating parameters ('set ?' for more)
unset      unset operating parameters ('unset ?' for more)
status     print status information
toggle     toggle operating parameters ('toggle ?' for more)
slc        change state of special charaters ('slc ?' for
           more)
!          invoke a subshellenvirom   change environment
variables ('envirom ?' for           more)
?          print help information
```

See also: **telnet** command, *Command Reference*

## Closing the Remote Connection

To close a connection to a remote host, you can:

- Enter the TELNET **quit** command
- Use the remote host's logout procedure
- Enter the TELNET **close** command

The **quit** command releases your remote connection, stops the TELNET client and server processes on both hosts, and returns you to the OS prompt on your local host.

The remote host logout procedure and the **close** command have the same effect as the **quit** command if you connected to the remote host from input mode (the iRMX system prompt). If you connected to the remote host from command mode (the telnet> prompt), you are returned to the telnet> prompt on your local host.

---

## Using Telnet for a Local Session

It is sometimes convenient to use the TELNET program locally without a connection to a remote host. For example, you might want to use TELNET locally to get information about its commands or to set up a new configuration (such as defining a new escape character) before you begin working on a remote host.

Whenever you use TELNET without a connection to a remote host, TELNET is in command mode and the `telnet>` prompt is displayed. You can enter only TELNET commands, not OS commands. To start the TELNET program without a remote host connection, enter the **telnet** command without a *hostname* parameter:

```
- telnet
telnet>
```

## Entering Commands in a Local Session

During a local session you can enter any of the TELNET commands except **close**. This command is valid only when you are connected to a remote host.

The **status** command prints information about the current TELNET session. In this example, it identifies the host's escape character:

```
telnet> status

No connection.
Character mode is enabled.
Escape character is '^]'.
telnet>
```

## Ending the Local Session

To end a local TELNET session, enter the **quit** command at the `telnet>` prompt. The TELNET process ends and you are returned to the iRMX system prompt on your local host.



TCP/IP for the iRMX OS includes implementations of two file transfer protocols: FTP. Each provides a different degree of security, reliability and functionality.

File Transfer Protocol (FTP) is the most powerful file transfer program available among the standard TCP/IP protocols and is, therefore, preferred by many users.

## Before You Begin

Before you begin a file transfer session, you must know:

- A user login name and password on the remote host
- One of the valid names of the remote host: its Internet address, its official name, or its alias

You can get information about valid remote host names from the `:config:hosts` file, which lists the Internet address, official name, and aliases for each host on the network.

If you need additional information or help setting up a remote host login, see your network administrator.

---

## File Transfer Protocol (FTP)

FTP lets you transfer accessible files between your local host and a remote host that supports TCP/IP. You don't need to know the remote host's OS to transfer files. FTP is implemented entirely as a command line interpreter, where the commands are processed by the FTP client process on the local host.

During an FTP session, you enter commands to the FTP process to control the file transfer and manage the files and directories on the remote host. For example, you can issue FTP commands to open and close a remote host connection, delete remote files, or create new directories on the remote host.

Some FTP commands, such as **bell**, **debug**, and **help**, are processed completely by the FTP client process on the local host. These commands can be executed with or without an established connection to an FTP server process on a remote host. However, most FTP commands require a connection. These commands are translated by the FTP client process into one or more FTP protocol commands, which the client sends to the FTP server process on the remote host for processing. The FTP server, called **ftpd**, is described later in this manual.

As with TELNET, you can start FTP without making a connection to the remote host, using this command at the iRMX prompt:

```
ftp
```

or you can start FTP and open the remote connection with the command

```
ftp hostname
```

In either case the FTP client process starts and displays its prompt, `ftp>`. You can now enter FTP commands as described in these sections.

See also: **ftp** command, *Command Reference*

### FTP Help Information

For on-line information about FTP commands, enter **?** to list all the commands and their descriptions. Use **? *command\_name*** for a description of a single command.

See also: **ftp**, *Command Reference*, or *Quick Reference to TELNET and FTP* for descriptions of all FTP commands

---

## FTP File Transfer Session

An FTP file transfer session consists of three general steps:

1. Connecting to the remote host
2. Using FTP commands
3. Ending the FTP session

### Connecting to the Remote Host

In most cases, you begin a file transfer session by entering a command to establish a connection to a particular remote host. Upon receipt of your command, the FTP client process on your local host activates an FTP server process on the remote host to service the session. If you did not invoke FTP with a *hostname* parameter, you establish a connection with the **open** command at the `ftp>` prompt.

Specify *hostname* as the Internet address, official name, or alias of the remote host. To connect to a remote host named *host2* at Internet address 128.215.12.21, you could use either name in either of the command methods shown below:

#### From the iRMX Prompt

```
- ftp 128.215.12.21
```

```
- ftp
```

#### From the FTP Prompt

```
ftp> open host2
```

FTP attempts to connect you to the specified remote host. If the connection is established, FTP prompts you to log in. The message is similar to this:

```
Connected to host2.intel.com.  
220 host2.intel.com FTP server (Version 1.2 May 02 1992)  
ready.  
Name (host2.intel.com:acct):
```

If the connection cannot be established, you are returned to the `ftp>` prompt.

When a connection is established, FTP prompts you to begin the remote host's login procedure. You must use a valid login name and password to gain access to the remote host. If you need help with logging in, see your system administrator.

When the login is successfully completed, FTP again displays the `ftp>` prompt. You can begin entering file transfer commands.

---

If the login is not successful, FTP displays a message to that effect and returns you to the `ftp>` prompt. At this point you are still connected to the remote host. To log in, enter:

```
ftp> user name
```

where *name* is your user name on the remote host. You are then prompted for your password.

You can automate the FTP login procedure to make it more convenient with a *netrc* file.

See also: FTP Initialization File, in this chapter

## Using FTP Commands

Two commands commonly used for file transfer, the **put** and **get** commands, are described here. Several other FTP commands can be used to manage files and directories on both the local and remote hosts during a session. For example:

- The commands **dir**, **ls**, and **mls** provide you with listings of the files and directories on the remote host.
- The commands **lcd** and **cd** enable you to change directories on the local and remote hosts, respectively.
- The commands **mkdir** and **rmdir** enable you to create or delete directories on the remote host.

See also: **ftp** command, *Command Reference*, for descriptions of these FTP commands

### Put Command

To copy a file from your local host to a remote host, enter this at the `ftp>` prompt:

```
put localfile [remotefile]
```

where *localfile* is the name of the local file to transfer and *remotefile* is the name for the remote copy of the file. If you do not enter a remote filename, FTP gives it the same name as the local copy.

You can use the **send** command as an alias for **put**.

The next example shows how FTP prompts for local and remote filenames when you enter **put** with no filename parameters. It also shows the message FTP displays when the transfer is successful.

---

⇒ **Note**

The **verbose** option must be on, as it is by default, to produce the display shown in this interactive session.

```
ftp> put
(local-file) payroll.1
(remote-file) payroll.2
200 PORT command okay.
150 Opening ASCII mode data connection for payroll.2.
226 Transfer complete.
2103 bytes sent in 0.29 seconds (6.9 Kbytes/s)
ftp>
```

The **put** command transfers one file per transaction. To transfer more than one file in a single transaction, use the **mput** command.

### Get Command

To copy a file from the remote host to your local host, enter this at the `ftp>` prompt:

```
get remotefile [localfile]
```

where *remotefile* is the name of the remote file to be transferred and *localfile* is the name for the local copy of the file. If you do not enter a local filename, FTP gives it the same name as the remote copy.

You can use the **recv** command as an alias for **get**.

The next example shows how FTP prompts for remote and local filenames when you enter **get** with no filename parameters, and the message FTP displays when the transfer is successful.

```
ftp> get
(local-file) personnel.1
(remote-file) personnel.2
200 PORT command okay.
150 Opening ASCII mode data connection for personnel.1 (5909
bytes).
226 Transfer complete.
6123 bytes received in 1 seconds (5.979 Kbytes/s)
ftp>
```

The **get** command transfers one file per transaction. To transfer more than one file in a single transaction, use the **mget** command.

---

## Transferring Files Between Systems With Different File Naming Conventions

When you transfer files between hosts with different operating systems, be sure to specify a name for the new file that conforms to the local file naming conventions. If you do not specify a destination name on the command line, FTP attempts to use the source name. If that name is not valid on the local host, the command fails. For example, you may need to copy a Unix tar file to the DOS file system on iRMX for PCs. Use a command line like one of these:

```
ftp> get bash.tar.Z bash_t.Z
ftp> put bash.tar.Z bash_t.Z
```

The destination file name, *bash\_t.Z*, conforms to the DOS 8.3 file name convention, so it can be used with the DOS file drivers.

## Transferring Large Files

To transfer large files to a remote Unix host using FTP, you might need to increase the value of the system parameter `ulimit` on the remote host. `Ulimit` is a Unix System V security feature that enables the network administrator to limit the size of files that can be created by local users. The default limit on many systems is 2048 512-byte blocks, or 1 MB. File transfer applications such as FTP and TFTP and `rcp` must obey the file size limitations imposed by the system on which the file is to be created. The default value of `ulimit` for the remote host governs the maximum size of a file that can be sent.

FTP allows you to change `ulimit` on a remote Unix host, but you must have root privileges on the Unix host to increase the value. Users without root privileges can only check the value or decrease it. If you do decrease the `ulimit` in a remote session, you cannot increase it, even to its original value, unless you have root privileges. If you need to increase `ulimit` on a host on which you do not have root privileges, contact your network administrator for assistance.

First you need to establish an FTP connection with the remote host, logging in as root. Then change the `ulimit` value for the remote session, using the FTP `site` command as follows:

```
ftp> site ulimit 16384
200 ULIMIT set to 16384 blocks
ftp> put big1
[File transfer information]
.
.
.
ftp>
```

---

There are other remote commands you can execute with **site**, depending on the commands made available by the remote FTP server, **ftpd**.

See also: **ftp** and **ftpd** commands, *Command Reference*

## Ending the FTP Session

To end a file transfer session, enter one of these commands:

- **bye**, or its alias **quit**
- **close**

The **bye** and **quit** commands release your connection, stop the FTP client and server processes in the local and remote hosts, and return you to the iRMX OS prompt on your local host.

The **close** command releases your connection to the remote host and returns you to the **ftp>** prompt on your local host.

## FTP Initialization File

If you set up an FTP initialization file, the FTP process will log you on to a remote host automatically. Name the file *netrc* or *r?netrc* and put it in your home directory on the local host. If the FTP process finds *:home:netrc* at startup, it reads the file to obtain the information it needs to complete remote host login procedures.

### ⇒ Note

For those familiar with FTP in a Unix environment, on iRMX this file is named *netrc* without a beginning . (period or dot) in the filename. To hide the file on an iRMX system, name it *r?netrc*. When any program refers to *netrc*, the iRMX OS automatically maps it to *r?netrc*.

To create *netrc*, build a file that contains this information about each remote host where you want to log in automatically:

- The official host name as set with the **hostname** command; an Internet address or alias is not acceptable
- Your user login name on the remote host
- Optional: the password to your login on the remote host

Each line of the *netrc* file describes a different host. There is no limit to the number of lines the file can contain. The format for each line is:

```
machine host login login-name [password password]
```

---

The keywords `machine` and `login` must appear in each line, followed by the official host name and your remote user login name, respectively. Each word on the line must be separated from other words by a space or tab.

The keyword `password` and your password are optional. If you do not enter password information for a remote host in the *netrc* file, FTP prompts you for it when you log in to the host. Because the *netrc* file might contain password information, make the file readable only by the owner. FTP for the iRMX OS, unlike other versions, does not enforce owner-only file access. FTP does print a warning if the *netrc* file contains account information or passwords.

Below is an example of a record in a *netrc* file. In this example, `tvi386` is the official name of the remote host and `nancy` is the login name on that host. Because the password is omitted, Nancy will be prompted for it during login.

```
machine tvi386 login nancy
```

See also: *netrc* file, Chapter 10



As network administrator, you determine which services each host on the network will provide. Many network services involve the interaction of a client process on one host and a server process on another. By defining the server processes that run on a particular host, you control the types of access available to remote clients.

An example of this type of network service is FTP, which is implemented by a client process (**ftp**) and a server process (**ftpd**). In general, the client and server share the same root name, and the server name includes the suffix *d*, which designates it as a *daemon*. A daemon operates in the background. A server daemon operates when it receives a client request. Virtually all of the networking commands available to the general user invoke the client process of a client/server pair.

Several additional network services are implemented by network daemons that are not associated with client processes. These daemons exchange messages with their counterparts on remote hosts and update local kernel tables or network databases based upon the information received. By defining the daemon processes that will be running on a particular host, you control the automatic (by daemon) or manual (by administrative command) updating of the related network tables.

These sections describe the network services that you can control for each host. Each section contains a brief description of the service, some guidelines for determining whether or not the service should be enabled, and instructions for configuring, enabling, or disabling the service, where applicable. The servers and daemons are described in alphabetical order.

See also:    TCP/IP daemons, Chapter 2;  
              Stopping and Restarting TCP/IP, Chapter 2

---

## Ftpd Server

**Ftpd.job** is the server process for the File Transfer Protocol (FTP). The client process is the **ftp** command. Running **ftpd.job** on the local machine allows remote **ftp** users to connect to this host to transfer files.

To enable FTPD on the local host, edit the startup script `:config:tcpstart.csd` and uncomment the line which sysloads the `ftpd.job`. If you are starting the TCP/IP stack from the `:config:loadinfo` file, uncomment the line which sysloads the `ftpd.job` in this file. Because FTP is one of the basic networking services provided by the TCP/IP package, it is very unusual to encounter a network host that is not listening for FTP requests.

If the local host is currently providing FTP access, the display from a **netstat -a** command includes an entry with a local address of `*.ftp`.

See also: **ftpd.job**, *System Configuration and Administration*

---

## Telnetd Server

**Telnetd.job** is the server process for the TELNET protocol, which defines the network virtual terminal access to a remote host. The client process is the **telnet** command.

To enable TELNETD on the local host, edit the startup script `:config:tcpstart.csd` and uncomment the line which sysloads the `telnetd.job`. If you are starting the TCP/IP stack from the `:config:loadinfo` file, uncomment the line which sysloads the `telnetd.job` in this file. Because TELNET is one of the basic networking services provided by the TCP/IP package, it is very unusual to encounter a network host that is not listening for TELNET requests.

If the local host is currently providing the TELNET service, the display from a **netstat -a** command includes an entry with a local address of `*.telnet`.

## Configuring Pseudo-terminals for Telnetd

The `telnetd` server node needs some additional configuration to set up pseudo-terminals for the remote client TELNET sessions to access. Like terminals, pseudo-terminals need to be identified and enabled in the `:config:terminals` file. Then the number of supported pseudo-terminals needs to be specified as a parameter to `telnetd.job` as follows:

1. Add an entry to the `:config:terminal(s)` file to initialize each iRMX pseudo-terminal device for users. List these devices as `pty_0`, `pty_1`, up through `pty_n-1` where `n` is the number of pseudo terminals supported. `N` can vary from 1 to 16 inclusively. For example:

```
pty_0,,,any
```

Also edit the first line of the file, increasing the number by one for each new entry added. If that number is smaller than the number of entries, the extra entries are ignored

For example, `pty_2` in the following file cannot be used because the 3 at the beginning means the Human Interface initializes only the first three terminals.

```
3
d_cons,,,pc
pty_0,,,any
pty_1,,,any
pty_2,,,any
```

2. Update the `:config:tcpstart.csd` and/or the `:config:loadinfo` files to uncomment the line which sysloads the `/rmx386/jobs/telnetd.job` service and specify the

---

number of pseudo-terminals to be supported. The sysload command has the following form:

```
Sysload /rmx386/jobs/telnet.job num_pttys=n
```

Where

n is the number of pseudo-terminals to be supported. This number can vary between 1 and 16 inclusively. If num\_pttys is not specified, the telnetd service assumes 4 pseudo-terminals.

See also: *Configuring terminals, System Configuration and Administration*;  
**telnetd.job**, *System Configuration and Administration*

□ □ □

# Configuring and Administering Network Files 6

---

As network administrator, you define the operation of several network daemons and servers by setting up their configuration files. The network configuration files are described in this chapter.

<b>Network Configuration File</b>	<b>Network Daemon or Job</b>
<i>:config:tcp.ini</i>	TCP/IP jobs

## Restricting and Updating Network Databases and Files

The following list shows files that maintain information about hosts, networks, protocols, and available network services. Some of these files enable remote user access. As network administrator, you should ensure that these files are updated whenever the topology of the network changes. Only the network administrator should have permission to modify these files.

<b>File</b>	<b>Purpose</b>
<i>:config:hosts</i>	Lists addresses and names of accessible hosts and interfaces on the net
<i>:config:services</i>	Lists names, port numbers, and protocols associated with available services
<i>:home:netrc</i>	User-specific file that provides login information to FTP servers

See also: Chapter 9 for details about the contents of each file





There are several TCP/IP commands that display configuration information and perform network maintenance. The network administrator uses these commands to monitor the overall status of the network, monitor and make available remote resources, test specific interfaces or functions, and configure certain interface characteristics. This chapter describes the purpose for using such commands.

See also: Command syntax and descriptions, *Command Reference*

## Administrative Commands

These are the network maintenance commands:

- netstat** Displays information from network data structures so you can identify network problems. This chapter describes network tests you can perform with this command.
- ping** Tests low-level communications between two hosts to determine if there is a fault between them.

## Performing Network Tests

As network administrator, you perform tests to determine whether the network services and daemons are running as expected, whether the interfaces and routes have been correctly configured, and whether each interface is functioning properly.

You should run a comprehensive set of tests after the network is first installed. These tests should include the functional tests of the software loopback interface as well as the basic assessment of the network configuration. At subsequent times when the network is brought up, you should run a subset of the initial tests to determine, at a minimum, that the correct daemons and interfaces are available. You should also thoroughly test each network interface when it is initially configured.

The tests described here are only suggestions. Your own networking environment will determine the tests that you select as most useful.

See also: **netstat** command, *Command Reference*, for more information about test results

---

## Verifying Network Services

When the network is first brought up, you can perform the Network Status Test to verify that the network startup script *tcpstart.csd* has been properly configured.

### Network Status Test

For the Network Status Test, perform these steps:

1. Use the **netstat -a** command to display all the active network connections and listening servers.
2. Verify that there is an entry in the **netstat -a** table for every network server daemon you have configured.

See also: Chapter 5 for definitions of network servers

- 
3. For TCP-based services, verify that the entries in the **netstat -a** table have these attributes:
    - The protocol is tcp.
    - The address part of the local address is wild-carded.
    - The port part of the local address shows the service name as defined in the *:config:services* file.
    - Both the address and port parts of the foreign address are wild-carded.
    - The state is LISTEN.
  4. For UDP-based services, verify that entries in the **netstat -a** table have these attributes:
    - The protocol is udp.
    - The address part of the local address is wild-carded.
    - The port part of the local address shows the service name as defined in the *:config:services* file.
    - The address part of the foreign address is a name, address, or wildcard.
    - The port part of the foreign address is wild-carded.
    - The state is empty.

## Verifying Network Configuration

You can perform the Interface Status Test, the Interface Configuration Test, and the Route Configuration Test to verify network configuration.

### Interface Status Test

For the Interface Status Test, perform these steps:

1. Use the **netstat -i** command to display the configured network interfaces.
2. Compare the **netstat -i** display with the contents of the network configuration file *tcp.ini* to verify that all interfaces have been successfully configured.

- 
3. Ensure these conditions are true for each entry in the **netstat -i** table:
    - The interface name is the same as the one defined in the *tcp.ini* file. This name is unique.
    - The maximum transfer unit (MTU) for each interface is a positive nonzero integer that reflects the type of communications medium used: 4096 for the software loopback interface and 1500 for Ethernet interfaces. If the MTU is zero, the interface did not initialize properly.
    - The network and address fields each contain a name, not an Internet address. The address field contains the host name assigned to the interface in the *tcp.ini* file. The network field contains the network name from the *:config:networks* file that matches the network portion of the address associated with that host name in the *:config:hosts* file. (If the network address is displayed, make sure the *:config:networks* file has an entry for the address also.)
    - The input and output error fields are 0. The input packets field is at least 2. The output packets field is 0 or a positive integer.

## Interface Configuration Test

CHRIS?

---

## Verifying Interface Functionality

The purpose of this type of network testing is to verify that each configured interface is functioning properly and that all three of the Transport Layer protocols (tcp, udp, and raw) are working as expected. Test the software loopback interface first as described below to determine that the basic streams have been properly constructed. Then test each network interface in the same manner.

1. To test the tcp transport layer, perform these steps:

- Enter:

```
telnet me
```

In response to your command, a DNS database or the `:config:hosts` file is accessed to obtain the Internet address for the host `me`, at which point TELNET displays this message:

```
Trying 127.0.0.1 .....
```

If this message is not displayed, check the `:config:hosts` file to make sure that the proper name-to-address translation is available.

As soon as TELNET makes the connection, it displays the connection status and then the login banner received from the remote host (in this case, the local host through the loopback connection).

- Log in and then log off to terminate the test.

The three errors most often encountered when running this test are:

- No address translation can be found for the remote host name (unknown host).
- The remote host is not listening for TELNET connections (connection refused).
- The remote host did not respond to the connection request (connection timed out).

The last error can be caused by a hardware problem. It can also occur if the remote host is down, does not have the network running, or is very busy.

2. To test the udp transport layer, use the command:

```
tftp me
```

3. To test the raw transport layer, use the command:

---

```
ping me 1 10
```

This sends ten one-byte ECHO\_REQUEST packets to the local host, using the loopback device. The transmission summary should show no packet loss and reasonably consistent round trip times for the individual packets.

You can use the **netstat** command to test the functionality of the udp and tcp transport layers.

See also: **telnet**, **netstat**, and **ping** commands, *Command Reference*

□□□

A number of tunable parameters affect the functionality and performance of TCP/IP software. For each TCP/IP job, there are parameters that define how that job operates.

Tuning is a tradeoff between allocating enough resources to facilitate networking operations and keeping the kernel small enough to be manageable. The recommendations made in this chapter are generally on the small end of the scale. You will almost certainly need to revise them to meet the needs of your network's configuration. Start with the values specified and monitor the system closely for a while to determine what your environment really needs.

## Determining When to Tune Parameters

The TCP/IP kernel is installed with default parameters that are adequate for a simple host configuration, with one network interface and a moderate amount of network traffic. After you determine your host and network configuration, you should review the TCP/IP parameters listed in this chapter and reset them as needed.

## TCP/IP Parameters

Parameters in the `:config:tcp.ini` file affect the TCP/IP jobs' operation, and performance.

⇒ **Note**

Values *not* enclosed in single quotes are hexadecimal numbers.

### [TCP]

File	Default Value	Description
DEFMSS	200	Default maximum segment size
RCVSPACE	4000	Maximum receive space per socket
SNDSpace	4000	Max send space per socket
CTLBUFS	40	Maximum total control buffers
TRANSBUFS	40	Maximum total transaction buffers
MAXTRANS	10	Maximum simultaneous IP transactions
MAXPORTS	1388	Maximum port ids
LOWFIXPID	1	Well-known port id range
HIFIXPID	3FF	
LOWAUTOPID	400	Ephemeral port id range
HIAUTOPID	1387	

### [UDP]

File	Default Value	Description
CHECKSUM	1	Enable checksum
RCVSPACE	0A000	Maximum receive space per socket
CTLBUFS	40	Maximum total control buffers
TRANSBUFS	40	Maximum total transaction buffers
MAXTRANS	10	Maximum simultaneous IP transactions
MAXPORTS	1388	Maximum port ids
LOWFIXPID	1	Well-known port id range
HIFIXPID	3FF	
LOWAUTOPID	400	Ephemeral port id range
HIAUTOPID	1387	

### [RIP]

File	Default Value	Description
CTLBUFS	20	Maximum total control buffers
TRANSBUFS	20	Maximum total transaction buffers
MAXTRANS	8	Maximum simultaneous IP transactions
MAXPORTS	80	Maximum port ids

### [IP]

File	Default Value	Description
------	---------------	-------------

IFNAMES	'ETH0, LO0'	Interface names
BUFHEAPSIZE	140	Total receive buffer size in Kbytes
LOCALSUBNETS	1	Enable local subnets
TTL	8	Default segment time to live
TOS	0	Default type of service
ARPTIMEOUT	20	ARP cache flush timeout in minutes
CTLBUFS	80	Maximum total control buffers
TRANSBUFS	80	Maximum total transaction buffers

### [ETH0]

File	Default Value	Description
HOST	'206.103.53.11 5'	Interface IP address
NETMASK	'255.255.255.0 '	Net mask
DEFROUTE	'206.103.53.25 0'	Default route
RCVBUFS	3F	Maximum receive buffers
MAXTRANS	6F	Maximum simultaneous transactions

### [LO0]

File	Default Value	Description
HOST	'127.0.0.1'	Interface IP address
NETMASK	'255.255.0.0'	Net mask
RCVBUFS	3F	Maximum receive buffers
MAXTRANS	6F	Maximum simultaneous transactions

## TCP Job Parameters

### DEFMSS

Default maximum size of segments sent by the TCP job. To avoid fragmentation at the IP level, set this parameter to the smallest maximum packet size that a sent packet is likely to encounter in its route to the destination. Once a connection is established, the source and destination TCPs negotiate an optimum maximum packet size.

### RCVSPACE

Size, in bytes, of the receive buffer area per TCP socket. The receive buffer holds incoming data until it is received at the socket by the application.

### SNDSPACE

Size, in bytes, of the send buffer area per TCP socket. The send buffer holds outgoing data until it is successfully sent to the destination.

### CTLBUFS

Maximum number of control buffers allocated for the TCP job. Control buffers are used by the TCP job whenever data is sent or received through a TCP socket.

If insufficient control buffers are available, an ENOBUFS error is returned to the application. This indicates that the number of configured control buffers for the TCP job should be increased. The default value should be used for most applications.

#### TRANSBUFFS

Maximum number of transaction buffers allocated for the TCP job. Transaction buffers are used by the TCP job whenever data is sent or received through a TCP socket.

If insufficient transaction buffers are available, an ENOBUFS error is returned to the application. This indicates that the number of transaction buffers for the TCP job should be increased. The default value should be used for most applications.

#### MAXTRANS

Maximum number of simultaneous transactions allowed between the TCP job and the IP job. Transactions are used by the TCP job whenever data is sent or received through a TCP socket.

If insufficient transactions are available, an ENOBUFS error is returned to the application. This indicates that the number of transactions for communication between the TCP job and the IP job should be increased. The default value should be used for most applications.

#### MAXPORTS

Maximum number of port ids available to the TCP job. Whenever a TCP socket is bound (see the `bind()` system call), a local port id is assigned to the socket. This parameter specifies the maximum number of unique port ids available.

#### LOWFIXPID, HIFIXPID

When a TCP socket is bound (see the **bind()** system call), the user may specify the local port id that is to be associated with the socket. LOWFIXPID and HIFIXPID:

- Define the range of port id values that may be specified.
- Must be within the range of 0 to MAXPORTS, exclusive, and must not overlap the port id range defined by LOWAUTOPID and HIAUTOPID.

#### LOWAUTOPID, HIAUTOPID

When a TCP socket is bound (see the **bind()** system call), the user may request that the TCP job select the local port id that is to be associated with the socket (known as an *ephemeral* port id). LOWAUTOPID and HIAUTOPID:

- Define the range of port id values that the TCP job may choose from.
- Must be within the range of 0 to MAXPORTS, exclusive, and must not overlap the port id range defined by LOWFIXPID and HIFIXPID.

## UDP Job Parameters

### CHECKSUM

A value of 0 disables checksum calculation on all segments sent or received by the UDP job. A value of 1 enables checksum calculation. This parameter should normally be set to 1.

### RCVSPACE

Size of the receive buffer area per UDP socket, in bytes. The receive buffer holds incoming data until it is received at the socket by the application.

### CTLBUFS

Maximum number of control buffers allocated for the UDP job. Control buffers are used by the UDP job whenever data is sent or received through a UDP socket.

If insufficient control buffers are available, an ENOBUFS error is returned to the application. This indicates that the number of configured control buffers for the UDP job should be increased. The default value should be used for most applications.

### TRANSBUFS

Maximum number of transaction buffers allocated for the UDP job. Transaction buffers are used by the UDP job whenever data is sent or received through a UDP socket.

If insufficient transaction buffers are available, an EBOBUFS error is returned to the application. This indicates that the number of transaction buffers for the UDP job should be increased. The default value should be used for most applications.

### MAXTRANS

Maximum number of simultaneous transactions allowed between the UDP job and the IP job. Transactions are used by the UDP job whenever data is sent or received through a UDP socket.

If insufficient transactions are available, an ENOBUFS error is returned to the application. This indicates that the number of transactions for communication between the UDP job and the IP job should be increased. The default value should be used for most applications.

### MAXPORTS

Maximum number of port ids available to the UDP job. Whenever a UDP socket is bound (see the **bind()** system call), a local port id is assigned to the socket. This parameter specifies the maximum number of unique port ids available.

#### LOWFIXPID, HIFIXPID

When a UDP socket is bound (see the **bind()** system call), the user may specify the local port id that is to be associated with the socket. LOWFIXPID and HIFIXPID:

- Define the range of port id values that may be specified
- Must be within the range of 0 to MAXPORTS, exclusive, and must not overlap the port id range defined by LOWAUTOPID and HIAUTOPID.

#### LOWAUTOPID, HIAUTOPID

When a UDP socket is bound (see the **bind()** system call), the user may request that the UDP job select the local port id that is to be associated with the socket (known as an *ephemeral* port id). LOWAUTOPID and HIAUTOPID:

- Define the range of port id values that the UDP job may choose from.
- Must be within the range of 0 to MAXPORTS, exclusive, and must not overlap the port id range defined by LOWFIXPID and HIFIXPID.

## Raw IP Job Parameters

#### CTLBUFS

Maximum number of control buffers allocated for the Raw IP job. Control buffers are used by the Raw IP job whenever data is sent or received through a Raw IP socket.

If insufficient control buffers are available, an ENOBUFS error is returned to the application. This indicates that the number of configured control buffers for the Raw IP job should be increased. The default value should be used for most applications.

#### TRANSBUFS

Maximum number of transaction buffers allocated for the Raw IP job. Transaction buffers are used by the Raw IP job whenever data is sent or received through a RAW IP socket.

If insufficient transaction buffers are available, an ENOBUFS error is returned to the application. This indicates that the number of transaction buffers for the Raw IP job should be increased. The default value should be used for most applications.

#### MAXTRANS

Maximum number of simultaneous transactions allowed between the Raw IP job and the IP job. Transactions are used by the Raw IP job whenever data is sent or received through a Raw IP socket.

If insufficient transactions are available, an ENOBUFS error is returned to the application. This indicates that the number of transactions for communication between the Raw IP job and the IP job should be increased. The default value should be used for most applications.

## MAXPORTS

Maximum total number of Raw IP sockets that may be created.

## IP Job Parameters

### IFNAMES

A list of interfaces that the IP job may communicate with to send and receive datagrams. Each interface name in the list must match an interface description included in the *:config:tcp.ini* file (e.g., [ETH0] ), and also must match the name associated with a NIC driver loaded in the *:config:tcpstart* submit file.

### BUFHEAPSIZE

Total buffer space, in Kbytes, available to the IP job for sending and receiving datagrams. The buffers specified in the interface descriptions (e.g., the RCVBUFS parameter of the [ETH0] interface description) are allocated from the buffer space defined here.

### LOCALSUBNETS

If this host is directly connected to a network that is divided into subnets, set this parameter to 1. If not, set it to 0.

**TTL** Default time to live for outgoing datagrams. The TTL is used to limit the life of TCP segments and prevent packets from endlessly circling the Internet on the way to some unreachable destination.

**TOS** Default type of service for outgoing datagrams. This parameter encodes both precedence and the type of service as defined by the MIL-STD 1777. The upper three bits of the byte encode the precedence; the lower five bits encode the type of service.

### ARPTIMEOUT

The number of minutes after which a complete ARP table entry will be deleted from the ARP cache if no ARP packets from the associated host are observed on the network.

### CTLBUFS

Maximum number of control buffers allocated for the IP job. Control buffers are used by the IP job whenever data is sent or received.

If insufficient control buffers are available, an ENOBUFS error is returned to the application. This indicates that the number of configured control buffers for the IP job should be increased. The default value should be used for most applications.

#### TRANSBUFS

Maximum number of transaction buffers allocated for the IP job. Transaction buffers are used by the IP job whenever data is sent or received.

If insufficient transaction buffers are available, an ENOBUFS error is returned to the application. This indicates that the number of transaction buffers for the IP job should be increased. The default value should be used for most applications.

### DNS Configuration Parameters

#### DOMAIN

A string containing the name of the local domain.

#### SERVER1

A string that contains the IP address of the primary DNS server used by the client

#### SERVER2

#### SERVER3

Each of these parameters takes a string containing the IP addresses of secondary DNS servers. A total of three servers may be configured. If this section is not defined, or no servers are defined, then DNS name resolution does not occur.

### Network Interface Parameters

**HOST** The IP address associated with this interface.

#### NETMASK

The net mask for the IP address associated with this interface.

#### DEFROUTE

The default route. If the destination of a datagram is not on the network attached to this interface, the default route is used as a destination. The host at the default route address will then forward the datagram to the desired destination.

#### RCVBUFS

The number of buffers allocated to receive datagrams from this interface. These buffers are allocated from the memory pool defined by the IP job's BUFHEAPSIZE configuration parameter.

Set this parameter to the maximum number of datagrams expected to be received at one time on this interface.

#### MAXTRANS

The maximum number of simultaneous transactions between the IP job and this interface. Each datagram sent or received consumes one transaction. The transaction is recycled when the send or receive is processed.

Set this parameter to the sum of the maximum number of incoming datagrams expected at one time (i.e., the value of the RCVBUFS parameter, above) plus the maximum expected number of simultaneous sends to this interface.

## **Loopback Pseudo-driver Interface Parameters**

**HOST** The IP address associated with the loopback interface.

**NETMASK**

The net mask for the IP address associated with the loopback interface.

**RCVBUFS**

The number of buffers allocated to receive datagrams from this interface. These buffers are allocated from the memory pool defined by the IP job's BUFHEAPSIZE configuration parameter.

Set this parameter to the maximum number of datagrams expected to be received at one time on this interface.

**MAXTRANS**

The maximum number of simultaneous transactions between the IP job and this interface. Each datagram sent or received consumes one transaction. The transaction is recycled when the send or receive is processed.

Set this parameter to the sum of the maximum number of incoming datagrams expected at one time (i.e., the value of the RCVBUFS parameter, above) plus the maximum expected number of simultaneous sends to this interface.

□□□



# Files 9

---

This chapter describes the format and contents of network files for TCP/IP. All the files below are installed in the *:config:* directory except *netrc*, which must be in each user's home directory.

<b>File</b>	<b>Description</b>
<i>hosts</i>	host name database
<i>protocols</i>	protocol name database
<i>services</i>	network services database
<i>netrc</i>	ftp autologin information

## hosts

The `:config:hosts` file contains information regarding the known hosts on the Internet. The file should contain an entry for each host and each interface accessible through the network. The primary purpose of the file is to provide the Internet address associated with a symbolic host name. This allows users to specify a name instead of an address.

For each host there should be a single line in the file with this information:

```
Internet_address official_host_name alias ...
```

Each entry begins in column one of the line. Fields are separated by any number of blanks and/or tab characters. A pound sign (#) indicates the beginning of a comment extending to the end of the line.

Specify Internet addresses in the conventional dot notation. The official host name should be the fully-qualified domain name as stored with a **hostname** command or **sethostname()** function. Alias names are optional; there may be more than one, but they must all be on the same line. Host names may contain any character or digit other than space, tab, newline, and pound sign.

See also: Internet addresses, Chapter 1

The `:config:hosts` file must contain an entry for every interface used in the network configuration file `:config:tcp.ini`. For example, if the local host is configured with the software loopback interface (`lo0`), the `hosts` file must contain an entry defining the Internet address (127.0.0.1), the official name (`loopback`) and the aliases (`me` and `localhost`) of that interface. The `hosts` file is the sole source for the name-to-address translations required to initialize the interface correctly.

The `hosts` file must contain the names and addresses of all local interfaces and remote hosts that will be accessed by name.

No specific order is required for either the entries in the file or the list of aliases in a specific entry. Because both the file and the alias list are searched sequentially for a given name, it may be useful to list the most often used names first in order to speed the process, although the file is rarely long enough to make a noticeable difference.

Below is a typical *hosts* file.

```
# :config:hosts
#
# FORMAT:
#   address official_name alias(es)
#
# software loopback interface
127.0.0.1 loopback me localhost
#
# add local network interface definitions here
#
# add remote definitions here (if desired/needed)
```

As network administrator, you should be the owner of this file. Modify it and update it as necessary.

## protocols

The `:config:protocols` file contains the official name, protocol number, and aliases of the protocols with which the `ip` module directly communicates. The protocols are standardized throughout the Internet community and are defined in RFC 1060, Assigned Numbers (Reynolds & Postel).

While the actual protocol numbers are used by the TCP/IP kernel modules, the number-to-name translation information is used primarily by the **netstat** command to display the symbolic name of the protocol instead of its number. There are no required entries in the `protocols` file; the information is used to make displays more readable and meaningful.

For each protocol there should be a single line in the file with this information:

```
official_protocol_name protocol_number aliases
```

The first field on each line should begin in column one. Fields are separated by any number of blanks and/or tab characters. A comment begins with a pound sign (#) and continues to the end of the line. A comment can appear on a separate line or at the end of a line listing network name and address information.

Protocol names can contain any printable character other than a space, tab, newline, or comment character. The official name and number of the protocol should be as defined by the RFC 1060. A list of one or more aliases is optional.

Although no specific order is required for entries in the file, entries are generally maintained in numerical order by protocol number. Below is an example of a `protocols` file.

```
# :config:protocols
#
# FORMAT:
#   official_name protocol_number alias(es)
#
# Internet protocols

ip      0   IP    # reserved for ip (pseudo-protocol number)
icmp    1   ICMP  # internet control message protocol
tcp     6   TCP   # transmission control protocol
egp     8   EGP   # exterior gateway protocol
igp     9   IGP   # any private interior gateway protocol
pup    12   PUP   # PARC universal packet protocol
udp    17   UDP   # user datagram protocol
```

As network administrator, you should be the owner of this file. Update it, if necessary, so that its contents always reflect the protocols operating on the local host. You can add entries if protocols interfacing with ip are added to the local host. The information for this file should be obtained from the most current relevant RFC.

See also: **getprotoent** function, Chapter 11

## netrc

The `:home:netrc` file contains information used to automatically validate FTP connections to one or more remote hosts.

### ⇒ Note

Unlike the Unix environment, the iRMX version of this file is named `netrc` without a beginning `.` (period or dot) in the filename. To hide the file, name it `r?netrc`. When any program refers to `netrc`, the iRMX OS automatically maps it to `r?netrc`.

When **ftp** opens a connection to a remote machine, it checks the user's home directory (`:home:`) for this file. If the file exists, **ftp** checks for an entry for the specified host machine. If such an entry is found, the login name (and optional password) in that entry is supplied to the FTP server without the user being prompted. If the normal validation process used by the FTP server succeeds, the FTP connection is completed without any interactive input by the user. If the file does not contain password information, the user is not prompted for a login name but is prompted for a password.

If `netrc` does not exist for that user, or it exists but contains no entry for the remote host, the user is prompted for a login name and password.

The `netrc` file may contain multiple entries, each specifying login information to a different host name. An entry begins with the keyword `machine` (or the special keyword `default`, described below) and ends with the next occurrence of the word `machine` or with the end of the file. Thus a single entry may be on one line or span multiple lines.

```
<machine name | default> login name [password string]
[account string] <[macdef name
string
]> ...
```

Each entry contains several keyword-value pairs in the format shown above. The first field on each line should begin in column one. Subsequent fields should be separated by spaces or tab characters. Comments begin with a pound sign (`#`) and can appear on a separate line or at the end of a line listing host and login information. The angle brackets shown above are not part of the syntax; they surround multiple items in the same field.

The `machine` keyword identifies the name of a remote host to which autologin is supported. The `name` can be either the official host name or an alias. FTP uses the first entry it finds in `netrc` that matches the name of the remote host specified on the **ftp** command line. The keyword `default` is a special instance of `machine` which matches any host name. Since `default` matches every host name, any entries appearing after it in the file are ignored.

The `login` keyword identifies a login name to be used on the remote machine.

The `password` keyword, where present, specifies the password to the given login. The `account` keyword, where present, specifies a resource access password to be used when required by the remote host. The `account` keyword does not apply to a Unix or iRMX OS and should not be used for such remote systems. Specifying a password or account is optional. If you include this information, also set the file permissions so only the owner can read it. FTP for the iRMX OS, unlike other versions, does not enforce the restriction of access permissions to the owner. FTP does print a warning if the `netrc` file contains account information or passwords.

The `macrodef` keyword identifies an FTP macro definition to be used during a connection to the specified host. The macro name should follow the keyword; the macro definition should begin on the next line of the file and continue until a blank line or the end of the file is encountered. Multiple macros can be defined in this manner, since the next entry does not start until the `machine` or `default` keyword is encountered. The special macro name `init` causes the associated macro to be invoked as the last step in the autologin process.

See also: **ftp** command, *Command Reference*;  
FTP Initialization File, Chapter 5

The following example is an empty `netrc` file. To prevent creation of an unauthorized `netrc` file, such as in the Super user's home directory, install an empty file that only Super can access.

```
# netrc
#
# FORMAT:
#     machine hostname      login name
#     machine hostname      login name      password passwd
#
```

The default permissions of the `netrc` file are to be readable and writable by the owner. All owners of a `netrc` file should modify this file and update it as necessary.

## services

The `:config:services` file contains information about the services available through the transport layer protocols. The services are defined in RFC 1060, Assigned Numbers (Reynolds & Postel), and are standardized throughout the Internet community. The service information is used by applications and TCP/IP kernel modules to identify and validate logical connections. The **netstat** command uses the `services` file to display the symbolic name of the service instead of its number.

The transport layer protocols use ports to identify the endpoints of a logical connection. Specific application services are associated with certain ports, often called *well-known ports*. The server process for the application listens at the assigned port for incoming connections. The Internet community, through RFC 1060, coordinates and standardizes the ports assigned to specific services. Wherever possible, the TCP, UDP, and ISO-TP4 service assignments are coordinated.

For each service there should be a single line in the `services` file with this information:

```
official_service_name port_number/protocol_name aliases
```

The first field on each line should begin in column one. Fields are separated by any number of blanks and/or tab characters. A comment begins with a pound sign (#) and continues to the end of the line. A comment can appear on a separate line or at the end of a line listing service information.

Service names may contain any printable character other than a space, tab, newline, or comment character. The port number and protocol name are considered a single field; a slash separates the port and protocol (for example, `512/tcp`). A list of one or more aliases is optional.

Although there is no specific order required for the entries in this file, entries are generally maintained in numerical order by port number.

As network administrator, you should be the owner of this file. Update it, if necessary, so that its contents always reflect the services available on the local host. Port numbers 0 through 1023 are reserved for privileged processes, and should be used only for the service identified by the Assigned Numbers through RFC. Assign port numbers 1024 and above to custom applications and services unique to the local networking environment.

See also: **getservent** function, Chapter 13

This is a typical *services* file:

```
# :config:services
#
# FORMAT:
# service port/protocol alias(es)
#
# ports 0 - 512 are privileged ports
#
netstat          15/tcp
netstat          15/udp
ftp-data        20/tcp
ftp             21/tcp
telnet          23/tcp
tftp            69/udp
rpcbind         111/udp
rpcbind         111/udp
nfsd            2049/udp
#
#
# ports > 1024 host-specific functions
#
```





This chapter describes the purpose of special files related to the network interface devices, protocols, and protocol families. These files are installed in the */rmx386/jobs* directory:

<b>File</b>	<b>See</b>	<b>Description</b>
ip.job		IP layer
rip.job		Raw IP layer
tcp.job		TCP layer
udp.job		UDP layer
eeepro100.job		NIC driver.
edl.job		NIC-sytle interface to iNA jobs
loopback.job		Loopback pseudo-driver
ne.job		NIC driver.
tulip.job		NIC driver.

## Protocol Jobs

All network protocols are associated with a specific *protocol family*, such as the Internet family *inet*. Associated with each protocol family is an address format, such as the Internet format AF\_INET. A protocol family provides basic services to the protocol implementation to allow it to function within a specific network environment. These services may include packet fragmentation and reassembly, routing, addressing, and basic transport.

A protocol family normally comprises a number of *protocols*, such as the Internet protocols tcp and ip. A protocol normally accepts only one type of address format, as determined by the addressing structure inherent in the design of the protocol family and network architecture.

A *network interface* corresponds to a path through which messages can be sent and received. It can be either a hardware device, such as an Ethernet driver, or a pseudo-device, such as the loopback driver. Network interfaces comprise the lowest layer of the networking subsystem, interacting with the actual transport hardware. A network interface may support more than one protocol family and/or address format. Interface structures and attribute flags are defined in the include file `<net/if.h>`.

The interface address structure contains information about an address associated with a particular interface, maintained by an address family. These structures are linked together so that all addresses for an interface can be located.

## ip.job

The ip.job implements both the Address Resolution Protocol (ARP) and the Internet Protocol (IP).

ARP is used to dynamically map between Internet software addresses and Ethernet hardware addresses.

ARP caches Internet-to-Ethernet address mappings. When the interface requests a mapping for an address not in the cache, ARP queues the message that requires the mapping and broadcasts a message on the associated network, requesting the address mapping. If ARP receives a response, it caches the new mapping and transmits any pending messages to that host. While waiting for a response, ARP will queue only one packet; it keeps only the most recently transmitted packet.

ARP watches passively for hosts impersonating the local host (that is, a host that responds to an ARP mapping request for the local host's address).

IP is the network layer protocol used by the Internet protocol family. It can be accessed through the higher-level Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) as well as directly through the Raw IP interface.

## rip.job

### Syntax

```
#include <netinet/in.h>  
#include <netinet/raw.h>
```

The raw ip service provides a direct interface to lower-level IP. It can be used to implement a new protocol above IP. The **ping** command uses the raw interface. Rip.job only receives packets for the protocol specified.

The IP header and any IP options are left intact by raw on receipt of datagrams.

## tcp.job

### Syntax

```
#include <sys/types.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
```

The Transmission Control Protocol (TCP) provides reliable, flow-controlled, two-way transmission of data. It is a byte-stream protocol used to support the `SOCK_STREAM` abstraction. TCP uses the standard Internet address format augmented by a host-specific collection of port addresses. Thus, each TCP address is composed of an Internet address specifying the host and network, with a specific TCP port on the host identifying the peer entity.

## udp.job

### Syntax

```
#include <sys/types.h>
#include <netinet/in.h>
#include <netinet/udp.h>
```

The User Datagram Protocol (UDP) is a simple, unreliable datagram protocol. UDP streams are connectionless.

UDP address formats are identical to those used by TCP; UDP provides a port identifier in addition to the normal Internet address format. Note that the UDP port space is separate from the TCP port space (that is, a UDP port may not be connected to a TCP port). If the underlying network interface supports broadcast, UDP can send broadcast packets by using a reserved broadcast address. The broadcast address is dependent on the network interface.

## Network Interface Controller (NIC) Jobs

These driver jobs provide an interface between the TCP/IP protocol stack and the network adapters themselves. At least one of the following NIC jobs must be loaded in addition to the `loopback.job` to allow the TCP/IP protocol stack to communicate with other peers on the network.

## loopback.job

The loopback job provides a NIC-style interface to a software loopback mechanism that can be used for performance analysis, software testing, or local communication. The loopback interface is accessible at Internet address 127.0.0.1. By convention, the interface name is *me*, *loopback*, or *localhost*.

The loopback interface should be the last interface configured, as protocols use the order of configuration as an indication of priority. The loopback interface should never be configured first unless no hardware interfaces exist.

## **edl.job**

The `edl.job` provides a NIC-style interface to an iNA 960 network interface `job`. Using this interface allows iRMX-NET and the new TCP/IP protocol stack to use the same hardware to gain access to the network.

## **eepro100.job**

The eepro100.job provides an interface to the Intel EtherExpressPro 100Plus PCI network adapter card.

## **ne.job**

The ne.job provides an interface to NE2000 compatible ISA network adapter cards.

## Tulip.job

The tulip.job provides an interface to a DEC 21143 based PCI network adapter card.



# Library Functions

# 11

This chapter describes functions for the network socket libraries listed below.

<b>C Library</b>	<b>Network Library</b>	<b>Compiler</b>	<b>Model</b>
ciff3m.lib	netf3m.lib	Microsoft	Flat
cifc32.lib	net3c.lib	Intel iC386	Compact

See also: Using Non-Intel Tools to Develop iRMX Application in *Programming Techniques* for non-Intel compiler version numbers.

The libraries are installed in the `/intel/lib` directory and facilitate the programmatic interface to TCP/IP. In the final bind of your application, add one or both libraries to the list of libraries to be linked to your program.



## **Note**

The socket primitives are embedded in the C library.

Table 11-1 lists functions from the socket library.

**Table 11-1. Functions in the Socket Library**

<b>Name</b>	<b>See</b>	<b>Description</b>
<b>accept</b>	<b>accept</b>	accept a connection on a socket
<b>bind</b>	<b>bind</b>	bind a name to a socket
<b>connect</b>	<b>connect</b>	initiate a connection on a socket
<b>getpeername</b>	<b>getpeername</b>	get name of connected peer
<b>getsockname</b>	<b>getsockname</b>	get socket name
<b>getsockopt</b>	<b>getsockopt</b>	get options on sockets
<b>listen</b>	<b>listen</b>	listen for connections on a socket
<b>recv</b>	<b>recv</b>	receive a message from a socket
<b>recvfrom</b>	<b>recv</b>	receive a message from a socket
<b>recvmsg</b>	<b>recv</b>	receive a message from a socket
<b>send</b>	<b>send</b>	send a message from a socket
<b>sendto</b>	<b>send</b>	send a message from a socket
<b>sendmsg</b>	<b>send</b>	send a message from a socket
<b>setsockopt</b>	<b>getsockopt</b>	set options on sockets
<b>shutdown</b>	<b>shutdown</b>	shut down part of a connection
<b>socket</b>	<b>socket</b>	create an endpoint for communication
<b>socketout</b>	<b>socketout</b>	define a timeout for a socket

Table 11-**Error! Bookmark not defined.** lists functions from the network library.

**Table 11-2. Functions in the Network Library (continued)**

<b>Name</b>	<b>See</b>	<b>Description</b>
<b>bcmp</b>	<b>bstring</b>	compare binary strings
<b>bcopy</b>	<b>bstring</b>	copy binary string
<b>bzero</b>	<b>bstring</b>	put zeros in binary string
<b>endhostent</b>	<b>gethostent</b>	close resolver connection
<b>endnetent</b>	<b>getnetent</b>	close networks database
<b>endprotoent</b>	<b>getprotoent</b>	close the protocols database
<b>endservent</b>	<b>getservent</b>	close service database
<b>ffs</b>	<b>ffs</b>	identify set bits
<b>gethostbyaddr</b>	<b>gethostent</b>	get host entry by address
<b>gethostbyname</b>	<b>gethostent</b>	get host entry by name
<b>gethostid</b>	<b>gethostid</b>	get unique id of current host
<b>gethostname</b>	<b>gethostname</b>	get host name
<b>getnetbyaddr</b>	<b>getnetent</b>	get network entry by address
<b>getnetbyname</b>	<b>getnetent</b>	get network entry by name
<b>getnetent</b>	<b>getnetent</b>	get next network entry
<b>getprotobyname</b>	<b>getprotoent</b>	get protocol entry by name
<b>getprotobynumber</b>	<b>getprotoent</b>	get protocol entry
<b>getprotoent</b>	<b>getprotoent</b>	get next protocol entry
<b>getservbyname</b>	<b>getservent</b>	get service entry by name
<b>getservbyport</b>	<b>getservent</b>	get service entry by port
<b>getservent</b>	<b>getservent</b>	get next service entry
<b>htonl</b>	<b>byteorder</b>	host to net order (long)
<b>htons</b>	<b>byteorder</b>	host to net order (short)
<b>inet_addr</b>	<b>inet</b>	string to Internet address
<b>inet_lnaof</b>	<b>inet</b>	get locnet part of address
<b>inet_makeaddr</b>	<b>inet</b>	construct Internet address
<b>inet_netof</b>	<b>inet</b>	get net part of address
<b>inet_network</b>	<b>inet</b>	string to network address
<b>inet_ntoa</b>	<b>inet</b>	Internet address to string
<b>ntohl</b>	<b>byteorder</b>	net to host order (long)
<b>ntohs</b>	<b>byteorder</b>	net to host order (short)
<b>sethostent</b>	<b>gethostent</b>	open resolver connection
<b>sethostid</b>	<b>gethostid</b>	set unique id of current host
<b>sethostname</b>	<b>gethostname</b>	set host name
<b>setnetent</b>	<b>getnetent</b>	open/rewind networks database
<b>setprotoent</b>	<b>getprotoent</b>	open/rewind protocols database
<b>setservent</b>	<b>getservent</b>	open/rewind services database

# Using Sockets

The socket compatibility library constitutes a self-contained interface to the transport level protocols.

A *socket* is an endpoint for communication between processes. Each socket has queues for sending and receiving data.

Sockets are typed according to their communications properties, including such things as whether messages sent and received at a socket require the name of the partner, whether communication is reliable, and what format is used in naming message recipients.

See also: **socket** in this chapter for more information about the types available and their properties

Each set of communications protocols supports addresses of a certain format. An address family is the set of addresses for a specific group of protocols. Each socket has an address chosen from the address family in which the socket was created.

Certain semantics of the basic socket abstractions are protocol-specific. All protocols are expected to support the basic model for their particular socket type, but may, in addition, provide nonstandard facilities or extensions to a mechanism. For example, a protocol supporting the SOCK\_STREAM type may allow more than one byte of out-of-band data to be transmitted per out-of-band (urgent) message.

Use the TCP protocol to support connection-oriented sockets of type SOCK\_STREAM. Use UDP to support connectionless, or datagram, sockets of type SOCK\_DGRAM.

## Calling Sequence for Connection-oriented Applications

Applications that communicate using connections are typically divided in two parts, designated as client and server. The server uses a *passive open*; it opens a socket, then listens for requests for service. The client uses an *active open*; it opens a socket and requests a connection to a specific server. Once the connection is established, the client and server send and receive data as necessary. Typically the client closes the connection, while the server continues to listen for further connection requests.

This is the sequence of calls used by the client:

<b>Client Call</b>	<b>Description</b>
socket( )	Create a SOCK_STREAM socket for connections
bind( )	Bind the socket to a local address (port A)
connect( )	Request a connection to a remote socket, specifying a remote IP address and well-known port B
send( ), recv( )	Send and receive data as determined by the application
shutdown( )	Close the connection

This is the sequence of calls used by the server:

<b>Server Call</b>	<b>Description</b>
socket( )	Create a SOCK_STREAM socket (S <sub>1</sub> ) for connections
bind( )	Bind the socket to well-known port B
listen( )	Listen for connection requests at port B
accept( )	Accept the connection on a new socket S <sub>2</sub>
create_task	Create a child task to perform the service
socket( )	Child task opens SOCK_INHERIT socket so it can receive socket S <sub>2</sub>
shutdown( )	Parent closes S <sub>2</sub> , specifying the job ID of child task, then continues to listen at port B (socket S <sub>1</sub> )
bind( )	Child binds socket S <sub>2</sub> to port C
send( ), recv( )	Child sends and receives data with client (port C to port A)
shutdown( )	Child closes S <sub>2</sub> and exits when client breaks connection

Active sockets initiate connections to passive sockets. By default, TCP sockets are created active; to create a passive socket you must use the **listen( )** function after binding the socket with the **bind( )** function. Only passive sockets may use the **accept( )** call to accept incoming connections. Only active sockets may use the **connect( )** call to initiate connections.

Passive sockets may underspecify their location to match incoming connection requests from multiple hosts. This technique, termed wildcard addressing, allows a single server to provide service to clients on multiple hosts. To establish a socket that listens for all network addresses, bind the Internet address INADDR\_ANY. You may specify the TCP port in this **bind( )** call; if the port is not specified the system will assign one.

Once a connection has been established, the socket's address is fixed by the peer entity's location. The address assigned to the socket is the address associated with the network interface through which packets are being transmitted and received. Normally, this address corresponds to the peer entity's network.

## Calling Sequence for Connectionless Applications

A connectionless application may also be established as a client and server. However, there is no calling sequence that establishes this division of duties. This is the typical sequence of calls for both parties:

<b>Call</b>	<b>Description</b>
socket( )	Create a SOCK_DGRAM socket for UDP
bind( )	Bind the socket to a local address
sendto( ), recvfrom( )	Send and receive data as determined by the application
shutdown( )	Close the connection

However, you can use the **connect()** call to fix the destination for future packets, in which case you can use **recv()** and **send()** calls with the `SOCK_DGRAM` socket.

## Internet Socket Addresses

An Internet address is defined as a discriminated union:

```
struct in_addr      {
    union {
        struct { uchar s_b1, s_b2, s_b3, s_b4; } S_un_b;
        struct { unsigned short s_w1, s_w2; } S_un_w;
        unsigned long S_addr;
    } S_un;
#define      s_addr S_un.S_addr
#define      s_imp  S_un.S_un_w.s_w2
#define      s_net  S_un.S_un_b.s_b1
#define      s_host S_un.S_un_b.s_b2
#define      s_lh   S_un.S_un_b.s_b3
#define      s_impno S_un.S_un_b.s_b4
};
```

In the Internet address family, sockets use this address structure:

```
struct      sockaddr_in {
    uchar          sin_len;
    uchar          sin_family;
    unsigned short sin_port;
    struct in_addr sin_addr;
    char          sin_zero[8];
};
```

### ⇒ Note

The structure above is more correctly called a name than an address. For example, this is the name that you bind the socket to in a **bind()** call, and the name returned by **getsockname()** and **getpeername()**, where the peer uses the Internet address family. The structure is more than just the address; it contains the address family and port number along with the Internet address. However, much of the literature refers to this structure (and the more general `struct sockaddr`) as an address.

See also: `<netinet/in.h>`

## Network and Host Byte Order

Two methods used to store data on different computers are little-endian (the least significant byte of multibyte data is stored in the lowest memory) and big-endian (the most significant byte is stored in the lowest memory).

Within these categories there may also be variation. For example, a certain machine may store words in one order, but swap bytes within the words. Whatever method is used is called *host byte order*; it is specific to the local host.

The Internet standard for binary data to be sent across the network is big-endian. The most significant byte of an integer is sent first. This is *network standard byte order*. It may or may not be the same as the byte order used on the local machine.

To write portable code, translate any binary data from host to network order before sending it. Translate from network to host order after receiving the data. This does not apply to data messages you send between applications; the applications themselves should use data in the same format. It does apply to items that will be used by the protocols on the remote machine.

For example, in the **bind()** and **connect()** calls you specify a port value as part of the local or remote socket address (`sin_port` in the `sockaddr_in` structure). Convert this unsigned short value from host to network order with **htons()**, before placing it in the structure. If your application uses such data (for example, doing a **printf** of a port value obtained from an address), convert from network to host order.

See also: **byteorder()** function, in this chapter

This code fragment shows how to convert the port value properly:

```
#include <netinet/in.h>
int s;
struct sockaddr_in sin;

sin.sin_len = sizeof sin
sin.sin_family = AF_INET;
sin.sin_port = htons (1200);
sin.sin_addr.s_addr = inet_addr ("128.215.18.2");

bind (s, &sin, sizeof sin)
```

This stores the local address in a structure whose elements appear in memory in this order:

Value	Description
0x10	Length of <code>sock_addr_in</code> structure
AF_INET	Address family
0xb004	port 1200 = 0x4b0, swapped to network byte order
0x80d71202	Internet address 128.215.18.2

## Changes From the Standard Socket Interface

This implementation of the socket library has these differences from the standard socket interface:

- In the standard socket interface, you can only specify whether socket calls are blocking or non-blocking. This library provides the **socktout()** call that allows you to define the maximum time to wait for completion of a socket call. The timeout resolution is **10 ms**.
- The address family `AF_UNIX` is not supported.
- The **select()** and **socketpair()** calls are not implemented.
- The `SIGPIPE` and `SIGPOLL` signals are not supported.



### CAUTION

The socket descriptors are not equivalent to the file descriptors used in the C *stdio* interface. Never use the **close()** function on a socket descriptor. You also cannot use such routines as **read()**, **fread()**, **write()**, and **fwrite()**, among others, to read and write data to socket connections.

## Task Priority

User applications that bind to *net3c.lib* should run at a priority between 131 and 254. If you use **rq\_create\_task**, be sure to create the new task with a priority in this range. When applications launch from the CLI, there should not be a problem, because the typical user priority falls in this range: 141 for Super user and 142 for other users.

## Multitasking Considerations

You must ensure that only one iRMX job accesses a connection. Connections may be shared between individual tasks within a single job.

Connections may be inherited by other child jobs if you specify this in the **shutdown()** and **socket()** calls. Since socket descriptors are not file descriptors, and under iRMX are not automatically inherited by child jobs as in Unix, these routines provide a means to imitate this functionality under iRMX.



### CAUTION

Never delete a task while it is executing a socket call. This will cause a general-protection trap in the TCP/IP job, with unpredictable results. Killing the job, on the other hand, is all right.

If a task is hung in a read call, and you want to kill it, first close the connection and wait until the task returns.

Only one task should operate on a socket until a connection is established. After the connection has been established, any number of tasks may use the socket simultaneously. A **shutdown()** may be performed at any time. All tasks executing a call on the socket at that time will return immediately with **errno** set to EBADF.

## Include Files

The descriptions of library functions show which files must be included for each function. The include directory is */intel/include*. To get the correct information from the include files, put this definition in your code:

```
#define _UNIX_SOURCE
```

or use these control statements in the iC-386, and Microsoft or Borland invocation lines, respectively:

```
-df _UNIX_SOURCE  
-d _UNIX_SOURCE  
-D _UNIX_SOURCE
```

To use socket functions, these include files are generally needed:

```
<sys/types.h>  
<sys/errno.h>  
<sys/socket.h>
```

Functions that use an argument of type `struct sockaddr` and use a socket in the DARPA Internet domain (`AF_INET`) may use the Internet view of the `sockaddr` structure, defined in *<netinet/in.h>* as `sockaddr_in`.

## Example Programs

Example programs are installed under the */rmx386/demo/c/tcpip* directory, including:

<i>tcpclient.c</i>	creating a TCP socket as a client
<i>tcpserver.c</i>	creating a TCP socket as a server

## Compiling

The libraries conform to the 386 compact model. When you compile, specify the `compact` option. You also need to use the appropriate control statement.

```
-df __NOALIGN__      (iC-386 compiler)
-D __NOALIGN__      (Borland and Microsoft compiler)
```

(`NOALIGN` preceded by two underscores and followed by two more) in the invocation line.

## Handling Errors

Most socket calls have one or more error returns. Error conditions are indicated by impossible return values (usually -1); individual descriptions specify details.

Unless otherwise noted, function return codes and values are of type integer. An error number is also made available in the external variable **errno**, which is not cleared on successful calls. Thus, you should test **errno** only after an error occurs.

Link to *cstart.obj* and *cifc32.lib* (or the third party compiler equivalent) if your application makes calls to the socket library and you use Intel 32-bit development tools. You must use in-line exception handling or socket calls will fail, often with the command aborted by EH error. To prevent this, add this code to the beginning of `main( )` in your program:

```
EXCEPTIONSTRUCT info;
unsigned short rq_status;
info.exceptionmode = 0;
rqsetexceptionhandler ((EXCEPTIONSTRUCT far *) &info,
&rq_status);
```

Always test the return status of iRMX system calls, and take action if there is an error.

See also: Using Interface Libraries in *Programming Techniques and System Call Reference* for shared C libraries to link to when not using Intel 32-bit application development tools.

## Errno Values for Network Functions

This list describes errors specific to networking as given in `<sys/errno.h>`.

**EADDRINUSE** Address already in use

Only one usage of each address is normally permitted.

**EADDRNOTAVAIL** Can't assign requested address

Normally results from an attempt to create a socket with an address not on this machine.

EAFNOSUPPORT Address family not supported by protocol family  
An address incompatible with the requested protocol was used. For example, you shouldn't necessarily expect to be able to use PUP (PARC Universal Packet) Internet addresses with ARPA Internet protocols.

EALREADY Operation already in progress  
An operation was attempted on a non-blocking object that already had an operation in progress.

EBADF Bad file  
The socket descriptor is invalid.

ECONNABORTED Software caused connection abort  
A connection abort was caused internal to your host machine.

ECONNREFUSED Connection refused  
No connection could be made; the target machine actively refused it. This usually results from trying to connect to a service that is inactive on the foreign host.

ECONNRESET Connection reset by peer  
A connection was forcibly closed by a peer. This normally results from the peer executing a **shutdown** call.

EDESTADDRREQ Destination address required  
A required address was omitted from an operation on a socket.

EHOSTDOWN Host down  
The specified host is not running.

EHOSTUNREACH Host unreachable  
There is no route to the host.

EINPROGRESS Operation now in progress  
An operation that takes a long time to complete (such as a **connect**) was attempted on a non-blocking object.

EISCONN Socket is already connected  
A **connect** request was made on an already connected socket, or a **sendto** or **sendmsg** request on a connected socket specified a destination other than the connected party.

EMSGSIZE Message too long  
A message sent on a socket was larger than the internal message buffer.

ENETDOWN Network is down  
A socket operation encountered a dead network.

ENETRESET Network dropped connection on reset  
The host you were connected to crashed and rebooted.

ENETUNREACH Network is unreachable  
A socket operation was attempted to an unreachable network.

**ENOBUFS** No buffer space available  
An operation on a socket or pipe was not performed because the system lacked sufficient buffer space.

**ENOPROTOPT** Bad protocol option  
A bad option was specified in a **getsockopt** or **setsockopt** call.

**ENOTCONN** Socket is not connected  
A request to send or receive data was disallowed because the socket is not connected.

**EOPNOTSUPP** Operation not supported on socket  
For example, trying to accept a connection on a datagram socket.

**EPFNOSUPPORT** Protocol family not supported  
The protocol family has not been configured into the system or no implementation for it exists.

**EPOWERFAIL** Power failure  
The connection was lost due to a power-fail/recovery cycle.

**EPROTONOSUPPORT** Protocol not supported  
The protocol has not been configured into the system or no implementation for it exists.

**EPROTOTYPE** Protocol wrong type for socket  
A protocol was specified that does not support the semantics of the socket type requested. For example, you cannot use the ARPA Internet UDP protocol with type **SOCK\_STREAM**.

**ESHUTDOWN** Can't send after socket shutdown  
A request to send data was disallowed because the socket had already been shut down with a previous **shutdown** call.

**ESOCKTNOSUPPORT** Socket type not supported  
The support for the socket type has not been configured into the system or no implementation for it exists.

**ETIMEDOUT** Connection timed out  
A **connect** request failed because the connected party did not properly respond after a period of time. The timeout period is dependent on the communication protocol.

**EUNATCH** Protocol driver not attached  
The TCP/IP kernel has not been loaded.

**EWouldBLOCK** Operation would block  
An operation that would cause a process to block was attempted on an object in non-blocking mode.

## Function Reference

This section provides a reference to the functions from the network and socket libraries. Each function reference page provides a brief description of the function, its syntax, any additional information, and related error messages. Functions are ordered alphabetically for quick reference.

## accept

Accepts a connection on a socket.

### Syntax

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

int accept(s, addr, addrlen)
int s;
struct sockaddr *addr;
int *addrlen;
```

### Parameters

<code>s</code>	A socket of type <code>SOCK_STREAM</code> , created with the <code>socket()</code> call, bound to an address with <code>bind()</code> , and currently listening for connections with <code>listen()</code> .
<code>addr</code>	Points to a structure that <code>accept()</code> fills in with the address of the connected peer. The format of the returned address is determined by the domain in which the communication occurs.
<code>addrlen</code>	Initialize to the number of bytes in the buffer referenced by <code>addr</code> . On return, <code>addrlen</code> will contain the actual length in bytes of the returned address.

### Return Value

If the call succeeds, it returns a non-negative integer that is a descriptor for the accepted socket, created by this call. The call returns -1 on an error.

### Additional Information

`Accept()` gets the first connection request from the queue of pending connections and creates a new socket with the same properties as `s`. The call accepts the connection on the new socket and returns a file descriptor for that socket. You cannot accept more connections on the new socket; the original socket `s` remains open.

If no pending connections are present on the queue and the socket is not marked as non-blocking, **accept()** blocks the caller until a connection request arrives. If the socket is marked non-blocking and no pending connections are present on the queue, **accept()** returns an error.

See also: **bind()**, **connect()**, **listen()**, and **socket()** functions, in this chapter

## Errors

[EBADF]

The descriptor is invalid.

[EFAULT]

The `addr` parameter is not in a writable part of the user address space.

[EINVAL]

One of these has occurred:

- The number of bytes allocated for an incoming argument is not sufficient to store the value of that argument.
- The function was issued in the wrong sequence on the transport endpoint referenced by `s`.
- The transport endpoint referred to by `s` is not in the idle state.
- The specified options were in an incorrect format or contained illegal information.
- The amount of user data specified was not within the bounds allowed by the transport provider.

[EIO] One of these has occurred:

- An asynchronous event has occurred on this transport endpoint and requires immediate attention.
- A system error has occurred during execution of this function.
- An unspecified I/O error has occurred.

[ENOTSOCK]

The descriptor references a file, not a socket.

[EOPNOTSUPP]

The referenced socket is not of type `SOCK_STREAM`.

[EUNATCH]

The TCP/IP kernel has not been loaded.

[EWOULDBLOCK]

The socket is marked non-blocking and no connections are present to be accepted.

## bind

Assigns a name to an unnamed socket. When a socket is created with `socket()` it exists in a name space (address family) but has no name assigned. A name must be bound to the socket before the socket can be used.

### Syntax

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

int bind(s, name, namelen)
int s, namelen;
struct sockaddr *name;
```

### Parameters

<code>s</code>	The socket to be bound.
<code>name</code>	Points to the structure containing the name. The rules used in name binding vary between communication domains. In the <code>AF_INET</code> domain, a name consists of the address family ( <code>AF_INET</code> ), a port ID, and an IP address.  See also: <code>Internet Socket Addresses</code> , in this chapter
<code>namelen</code>	The length of the name.

### Return Value

Zero if the call is successful or -1 if an error occurs.

### Errors

[ <code>EADDRINUSE</code> ]	The specified address is already in use.
[ <code>EADDRNOTAVAIL</code> ]	The specified address is not available from the local machine.
[ <code>EBADF</code> ]	<code>s</code> is not a valid descriptor.
[ <code>EFAULT</code> ]	The name parameter is not in a valid part of the user address space.
[ <code>EINVAL</code> ]	The socket is already bound to an address.

[EIO] An unspecified I/O error has occurred.

[ENOTSOCK]

S is not a socket.

[EUNATCH]

The TCP/IP kernel has not been loaded.

## bstring

The **bcmp()**, **bcopy()**, and **bzero()** functions execute binary string operations. They operate on variable length strings of bytes but do not check for null bytes as the routines in **string** do.

### Syntax

```
#include <sys/types.h>
#include <strings.h>

int bcmp(b1, b2, length)
char *b1, *b2;
int length;

int bcopy(b1, b2, length)
char *b1, *b2;
unsigned int length;

void bzero(b, length)
char *b;
int length;
```

### Additional Information

**Bcmp()** compares the first `length` bytes of strings `b1` and `b2`, returning 0 if they are identical, non-zero otherwise. Both strings are assumed to be at least `length` bytes long.

**Bcopy()** copies the first `length` bytes from string `b1` to string `b2`. **Bcopy()** always returns 0.

**Bzero()** places 0s in the first `length` bytes of string `b`.

⇒ **Note**

The **bcopy()** function takes its two `char *` parameters in the reverse order from **strcpy()** and **memcpy()**.

## byteorder

The **htonl()**, **htons()**, **ntohl()**, and **ntohs()** functions convert short (16-bit) and long (32-bit) quantities between network byte order and host byte order.

### Syntax

```
#include <sys/types.h>
#include <sys/endian.h>

unsigned long htonl(hostlong)
unsigned long hostlong;

unsigned short htons(hostshort)
unsigned short hostshort;

unsigned long ntohl(netlong)
unsigned long netlong;

unsigned short ntohs(netshort)
unsigned short netshort;
```

### Additional Information

These routines are most often used in conjunction with Internet addresses and ports as returned by **gethostent()** and **getservent()**. The conversion involves reversing the order of the bytes in the short or long value.

See also: **gethostent()** and **getservent()** functions, in this chapter

## connect

Initiates a connection on a socket. If the socket type is `SOCK_DGRAM`, this call permanently specifies the peer to which datagrams are to be sent. If the type is `SOCK_STREAM`, this call attempts to make a connection to another socket.

### Syntax

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

int connect(s, name, namelen)
int s, namelen;
struct sockaddr *name;
```

### Parameters

<code>s</code>	The local socket
<code>name</code>	The remote socket, specified as an address in the communications space of the socket. Each communications space interprets the name parameter in its own way.
<code>namelen</code>	The length of the name parameter, in bytes.

### Return Value

Zero if the call is successful or -1 if an error occurs.

### Errors

[EADDRINUSE]	Unused.
[EADDRNOTAVAIL]	The specified address is not available on this machine.
[EAFNOSUPPORT]	Unused.
[EBADF]	<code>s</code> is not a valid descriptor.
[ECONNREFUSED]	The attempt to connect was forcefully rejected.
[EFAULT]	The name parameter specifies an area outside the process address space.

[EINVAL]

One of these has occurred:

- The function was issued in the wrong sequence.
- The specified protocol options were in an incorrect format or contained illegal information.
- The amount of user data specified was not within the bounds allowed by the transport provider.
- The number of bytes allocated for an incoming argument is not sufficient to store the value of that argument.

[EIO] An unspecified I/O error has occurred.

[EISCONN]

The socket is already connected.

[ENETUNREACH]

The network isn't reachable from this host.

[ENOTSOCK]

s is a descriptor for a file, not a socket.

[EOPNOTSUPP]

This function is not supported by the underlying transport provider.

[ETIMEDOUT]

Connection establishment timed out without establishing a connection.

[EUNATCH]

The TCP/IP kernel has not been loaded.

[EWOULDBLOCK]

The socket is non-blocking and the connection cannot be completed immediately.

## ffs

Identifies the first set bit in a value.

### Syntax

```
#include <strings.h>

int ffs(mask)
long mask;
```

### Additional Information

This function returns the index of the first (low order) set bit in the argument. Bits are numbered starting at one. If no bits were set (`mask` was 0) a 0 will be returned.

## gethostent

The `gethostbyaddr()`, `gethostbyname()`, `sethostent()`, `endhostent()`, `_gethtbyaddr()`, `_gethtbyname()`, `_sethtent()`, `_gethtent()`, and `_endhtent()` functions set and return entries that identify the network host.

### Syntax

```
#include <netdb.h>

struct hostent *gethostbyaddr(addr, len, type)
char *addr;
int len, type;

struct hostent *gethostbyname(name)
char *name;

void sethostent(stayopen)
int stayopen;

void endhostent( )

struct hostent *_gethtbyaddr(addr, len, type)
char *addr;
int len, type;

struct hostent *_gethtbyname(name)
char *name;

void _sethtent(stayopen)
int stayopen;

struct hostent * _gethtent( )

void _endhtent( )
```

### Additional Information

Network host information can be obtained from either of two places, the *hosts* database or the Domain Name Service (DNS). The iRMX TCP/IP software does not include **named**, the DNS name server. However, it does include a DNS client. The client contacts any DNS name servers running on other hosts on the network and uses their name translation services.

The environment variable `NONAMESERVER` specifies how the two sources are accessed for requested information. If `NONAMESERVER` is set, host information is retrieved from the host's database *hosts*. If `NONAMESERVER` is not set, the host database is searched first; if the search does not succeed, an attempt is made to retrieve the information from a DNS name server on the network.

You can set this environment variable by adding “`NONAMESERVER = 1`” to `:config:r?env`.

A set of functions is also provided to explicitly retrieve information from the *hosts* database. All information obtained from the *hosts* database is contained in a static area, so it must be copied if it is to be saved. Only Internet addresses are understood.

The `gethostbyname()` and `_gethtbyname()` functions retrieve a specific entry by host name. `Gethostbyname()` uses the `NONAMESERVER` environment variable to determine the source; `_gethtbyname()` always searches from the *hosts* database.

The `gethostbyaddr()` and `_gethtbyaddr()` functions retrieve a specific entry by Internet address. `Gethostbyaddr()` uses the `NONAMESERVER` environment variable to determine the source; `_gethtbyaddr()` always searches from the *hosts* database. The Internet address used in both calls should be in host order. The network type should be `AF_INET`, as defined in the system include file *sys/socket.h*. The `len` argument is the length, in bytes, of the address.

To retrieve a sequential series of host entries from the *hosts* database, it is more efficient to use the `_sethtent()`, `_gethtent()`, and `_endhtent()` functions. However, the `sethostent()`, `gethostent()`, and `endhostent()` functions have the same basic behavior described below.

You must pair the calls to `_sethtent()` and `_endhtent()`.

The `_sethtent()` function opens or rewinds (sets the file pointer to 0) the *hosts* database. If passed a 0 value for the argument `stayopen`, `_sethtent()` opens the `:config:hosts` file. Subsequent calls to the `_gethtent()` function return the next entry in the *hosts* database until end of file, opening it if necessary. The `_endhtent()` function closes the database.

If passed a non-zero value for the argument `stayopen`, `_sethtent()` rewinds the `:config:hosts` file or opens it, if it is not already open. Subsequent calls to the `_gethtent()` function return the next entry in the `hosts` database until end of file, opening it if necessary. The `hosts` database remains open until the application executes `exit()`. Calling `_endhtent()` does not close the database.

The host entry has this structure:

```
struct hostent {
    char          * h_name;
    char          ** h_aliases;
    int           h_addrtype;
    int           h_length;
    char          ** h_addr_list;
#define h_addr   h_addr_list[0]
};
```

Where:

`h_name`      The official name of the host.

`h_aliases`   A list of alternate names for the host. The list is terminated by a null string.

`h_addrtype`   The type of host address; `AF_INET` is the only type supported.

`h_length`    The length, in bytes, of the host address.

`h_addr_list`   A list of addresses for the host. The first entry in the list can be retrieved by the defined name `h_addr` as well as by its position in the list. The list is terminated by a 0 address. All host addresses are returned in network byte order.

See also:      `hosts` file, Chapter 11, and the system include file `<sys/socket.h>`

## Errors

A null pointer is returned by `gethostbyaddr()`, `gethostbyname()`, `_getthbyaddr()`, `_getthbyname()`, and `_gethtent()` on an EOF or on an error.

## gethostid

The **gethostid()** and **sethostid()** functions get or set the unique 32-bit identifier of the local host.

### Syntax

```
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>

unsigned long gethostid( )

int sethostid(hostid)
unsigned long hostid;
```

### Return Value

For a successful call, **gethostid()** returns the host ID and **sethostid()** returns 0. If an error occurs, both calls return -1.

### Additional Information

**sethostid()** establishes a 32-bit identifier for the current processor which is intended to be unique among all Internet systems in existence. This is normally an Internet address for the local machine's primary network interface. This call is normally performed at boot time. Only the Super user can set host identifier.

**gethostid()** returns the 32-bit identifier for the current processor.

See also: **hostid** command, *Command Reference*;  
**gethostname()** function, in this chapter

### Errors

- [ EADDRNOTAVAIL ]      The specified host ID is invalid.
- [ EPERM ]              Only the Super user is allowed to set the host identifier.
- [ EUNATCH ]            The TCP/IP kernel has not been loaded.

## gethostname

The `gethostname()` and `sethostname()` functions get and set the local host name.

### Syntax

```
#include <arpa/inet.h>

int gethostname(name, len)
char *name;
int len;

int sethostname(name, len)
char *name;
int len;
```

### Additional Information

**Gethostname()** retrieves the host name and places it in the character string pointed to by the argument `name`. The `len` is the maximum number of characters of the name that can be returned; it should be set to the size of `name`. If the host name is longer than `len`, it will be truncated; it will be null terminated only if the name is shorter than `len`.

**Sethostname()** sets the host name to the argument `name`. Only the Super user can set the host name.

### Errors

Both functions return 0 on success and -1 on failure; **errno** may be one of these:

[EFAULT]

The name was a null pointer.

[EINVAL]

The `len` was less than one.

[EPERM]

Only the Super user can set the host name.

[EUNATCH]

The TCP/IP kernel has not been loaded.

See also: **uname** and **hostname** commands, *Command Reference*

## getnetent

The **getnetbyaddr()**, **getnetbyname()**, **setnetent()**, **getnetent()**, and **endnetent()** functions return information about a network entry from the *:config:networks* database.

### Syntax

```
#include <netdb.h>

struct netent *getnetbyname(name)
char *name;

struct netent *getnetbyaddr(net, type)
int net, type;

void setnetent(stayopen)
int stayopen;

struct netent *getnetent( )

void endnetent( )
```

### Additional Information

A specific entry can be retrieved by the network name with **getnetbyname()**, or by its Internet address with **getnetbyaddr()**. Both functions sequentially search the database for the specified entry. The network address used in the **getnetbyaddr()** call should be in host order; the network type should be `AF_INET`, as defined in the system include file *<sys/socket.h>*.

All returned information is contained in a static area, so it must be copied if it is to be saved. Only Internet network numbers are understood.

To retrieve a sequential series of network entries, it is more efficient to use the **setnetent()**, **getnetent()**, and **endnetent()** functions. You must pair the calls to **setnetent()** and **endnetent()**.

The **setnetent()** function opens or rewinds (sets the file pointer to 0) the *networks* database. If passed a 0 value for the argument *stayopen*, **setnetent()** opens the *:config:networks* file. Subsequent calls to the **getnetent()** function return the next entry in the *networks* database until end of file, opening it if necessary. The **endnetent()** function closes the database.

If passed a non-zero value for the argument `stayopen`, **setnetent()** rewinds the `:config:networks` file or opens it, if it is not already open. Subsequent calls to the **getnetent()** function return the next entry in the `networks` database until end of file, opening it if necessary. The `networks` database remains open until the application executes **exit()**. Calling **endnetent()** does not close the database.

The network entry has this structure:

```
struct netent {
    char          * n_name;
    char          ** n_aliases;
    int           n_addrtype;
    unsigned long n_net;
};
```

Where:

`n_name`      The official name of the network.

`n_aliases`   A list of alternate names for the network. The list is terminated by a null string.

`n_addrtype`   The type of network address; `AF_INET` is the only type supported.

`n_net`        The network number in host order.

See also:    `networks` file, Chapter 9, and the system include file `<sys/socket.h>`

## Errors

A null pointer is returned by **getnetbyaddr()**, **getnetbyname()**, and **getnetent()** on an EOF or on an error.

## getpeername

Returns the socket name of the connected remote socket.

### Syntax

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

int getpeername(s, name, namelen)
int s;
struct sockaddr *name;
int *namelen;
```

### Parameters

<code>s</code>	The local socket.
<code>name</code>	A pointer to the space where the call returns a name.
<code>namelen</code>	Initialize this to indicate the amount of space pointed to by <code>name</code> . On return it contains the actual size of the name returned, in bytes.

### Return Value

Zero if the call is successful or -1 if an error occurs.

### Additional Information

A socket name in the AF\_INET family contains the length, address family, a port number, and the IP address.

See also: **bind()** and **getsockname()** functions, in this chapter

### Errors

[EBADF]

The argument `s` is not a valid descriptor.

[EFAULT]

The `name` parameter points to memory not in a valid part of the process address space.

[EINVAL]

The `namelen` parameter is too small.

[ENOBUFS]

Insufficient resources were available in the system to perform the operation.

[ ENOTCONN ]

The socket is not connected.

[ ENOTSOCK ]

The argument *s* is a file, not a socket.

[ EUNATCH ]

The TCP/IP kernel has not been loaded.

## getprotoent

The `getprotobyname()`, `getprotobynumber()`, `setprotoent()`, `getprotoent()`, and `endprotoent()` functions return an entry from the `:config:protocols` database file.

### Syntax

```
#include <netdb.h>

struct protoent *getprotobyname(name)
char *name;

struct protoent *getprotobynumber(proto)
int proto;

void setprotoent(stayopen)
int stayopen;

struct protoent *getprotoent( )

void endprotoent( )
```

### Additional Information

All returned information is contained in a static area, so it must be copied if it is to be saved. Only Internet protocols are understood.

A specific entry can be retrieved by the protocol name with `getprotobyname()`, or by its number with `getprotobynumber()`. Both functions sequentially search the database for the specified entry. The protocol number used in the `getprotobynumber()` call should be in host order.

To retrieve a sequential series of protocol entries, it is more efficient to use the `setprotoent()`, `getprotoent()`, and `endprotoent()` functions. You must pair the calls to `setprotoent()` and `endprotoent()`.

The `setprotoent()` function opens or rewinds (sets the file pointer to 0) the `protocols` database. If passed a 0 value for the argument `stayopen`, `setprotoent()` opens the `:config:protocols` file. Subsequent calls to the `getprotoent()` function return the next entry in the `protocols` database until end of file, opening it if necessary. The `endprotoent()` function closes the database.

If passed a non-zero value for the argument `stayopen`, **setprotoent()** rewinds the `:config:protocols` file or opens it, if it is not already open. Subsequent calls to the **getprotoent()** function return the next entry in the `protocols` database until end of file, opening it if necessary. The `protocols` database remains open until the application executes **exit()**. Calling **endprotoent()** does not close the database.

The returned protocol entry has this structure:

```
struct protoent {
    char          * p_name;
    char          ** p_aliases;
    unsigned long  p_proto
};
```

Where:

`p_name`      The official name of the protocol.

`p_aliases`   A list of alternate names for the protocol. The list is terminated by a null string.

`p_proto`     The protocol number in host byte order.

so:            *protocols* file, Chapter 9

## Errors

A null pointer is returned by **getprotobynumber()**, **getprotobyname()**, and **getprotoent()** on an EOF or on an error.

## getservent

The `getservbyport()`, `getservbyname()`, `setservent()`, `getservent()`, and `endservent()` functions set or return an entry from the `:config:services` database file.

### Syntax

```
#include <netdb.h>

struct servent *getservbyname(name, proto)
char *name, *proto;

struct servent *getservbyport(port, proto)
int port;
char *proto;

void setservent(stayopen)
int stayopen;

struct servent *getservent( )

void endservent( )
```

### Additional Information

All returned information is contained in a static area, so it must be copied if it is to be saved. Only Internet services are understood.

A specific entry can be retrieved by the service name with `getservbyname()`, or by its port with `getservbyport()`. Both functions sequentially search the database for the specified entry. The port number used in the `getservbyport()` call must be in network order. Use the `htons()` function to convert the port number from host byte order to network byte order.

See also: `htons()` function, in this chapter

To retrieve a sequential series of service entries, it is more efficient to use the `setservent()`, `getservent()`, and `endservent()` functions. You must pair the calls to `setservent()` and `endservent()`.

`Setservent()` opens or rewinds the `services` database. If passed a non-zero value for the argument `stayopen`, `setservent()` will set a flag to prevent the database from being closed until `endservent()` is called.

`Endservent()` closes the `services` database.

**Getservent()** returns the next entry in the *services* database, opening it if necessary. If preceded by a call to **setservent()** with the `stayopen` flag set, it can be called successively to retrieve, in order, all of the database entries. When **getservent()** is called without a previous call to **setservent()**, it opens the database, retrieves the first entry, and closes the database.

The returned service entry has this structure:

```
struct          servent {
    char         * s_name;
    char         ** s_aliases;
    int          s_port;
    char         * s_proto;
};
```

Where:

`s_name`      The official name of the service.

`s_aliases`   A list of alternate names for the service. The list is terminated by a null string.

`s_port`      The port number at which the service can be reached, in network byte order.

`s_proto`      The name of the protocol to be used when contacting the service.

See also:    *protocols* and *services* files, Chapter 9

## Errors

A null pointer is returned by **getservbyaddr()**, **getservbyname()** and **getservent()** on an EOF or on an error.

## getsockname

Returns the current name for the specified socket.

### Syntax

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

int getsockname(s, name, namelen)
int s;
struct sockaddr *name;
int *namelen;
```

### Parameters

<code>s</code>	A local socket.
<code>name</code>	A pointer to the space where the call returns a name.
<code>namelen</code>	Initialize this to indicate the amount of space pointed to by <code>name</code> . On return it contains the actual size of the name returned, in bytes.

### Return Value

Zero if the call is successful or -1 if an error occurs.

### Additional Information

A socket name in the AF\_INET family contains the length, address family, a port number, and the IP address.

See also: **bind()** and **getpeername()** functions, in this chapter

### Errors

[EBADF]

The argument `s` is not a valid descriptor.

[ENOTSOCK]

The argument `s` is a file, not a socket.

[ENOBUFS]

Insufficient resources were available in the system to perform the operation.

[EFAULT]

The name parameter points to memory not in a valid part of the process address space.

[ EADDRNOTAVAIL ]

Socket not bound.

[ EUNATCH ]

The TCP/IP kernel has not been loaded.

See also: **bind()** function, in this chapter

## getsockopt

The **getsockopt()** and **setsockopt()** functions return or set options associated with a socket.

### Syntax

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

int getsockopt(s, level, optname, optval, optlen)
int s, level, optname;
char *optval;
int *optlen;

int setsockopt(s, level, optname, optval, optlen)
int s, level, optname, optlen;
char *optval;
```

### Parameters

<code>s</code>	The socket whose options will be set or returned.
<code>level</code>	The level at which the option is maintained. At the socket level, specify <code>SOL_SOCKET</code> . To manipulate options at any other level, specify the protocol number of the appropriate protocol controlling the option. For example, if the option is to be interpreted by the TCP protocol, set <code>level</code> to the protocol number of TCP ( <code>IPPROTO_TCP</code> ).  See also: <b>getprotoent()</b> function, in this chapter
<code>optname</code>	Specify the name of the option to set or return.
<code>optval</code>	For <b>setsockopt()</b> , specify the value of the option. For <b>getsockopt()</b> , the value is returned in this buffer.
<code>optlen</code>	Specify the length of the <code>optval</code> buffer, in bytes. For <b>getsockopt()</b> , <code>optlen</code> is a pointer; the value it points to is modified on return to indicate the actual size of the <code>optval</code> parameter.

### Return Value

**Getsockopt()** returns 0 if the call succeeds and the specified option is set; otherwise, the return is -1. **Setsockopt()** returns 0 if the call succeeds or -1 if it fails.

## Additional Information

Options may exist at multiple protocol levels; they are always present at the uppermost, or socket, level. To manipulate socket options, you must specify the level at which the option resides and the name of the option. If no option value is to be supplied or returned, `optval` may be set to 0.

The following options are supported:

<i>level</i>	<i>optname</i>	<b>Description</b>
SOL_SOCKET	SO_ATMARK	Report if at OOB mark
	SO_BROADCAST	Permit sending of broadcast msgs
	SO_DONTROUTE	Just use interface addresses
	SO_KEEPAVIVE	Keep connections alive
	SO_LINGER	Linger on close if data present
	SO_OOBINLINE	Leave received OOB data in-line
	SO_RCVLOWAT	Receive low-water mark
	SO_SNDLOWAT	Send low-water mark
	SO_REUSEADDR	Allow local address reuse
	SO_RCVBUF	Size of socket receive buffer
IPPROTO_TCP	SO_SNDBUF	Size of socket send buffer
	SO_REUSEPORT	Allow local port reuse
	TCP_MAXSEG	Get TCP maximum segment size
	TCP_NODELAY	Don't delay send to coalesce packets
IPPROTO_IP	TCP_NOOPT	Don't use TCP options
	TCP_NOPUSH	Don't push last block of write
	IP_TOS	Type of service
	IP_TTL	Segment time to live
	IP_HDRINCL	Application (RAW IP) supplies IP header

`optname` and any specified options are passed without interpretation to the appropriate protocol module for interpretation. Options at other protocol levels vary in format and name.

See also: `socket()` and `getprotoent()` function, in this chapter; *protocols*, Chapter 9

**Errors**

[EBADF]

The argument *s* is not a valid descriptor.

[ENOTSOCK]

The argument *s* is a file, not a socket.

[ENOPROTOOPT]

The option is unknown at the level specified.

[EFAULT]

The options are not in a valid part of the process address space.

[ENOBUFS]

No buffer space is available.

[EINVAL]

Invalid option specified.

[EPROTO]

Invalid level specified.

[EUNATCH]

The TCP/IP kernel has not been loaded.

## inet

The `inet_addr()`, `inet_lnaof()`, `inet_makeaddr()`, `inet_netof()`, `inet_network()`, and `inet_ntoa()` functions manipulate Internet addresses.

### Syntax

```
#include <sys/types.h>
#include <netinet/in.h>

unsigned long inet_addr(cp)
char *cp;

int inet_lnaof(in)
struct in_addr in;

struct in_addr inet_makeaddr(net, host)
int net, host;

int inet_netof(in)
struct in_addr in;

unsigned long inet_network(cp)
char *cp;

char *inet_ntoa(in)
struct in_addr in;
```

### Additional Information

The functions `inet_addr()` and `inet_network()` convert dot notation character strings to the equivalent Internet address and network number, respectively. The function `inet_ntoa()` performs the reverse operation, converting an Internet address to the equivalent dot notation character string.

The function `inet_makeaddr()` constructs an Internet address from a network number and host address. The functions `inet_netof()` and `inet_lnaof()` return the network and local network portions, respectively, of the Internet number passed as an argument.

All functions correctly handle Class A, B, and C Internet addresses; Internet addresses are returned in network byte order.

The dot notation form of an Internet address consists of one to four numbers separated by dots (periods). Each number can be expressed in decimal, octal (leading 0), or hexadecimal (leading 0x).

A four-part address (a.b.c.d) consists of four 8-bit numbers, each in the range 0- 255. The four parts are assigned, in order, to the four bytes in the long Internet address. This is the most commonly used format.

A three-part address (a.b.c) consists of two 8-bit numbers followed by a 16-bit number. The first two parts are assigned in order to the leftmost two bytes of the long Internet address; the third part is placed in the rightmost two bytes. This format is often used for specifying Class B network addresses as *128.net.host*.

A two-part address (a.b) consists of a single 8-bit number followed a 24-bit number. The first part is assigned to the leftmost byte of the long Internet address; the second part is placed in the rightmost three bytes. This format is often used for specifying Class A addresses as *net.host*.

A one-part address is converted to a 32-bit quantity and stored directly in the long Internet address without any byte rearrangement.

See also: **gethostent()** and **getnetent()** functions, in this chapter;  
*hosts* and *networks*, Chapter 9

## Errors

The value -1 is returned by **inet\_addr()** and **inet\_network()** for malformed requests.

## listen

Listens for connection requests on a socket.

### Syntax

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

int listen(s, backlog)
int s, backlog;
```

### Parameters

- s** An unconnected socket of type `SOCK_STREAM`, which has been bound to a name with `bind()`.
- backlog** The maximum number of incoming connection requests that can be queued. If a connection request arrives with the queue full, the client will receive an error with an indication of `ECONNREFUSED`.



#### Note

This parameter is ignored.

### Return Value

Zero if the call is successful or -1 if an error occurs.

### Additional Information

For a server application to accept connections, it must first create a socket with `socket()`, then specify a backlog for incoming connection requests with `listen()`. To complete a connection, accept connection requests with `accept()`.

A `listen(s,0)` call succeeds and sets a connection queue length of 0. This causes all `connect()` attempts to the listening port to fail, with the error `ECONNREFUSED`. A `listen(s,1)` call accepts only a single connection with no pending requests allowed.

See also: `accept()`, `connect()`, and `socket()` functions, in this chapter

**Errors**

[EBADF]

The argument *s* is not a valid descriptor.

[EIO] An **ioctl** of SETQUEUELENGTH has failed.

[ENOTSOCK]

The argument *s* is not a socket.

[EOPNOTSUPP]

The socket is not of a type that supports the operation **listen()**.

[EUNATCH]

The TCP/IP kernel has not been loaded.

## recv

The **recv()**, **recvfrom()**, and **recvmsg()** functions receive a message from a socket. You can use the **recv()** call only on a connected socket, while **recvfrom()** and **recvmsg()** can receive data on a socket whether it is in a connected state or not.

### Syntax

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

int recv(s, buf, len, flags)
int s, len, flags;
char *buf;

int recvfrom(s, buf, len, flags, from, fromlen)
int s, len, flags;
char *buf;
struct sockaddr *from;
int *fromlen;

int recvmsg(s, msg, flags)
int s, flags;
struct msghdr msg[];
```

### Parameters

s	The socket to receive the message from.						
buf	A pointer to a buffer where the received message will be placed.						
len	The length in bytes of the buffer indicated by buf.						
flags	You may set flags to one of the following: <table> <tr> <td>0</td> <td>No special handling.</td> </tr> <tr> <td>MSG_PEEK</td> <td>Peek at the incoming data present on the socket; the data is returned but not consumed, so that subsequent receive operation will see the same data.</td> </tr> <tr> <td>MSG_WAITALL</td> <td>Wait for all data requested.</td> </tr> </table>	0	No special handling.	MSG_PEEK	Peek at the incoming data present on the socket; the data is returned but not consumed, so that subsequent receive operation will see the same data.	MSG_WAITALL	Wait for all data requested.
0	No special handling.						
MSG_PEEK	Peek at the incoming data present on the socket; the data is returned but not consumed, so that subsequent receive operation will see the same data.						
MSG_WAITALL	Wait for all data requested.						



#### Note

This parameter is ignored.

⇒ **Note**

This parameter is ignored.

`from` If `from` is non-zero, the source address of the message is filled in.

`fromlen` Initialize to the size of the buffer associated with `from`. `Fromlen` is modified on return to indicate the actual size of the address stored there.

`msg` The `recvmsg()` call uses a `msghdr` structure to minimize the number of directly supplied parameters. This structure has this form, as defined in `<sys/socket.h>`:

```
struct msghdr {
    caddr_t    msg_name;        /* optional address */
    int        msg_namelen;    /* size of address */
    struct     iovec *msg_iov;  /* scatter/gather array */
    int        msg_iovlen;     /* # elements in msg_iov */
    caddr_t    msg_accrightrights; /* access rights sent/received */
    int        msg_accrightrightslen;
};
```

Here `msg_name` and `msg_namelen` specify the destination address if the socket is unconnected; `msg_name` may be given as a null pointer if no names are desired or required.

## Return Value

The number of bytes received in the message, or -1 if an error occurs.

If a message is too long to fit in the supplied buffer, excess bytes may be discarded depending on the type of socket the message is received from. If no messages are available at the socket, the receive call waits for a message to arrive, unless the socket is non-blocking. In this case a value of -1 is returned with `errno` set to `EWOULDBLOCK`.

See also: `send()` and `socktout()` functions, in this chapter

## Errors

[EBADF]

The argument `s` is an invalid descriptor.

[EFAULT]

The data was specified to be received into a non-existent or protected part of the process address space.

[EINTR]

The receive was interrupted by delivery of a signal before any data was available for the receive.

[EINVAL]

Invalid `flags`, `len` or `fromlen` parameters specified; the number of bytes allocated for the incoming protocol address or options is not sufficient to store the information.

[ENOTSOCK]

The argument `s` is not a socket.

[EOPNOTSUPP]

This function is not supported by the underlying transport provider.

[EUNATCH]

The TCP/IP kernel has not been loaded.

[EWOULDBLOCK]

The socket is marked non-blocking and the receive operation would block.

## send

The **send()**, **sendto()**, and **sendmsg()** functions send a message from one socket to another. **Send()** may be used only when the socket is in a connected state, while **sendto()** and **sendmsg()** may be used at any time.

### Syntax

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

int send(s, buf, len, flags)
int s, len, flags;
char *buf;

int sendto(s, buf, len, flags, to, tolen)
int s, len, flags, tolen;
char *buf;
struct sockaddr *to;

int sendmsg(s, msg, flags)
int s, flags;
struct msghdr msg[];
```

### Parameters

<code>s</code>	The local socket.				
<code>buf</code>	Points to the buffer holding the message to be sent.				
<code>len</code>	The length of the message in bytes, for <b>send()</b> and <b>sendto()</b> .				
<code>flags</code>	May be set to <code>MSG_OOB</code> , to send out-of-band data on sockets that support this notion (for example, <code>SOCK_STREAM</code> ). The underlying protocol must also support out-of-band data. The BSD <code>MSG_DONTROUTE</code> flag is not supported. You may set the flag to one of the following: <table><tr><td><code>0</code></td><td>No special handling.</td></tr><tr><td><code>MSG_OOB</code></td><td>Process out of band data</td></tr></table>	<code>0</code>	No special handling.	<code>MSG_OOB</code>	Process out of band data
<code>0</code>	No special handling.				
<code>MSG_OOB</code>	Process out of band data				
<code>to</code>	The address of the target socket.				
<code>tolen</code>	The length in bytes of the <code>to</code> argument.				
<code>msg</code>	Points to a structure holding the message and information about it.				

The `msg_hdr` structure is as follows:

```
struct msg_hdr {
    caddr_t    msg_name;        /* optional address */
    int        msg_namelen;    /* size of address */
    struct     iovec *msg_iov;  /* scatter/gather array */
    int        msg_iovlen;     /* # elements in msg_iov */
    caddr_t    msg_accrightrights; /* access rights sent/received */
    int        msg_accrightrightslen;
};
```

Here `msg_name` and `msg_namelen` specify the destination address if the socket is unconnected; `msg_name` may be given as a null pointer if no names are desired or required.

## Return Value

The number of characters sent, or -1 if an error occurs.

## Additional Information

No indication of failure to deliver is implicit in a `send()`. Return values of -1 indicate some locally detected errors.

If no message space is available at the socket to hold the message to be transmitted, `send()` normally blocks, unless the socket has been placed in non-blocking I/O mode.

## Errors

[EBADF]

`s` is a invalid descriptor.

[EFAULT]

An invalid user space address was specified for a parameter.

[ENOTSOCK]

The argument `s` is not a socket.

[EOPNOTSUPP]

This function is not supported by the underlying transport provider.

[EUNATCH]

The TCP/IP kernel has not been loaded.

[EWOULDBLOCK]

The socket is marked non-blocking and the requested operation would block.

[EPIPE]

A broken connection exists or a peer has closed the connection.

## shutdown

Shuts down all or part of a full-duplex connection.

### Syntax

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

int shutdown(s, how)
int s, how;
```

### Parameters

*s*                    A connected socket.

*how*                Specifies what part(s) of the connection to shut down:

Value	Description
0	Disallow further receives (not currently implemented)
1	Disallow further sends (not currently implemented)
2	Disallow further receives and sends
<i>job-ID</i>	Transfer the socket to the specified iRMX job.

### Additional Information

This call closes the socket when you disallow both receive and send functions. This can occur with a `how` of 2, or with subsequent calls specifying a `how` of 1 and a `how` of 0.

There is an extension to this call which allows the transfer of a socket to another iRMX job. If the `how` parameter is the job ID of a valid iRMX job, the connection remains and is transferred along with the socket to the specified job. To inherit the socket, the other job must specify `SOCK_INHERIT` as the `type` parameter in a `socket()` call.

The task that bequeaths a socket (using the inherit-style shutdown) will block in the `shutdown()` call until the task in `job-ID` inherits it (calls `socket()` with `SOCK_INHERIT`). If the bequeathing task creates the inheriting task, it must do so prior to calling `shutdown()`.

See also:     **connect()** and **socket()** functions, in this chapter

### Return Value

Zero if the call is successful or -1 if an error occurs.

**Errors**

[EBADF]

s is not a valid descriptor.

[EINVAL]

Invalid value specified for how.

[ENOTSOCK]

s is not a socket.

[EUNATCH]

The TCP/IP kernel has not been loaded.

## socket

Creates an endpoint for communication.

### Syntax

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

int socket(af, type, protocol)
int af, type, protocol;
```

### Parameters

**af** An address format for interpreting addresses specified in later operations:

Value	Format
AF_INET	Internet addresses

**type** Specifies the semantics of communication; one of these:

Value	Meaning
SOCK_STREAM	The socket will be used for connections.
SOCK_DGRAM	The socket will be used for datagrams.
SOCK_RAW	The socket gives direct access to the IP layer.
SOCK_INHERIT	This iRMX job blocks, waiting to inherit another job's open socket

**protocol** The protocol to be used with the socket. For a socket of type SOCK\_STREAM or SOCK\_DGRAM, specify 0 to get the default protocol, IPPROTO\_TCP and IPPROTO\_UDP, respectively. A SOCK\_RAW socket can use IPPROTO\_ICMP or IPPROTO\_RAW. Specify 0 for a SOCK\_INHERIT socket. If you include *<netinet/in.h>*, these values are defined:

Literal	Value	Meaning
IPPROTO_IP	0	dummy for IP
IPPROTO_ICMP	1	Internet control message protocol
IPPROTO_GGP	3	gateway-gateway protocol
IPPROTO_TCP	6	transmission control protocol
IPPROTO_EGP	8	exterior gateway protocol
IPPROTO_PUP	12	PARC universal packet protocol
IPPROTO_UDP	17	user datagram protocol
IPPROTO_IDP	22	Xerox XNS IDP
IPPROTO_RAW	255	raw IP packet

See also: *services* and *protocols* files, Chapter 9

## Return Value

A descriptor referencing the socket, or -1 if an error occurs.

## Additional Information

Sockets of type `SOCK_STREAM` are sequenced, reliable, two-way connection-based byte streams with an out-of-band data transmission mechanism. They are similar to Unix pipes. A stream socket must be in a *connected* state before any data may be sent or received on it. A connection to another socket is created with a `connect()` call. Once connected, data may be transferred using some variant of the `send()` and `recv()` calls. When a session has been completed a `shutdown()` must be performed. Out-of-band data may also be transmitted and received.

The communications protocols used to implement a `SOCK_STREAM` ensure that data is not lost or duplicated. If a piece of data for which the peer protocol has buffer space cannot be successfully transmitted within a reasonable length of time, the connection is considered broken. Such calls indicate an error with -1 returns and with `ETIMEDOUT` as the specific code in the global variable `errno`. The protocols optionally keep sockets viable by forcing transmissions approximately every minute, in the absence of other activity. An error is then indicated if no response can be elicited on an otherwise idle connection for an extended period (e.g., five minutes).

`SOCK_DGRAM` sockets allow you to send and receive datagrams. A datagram is a connectionless, unreliable message with a fixed maximum length, typically small.

See also: `send()` and `recv()` functions, in this chapter

A `SOCK_RAW` socket gives direct access to the IP layer.

If `SOCK_INHERIT` is specified as the `type` parameter, the current job will block in the `socket()` call until another job closes a socket using the current job's ID number as the `how` parameter to the `shutdown()` call. The result is that the job which specifies `SOCK_INHERIT` in its `socket()` call actually inherits an open socket from another iRMX job. This is a non-standard extension to the iRMX implementation of TCP/IP.

See also: `shutdown()` function, in this chapter

All sockets are, by default, `SO_LINGER`. If the socket promises reliable delivery of data, the system will block the process on a shutdown attempt until it is able to transmit the data or until it decides it is unable to deliver the information.

**Errors**

[ EAFNOSUPPORT ]

The specified address family is not supported in this version of the system.

[ EINVAL ]

An unknown error occurred.

[ EIO ] TCP/IP is not configured into the iRMX system.

[ ENOBUFS ]

Unused.

[ EPROTONOSUPPORT ]

Unused.

[ ESOCKTNOSUPPORT ]

The specified socket type is not supported in this address family.

[ EUNATCH ]

The TCP/IP kernel has not been loaded.

## socktout

Defines a maximum time to wait for completion of any subsequent calls on the socket.

### Syntax

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

int socktout(s, val)
int s;
unsigned int val;
```

### Parameters

s	The socket.
val	The timeout period in <b>10-ms</b> units. Setting val to 0xffff disables the timeouts.

### Return Value

Zero if the call is successful or -1 if an error occurs.

### Additional Information

After reaching the timeout limit, the timed-out socket call returns with the return value -1, and **errno** is set to EWOULDBLOCK. The **socktout()** call is a nonstandard extension to the iRMX implementation of TCP/IP. Since the iRMX environment does not have the alarm function built into Unix, this call serves as a substitute measure.

An example of using this function is when you want to receive a datagram. Since UDP is unreliable service, the datagram might be sent but never received. If this occurred, your **recvfrom()** call would block forever unless you had first issued a **socktout()** call.

See also: **accept()**, **connect()**, **recv()**, and **send()** functions, in this chapter

**Errors**

[E2BIG]

val is too big.

[EBADF]

s is not a valid descriptor.

[EUNATCH]

The TCP/IP kernel has not been loaded.



# Recommended Reading

# A

This appendix names a few of the many books available on TCP/IP, NFS, and related subjects.

## TCP/IP

D. E. Comer. *Internetworking with TCP/IP: Principles, Protocols, and Architectures*. Volume I, second edition. Englewood Cliffs, NJ: Prentice-Hall, 1991. This definitive textbook contains both introductory material and detailed reference material on the TCP/IP protocol suite and applications.

M. T. Rose. *The Simple Book: An Introduction to Management of TCP/IP-based Internets*. Englewood Cliffs, NJ: Prentice-Hall, 1991. Describes network management of a TCP/IP internet based on the Simple Network Management Protocol (SNMP).

A. S. Tannenbaum. "Network Protocols," *ACM Computing Surveys*. Volume 13, Number 4, December 1981. This article discusses different types of protocols and network architectures.

W. Richard Stevens. *UNIX Network Programming*. Englewood Cliffs, NJ: Prentice-Hall, 1990. Thoroughly covers common implementations of TCP/IP networking and related UNIX functions, complete with example code.

## NFS

Douglas E. Comer. *Internetworking with TCP/IP*. Volumes I, II, and III. Englewood Cliffs, NJ: Prentice-Hall, 1993. Volume I discusses the fundamental concepts that permeate the Internet suite. Volume II describes how the Internet technological infrastructure is realized. Volume III discusses how to architect and build client/server applications.

# Networks

R. Bowker. *Racal InterLan on Interoperability*. Boxborough, MA: Racal InterLan, 1989. A survey of interoperability that covers industry standards, operating systems, Application Program Interfaces, network operating systems and architectures, and local area networks. Bowker describes the current state of interoperability, and where it is headed.

E. B. Brooner. *The Local Area Network Book*. Indianapolis, IN: Howard W. Sams & Co., Inc., 1984. An introduction to local area networking written for readers with minimal technical background.

J. H. Green. *Local Area Networks*. Glenview, IL: Scott, Foresman and Company, 1985. Discusses networking from the perspective of the business professional.

T. W. Madron. *LANs: applications of IEEE/ANSI 802 standards*. John Wiley & Sons, Inc., 1989. A description of IEEE/ANSI 802 standards that includes a chapter on TCP/IP internetworking.

J. S. Quarterman. *The Matrix: Computer Networks and Conferencing Systems Worldwide*. Bedford, MA: Digital Press, 1990. A comprehensive discussion of networks, protocols and conferencing systems used throughout the world. Part I includes chapters on layers, protocols, network administration, and various standards bodies in the U.S., Europe and Japan. Part II describes what Quarterman calls the *Matrix*: the complex infrastructure of networks and conferencing systems. Quarterman's book ends with an appendix that highlights some applications of law to Computer Mediated Communication (CMC).

M. T. Rose. *The Open Book: A Practical Perspective on OSI*. Englewood Cliffs, NJ: Prentice-Hall, 1991. A clear discussion of the OSI architecture and protocols, with an emphasis on TCP/IP-to-OSI transition issues. Also includes implementation examples.

A. S. Tannenbaum. *Computer Networks*. Englewood Cliffs, NJ: Prentice-Hall, 1981. Tannenbaum discusses all aspects of computer networking.



# Glossary

---

alias	A symbolic name for a domain, host, or user.
ARP	Address Resolution Protocol. An Internet protocol which runs on Ethernets and Token Rings which maps Internet addresses to MAC addresses.
ARPA	Advanced Research Projects Agency. The former name of what is now called DARPA.
ARPANET	A wide area network developed in the 1960s by the Advanced Research Projects Agency. The ARPANET links government, commercial, and academic installations around the world.
BIOS	The Basic I/O System layer of the iRMX OS. This is different from the ROM BIOS stored in ROM on a DOS system.
bps	Bits per second. A measure of data transmission speed.
broadcast	A technique by which a single system on a network can send information to all other systems on the network using a single operation.
BSD	Berkeley Software Distribution. An enhanced Unix operating system that was designed at the University of California at Berkeley. Local network support is one of the enhancements provided by BSD-based systems.
canonical	The standard or regular name or expression, not the alias.
client process	A process activated by a user when issuing a networking command. The client process sends a request for service to a process on the remote host. If the request is honored, a connection is established between the local client and the remote server process.
connection	The path between two protocol modules that provides reliable stream delivery service. In TCP/IP Internet, a connection extends from a TCP module on one machine to a TCP module on the other.

connectionless service	Characteristic of the packet delivery service offered by most hardware and Internet Protocol (IP). The connectionless service treats each packet or datagram as a separate entity that contains source and destination addresses. Usually, connectionless service can drop packets or deliver them out of sequence.
DARPA	Department of Defense Advanced Research Projects Agency. The government agency that funded the ARPANET and later started the Internet.
datagram	The unit transmitted between a pair of internet modules. The Internet Protocol provides for transmitting blocks of data, called datagrams, from sources to destinations. The Internet Protocol does not provide a reliable communication facility. There are no acknowledgments either end-to-end or hop-by-hop. There is no error control for data, only a header checksum. There are no retransmissions. There is no flow control. See IP.
DDN	Defense Data Network. Comprises the MILNET and several other networks.
decimal address	<i>See</i> dotted decimal
default route	A routing table entry which is used to direct any data addressed to any network numbers not explicitly listed in the routing table.
domain	A grouping of hosts according to affiliation. For example, most universities belong to the EDU domain of educational institutions.
DNS	The Domain Name System is a mechanism used in the Internet for translating names of host computers into addresses. The DNS also allows host computers not directly on the Internet to have registered names in the same style.
dotted decimal	An Internet address that uses the base-10 number system, with the parts of the address separated by periods (dots).
EGP	External Gateway Protocol. A protocol which distributes routing information to the routers and gateways which interconnect networks.
EIOS	The Extended I/O System.

Ethernet	A network standard for the hardware and Data Link levels. There are two types of Ethernet: Digital/Intel/Xerox (DIX) and IEEE 802.3.
frame	A self-contained package of data at the link layer.
FTP	File Transfer Protocol. A TCP/IP protocol used for transferring files between hosts on the network.
gateway	A special-purpose dedicated computer that attaches to two or more networks and routes packets from one network to the other. In particular, an Internet gateway routes IP datagrams among the networks it connects. Gateways route packets to other gateways until they can be delivered to the final destination directly across one physical network. This definition is more commonly used in TCP/IP literature for a gateway. However, a more strict definition is that a gateway not only routes between networks but can translate between network protocols as it routes.
globbing	Determines how local filenames are processed by the shell in FTP. With globbing disabled, names specified on the command line are treated literally. With globbing enabled, each local file or pathname is processed for the shell metacharacters * ? [ ] ~ { }. Globbing is always enabled for references to remote files.
header	The portion of a packet, preceding the actual data, containing source and destination addresses and error-checking fields.
host	An individual computer on a network.
host name	A text name that can be used to identify a network host.
host number	The part of an internet address that designates which node on the (sub)network is being addressed.
ICMP	Internet Control Message Protocol. A protocol used by the Internet Protocol to report errors, give limited routing advice, and provide simple low-level services.
ICU	Interactive Configuration Utility. A screen-oriented utility provided by the iRMX III OS to help build the OS desired.
IEEE	Institute of Electrical and Electronics Engineers.

IGP	Interior Gateway Protocol. The generic term applied to any protocol used to propagate how reachable a network is and the routing information within an autonomous system. Although there is no Internet standard IGP, RIP is among the most popular.
internet	Short for internetwork, meaning any connection of two or more local or wide-area networks.
Internet	The global collection of interconnected regional and wide-area networks that use IP as the network layer protocol.
Internet address	A unique address that identifies a host on a TCP/IP network. The Internet address or IP address, consists of four decimal numbers separated by periods (129.84.3.71, for example). Each number has a value between 0 and 255 and represents eight bits of the complete 32-bit address. The Internet address is independent of the hardware to which it is assigned.
Internet Protocol (IP)	The network layer protocol for the Internet. It is the datagram protocol defined by RFC 791.
InterNIC	An organization that provides network users with information about services provided by the network. It is the primary repository for RFCs and Internet drafts.
IP	<i>See</i> Internet Protocol.
IP address	The 32-bit address assigned to hosts that want to participate in the Internet using TCP/IP.
IP datagram	The basic unit of information passed across the Internet. An IP datagram is to the Internet as a hardware packet is to a physical network. It contains a source and destination address along with data.
ISO	International Standards Organization. It developed the OSI (Open Systems Interconnection) reference model for networking.
LAN	Local Area Network. A collection of computers, typically connected by a single transmission cable, joined together for the purpose of sharing resources and facilitating communication. A LAN is limited to a small area such as a single building or a set of closely grouped buildings.

local host	The computer from which the user originates a networking command.
MAC	Medium (or Media) Access Control. For broadcast networks, it is the method which devices use to determine which device has access to the line at any given time.
MAC address	The hardware-level address, such as an Ethernet address.
MTU	The maximum transfer unit for a given interface. This is the largest number of bytes of data that can be transferred in a single packet. For example, the maximum frame size for Ethernet is 1526 bytes, including header information. The MTU is 1500.
network number	The part of an internet address that designates the network to which the addressed node belongs.
NFS	Network File Support. NFS enables hosts to share their local resources with remote hosts (clients) in a manner that hides the heterogeneous nature of a network. For example, a server running the iRMX OS may share a specific directory with a client machine running the Unix OS. The client can access the directory using commands and calls that appear to be directed at local resources.
nslookup	A tool that queries a name server for information about hosts on the network.
octet	Eight bits. Since data is sent across the network as individual bits, the logical 8-bit groups are sometimes called octets instead of bytes.
octal address	An Internet address that uses the base-8 number system.
out-of-band	An urgent data message. TCP attempts to expedite out-of-band data by notifying the application of its urgency. Normal (in-band) data is received after any out-of-band data.
packet	A single unit of data and control information that is transmitted over the network. The length of a packet varies. A single message may be transmitted in one packet or a series of packets.
point-to-point network	A network configuration that consists of two computers connected by a single communications line.

port	A number associated with a particular service. The port number is part of the address bound to a socket. As the Internet address defines a particular host, the port (combined with the protocol) defines the destination on that host. Certain well-known ports are reserved for certain services; for example, 21 for FTP and 23 for TELNET. In general, port numbers greater than 1024 are available for definition by a local application. However, some port numbers in this range have become standardized for certain services through common usage.
POSIX	Portable Operating System Interface. An operating system procedure call interface, based on Unix.
protocol	A formal description of message formats and the rules two computers must follow to exchange those messages. Protocols can describe low-level details of machine-to-machine interfaces (e.g., the order in which bits and bytes are sent across a wire) or high-level exchanges between allocation programs (e.g., the way in which two programs transfer a file across the Internet).
RawEDL	The raw External Data Link layer of iNA software. This interface allows non-OSI protocols such as TCP/IP to use iNA.
RFC	The Internet's Request for Comments documents series. The RFCs are working notes of the Internet research and development community. A document in this series may be on any topic related to computer communication, and may be anything from a meeting report to the specification of a standard.
router	A computer that attaches to two or more networks and routes packets from one network to the other. A router may understand more than one address protocol but does not translate from one protocol to another.
RPC	Remote Procedure Call. A procedure-oriented interface to remote services used to implement the client-server model of distributed computing.
server	A computer that shares its resources, such as printers and files, with other computers on the network.

server process	The remote host process that services the request made by the client process. The server is started up at network boot time as a background process that listens for incoming service requests. When it receives a request, it establishes a connection with the requesting client, spawns a child process, and goes back to listening for more incoming requests.
socket	A communication endpoint. A socket is identified by an address derived from a host's Internet address concatenated with a TCP port number.
Streams	This emulates the STREAMS mechanism on Unix systems. It constructs a series of protocol drivers and code modules to sequentially act on data passing through them. The series of drivers is called a stream, and can act on data flowing in either direction. Upstream is the stream head, put in place below a user process. Downstream is the stream end, a device driver (interface to a hardware device) or pseudo-device driver (interface to other software rather than directly to hardware). With the stream in place, a user process such as FTP makes use of the network hardware without needing to be aware of the protocols managing the data in between.
subnet	A portion of a network, which may be a physically independent network. A subnet shares a network address with other portions of the network and is distinguished by a subnet number. A subnet is to a network what a network is to an internet.
subnet number	A part of the internet address which designates a subnet. It is ignored for the purposes of internet routing, but is used for intranet routing.
TCP	Transmission Control Protocol. A transport layer protocol for the Internet. It is a connection-oriented, stream protocol defined by RFC 793.
TCP/IP	Transmission Control Protocol/Internet Protocol. A set of computer networking protocols and applications that enables two or more hosts to communicate. TCP/IP includes a suite of protocols besides TCP and IP; it has been widely adopted as a networking standard.
TELNET	A TCP/IP protocol used for remote login between hosts.

TFTP

Trivial File Transfer Protocol. A Department of Defense standard for transferring files between hosts. TFTP lacks the error-checking and user-authentication facilities offered by FTP.

UDP

User Datagram Protocol. A transport layer protocol for the Internet, defined by RFC 768. It is a datagram protocol that adds a level of reliability to IP datagrams.



- /dev/loop file, 70
- /etc/hosts file
  - retrieving entries from, 98
- /etc/networks file
  - retrieving entries from, 102
- /etc/protocols file
  - retrieving entries from, 106
- /etc/services file
  - retrieving entries from, 108
- :config/
  - services file, 66
- :config:ftpusers file, 41
- :config:hosts file, 14, 41, 46, 60
- :config:hosts.equiv file, 41
- :config:inetinit.cf file, 41
  - verifying configuration of, 45
- :config:networks file, 41
- :config:nfsstart.csd file, 16
- :config:nfsstop.csd file, 16
- :config:protocols file, 41, 62
- :config:r/init2.log file, 17
- :config:services file, 41
- :\config:tcpstart.csd file, 17
- :config:tcpstart.csd file, 15
- :config:tcpstop.csd file, 15
- :home:netrc file, 64
- :home:rhosts file, 41

## A

- accept(), 79, 88
- accepting connections, 88
- access permissions
  - controlling, 13
  - of netrc file, 65
- active open, 78
- address

- broadcast, 7
- host, 7, 60, 97
- Internet, 6, 7, 9, 60
- Internet, 8
- InterNIC Registration Services, 8
- network, 7
- subnet, 7

- address family, 70
- Address Resolution Protocol, see ARP
- administrative commands, 43
- AF\_INET, 70, 83, 90, 126
- AF\_UNIX, 82
- ARP, 71
- automatic FTP login, 35

## B

- bcmp(), 92
- bcopy(), 92
- Berkeley R-series commands, 4, 5
- big-endian, 81
- binary string operations, 92
- bind(), 75, 78, 79, 80, 81
- bits, set, 96
- books
  - network, 78
  - TCP/IP, 77
- broadcast address, 7
  - and UDP, 74
- byte order, 81, 93, 106, 108
- bzero(), 92

## C

- client, 1
  - process, 37
  - using sockets, 78
- clonable device, 69
- close(), 82

- commands
  - TCP/IP administrative, 43
- config:inetinit.cf file, 60
- configuration
  - verifying network, 45
- configuration files, 59
  - /
    - config/
      - services, 66
    - :config:hosts, 14, 60
    - :config:protocols, 62
    - netrc, 64
- connect(), 78, 81, 94
- connections
  - accepting, 88
  - closing, 124
  - inheriting, 82, 124, 127
  - queuing requests for, 117
  - requesting, 94
  - sharing by tasks, 82
  - to remote host, 24, 31
  - waiting for, 117
- create\_task call, 79

## D

- daemons and services, 37
- datagram, 127
  - protocol, 74
  - socket calls, 79
- domain name, 9, 60
- domain name service (DNS), 9, 14
- dot notation, 6, 115

## E

- endhostent(), 97
- endnetent(), 102
- endpoint, 78, 126
- endprotoent(), 106
- endservent(), 108
- errno, 83
  - testing, 84
  - values, 84
- errors
  - general-protection, 83

- handling, 84
- returned by network functions, 84
- escape character
  - telnet, 23, 24
- Ethernet
  - adapter card, see NIC
- example programs, 83

## F

- ffs(), 96
- file descriptors, 82
- File Transfer Protocol, see FTP
- ftp
  - automatic login, 64
- FTP, 3
  - ? command, 30
  - automatic login, 35
  - client, 37
  - commands, 30
  - commands, accessing on-line help, 30
  - connecting to hosts, 31
  - disabling, 38
  - file size limitations, 34
  - get command, 33
  - macros, 65
  - naming conventions when transferring files, 34
  - netrc file, 65
  - open command, 31
  - put command, 32
  - quitting, 35
  - remote connection, 31
  - server, 37, 38
  - starting, 30
  - transferring files, 32, 33
  - transferring large files, 34
  - using, 29
- ftpd server, 37, 38
- ftpusers file, 41
- full-duplex, 124

## G

- gateway, 2
- gethostbyaddr(), 97
- gethostbyname(), 97

- gethostid(), 100
- gethostname(), 101
- getnetbyaddr(), 102
- getnetbyname(), 102
- getnetent(), 102
- getpeername(), 75, 80, 104
- getprotobyname(), 106
- getprotobynumber(), 106
- getprotoent(), 106
- getservbyname(), 108
- getservbyport(), 108
- getservent(), 108
- getsockname(), 75, 80, 110
- getsockopt(), 112

## H

- hardware requirements, 13
- host
  - address, 7, 97
  - byte order, 81, 93, 106
  - local, 1
  - local ID, 100
  - local name of, 98, 101
  - official name of, 1, 9
  - remote, 1
- host name, 97, 101
  - mapping to Internet address, 60
- hostid command, 3, 17
- hostname command, 3, 17
- hosts file, 14, 41, 60
- hosts.equiv file, 41
- htonl(), 93
- htons(), 81, 93

## I

- ICMP (Internet Control Message Protocol), 6, 126
- inet\_addr(), 115
- inet\_lnaof(), 115
- inet\_makeaddr(), 115
- inet\_netof(), 115
- inet\_network(), 115
- inet\_ntoa(), 115
- inheriting sockets, 82, 124, 127
- interfaces

- verifying functionality of, 46
- Internet address, 6, 7, 9
  - classes of, 6, 7
  - converting formats of, 115
  - dot notation, 6
  - get or set local, 100
  - mapping to host name, 60
  - obtaining, 8
  - structure of, 80
- InterNIC Registration Services, 8
- IP, 71
  - address, see Internet address
- IPPROTO\_ICMP, 126
- IPPROTO\_RAW, 126
- IPPROTO\_TCP, 112, 126
- IPPROTO\_UDP, 126

## J

- job
  - inherits socket, 124
  - sharing connections, 82
  - TCP/IP kernel, 15

## L

- library functions, 77
- listen(), 79, 117
- little-endian, 81
- logging in
  - to remote host, 24, 31, 35
- loopback, 60

## M

- macro, defining in netrc file, 65
- maximum transfer unit, see MTU
- message
  - receiving, 119
  - sending, 122
- MSG\_DONTROUTE flag, 122
- msghdr structure, 120, 123
- MTU
  - checking, 46
- multitasking, 82

## N

name

- domain, 9
- host, 60

name server, 21, 29

net3c.lib library, 77

netrc file, 35, 41, 64

netstat command, 38, 43

- a option, 39, 44

- i option, 45

network

- address, 7, 102

- books about, 78

- byte order, 81, 93, 108

- configuration files, 41

- daemons and servers

  - telnetd, 39

- databases, 41

- interface adapter (NIA), see NIC

- library functions, 77

- name, 102

- services, 37

- testing the TCP/IP, 43

- verifying configuration of, 45

- verifying TCP/IP services, 44

Network Information Center, 10

networks file, 41

NIC (network interface controller), 13

nslookup command, 21

ntohl(), 93

ntohs(), 93

## O

options, for socket, 112

OVL286 (80286 overlay generator), 117

## P

passive open, 78

password

- restricting access to, 36, 65

ping command, 47, 72

port, 73, 74

- changing byte order of, 81

- numbers, 66, 108

- well-known, 66

prompt

- ftp, 30

- telnet, 22

protocols, 1

- family, 70

- file, 41, 62

- name and number, 106

- name database, 62, 106

- translating numbers to names, 62

pseudo-device, 70

## Q

query commands

- hostid, 3

- hostname, 3

## R

r?netrc file, 35, 64

raw

- interface, 72

- testing transport layer, 47

rcp command, 4

rcp, file size limitations, 34

recv(), 78, 79, 119

recvfrom(), 79, 119

recvmsg(), 119

remote copy, 4

remote login, 5, 24, 31, 35

remote shell, 4

remote Unix host for telnet, setting up, 17

Request for Comments, see RFC

RFC, 10

- 1060, Assigned Numbers, 62, 66

- where to obtain, 10

rhosts file, 41

rlogin command, 5

rsh command, 4

ruptime command, 5

ruptime command, -a option, 5

## S

security

- controlling access to files, 13

- password information, 36, 65
- select(), 82
- send(), 78, 79, 122
- sendmsg(), 122
- sendto(), 79, 122
- server, 1
  - process, 37
  - using sockets, 79
- services and daemons, 37
- services file, 41, 66, 108
- set bits, 96
- sethostent(), 97
- sethostid(), 100
- sethostname(), 101
- setnetent(), 102
- setprotoent(), 106
- setservent(), 108
- setsockopt(), 112
- shutdown(), 78, 124
- site commands, 34
- slipd, 41
- slipd.cf file, 41
- SO\_LINGER, 127
- SOCK\_DGRAM socket, 80, 94
  - creating, 126
- SOCK\_INHERIT type, 124, 126
- SOCK\_RAW socket
  - creating, 126
- SOCK\_STREAM socket, 78, 88, 94
  - creating, 126
- sockaddr\_in structure, 80, 83
- socket, 5
  - calls made by client, 78, 84
  - calls made by server, 79
  - connection-oriented calls, 78, 79
  - creating, 126
  - datagram calls, 79
  - definition of, 78
  - descriptor, 82, 127
  - inheriting, 82, 124, 127
  - name of local, 110
  - name of remote, 104
  - naming, 90
  - nonstandard implementation, 82, 75
  - options for, 112
- socket(), 78, 79, 126
- socket3c.lib library, 76, 84

- socketpair(), 82
- socktout(), 82, 75
- SOL\_SOCKET level, 112
- startup script, see tcpstart.csd
- strings
  - binary, 92
- subnet mask, 7
- system calls, 5

## T

- task
  - deleting, 83
- tcp driver, 73
- TCP/IP, 73
  - books about, 77
  - configuring
    - sysloadable job, 14
  - installing, 11
  - kernel job, 37
  - protocols, 1
  - required hardware, 13
  - stopping, 15
  - testing, 45
  - testing setup, 16
  - troubleshooting, 16
- tcplisten daemon, 37
- tcpstart.csd file, 15, 17, 37, 38, 39
- tcpstop.csd file, 15
- telnet, 21, 39
  - close command, 27
  - command mode, 21, 22, 26, 28
  - commands, 27, 28
  - connecting to hosts, 24
  - disabling, 39
  - escape character, 23, 24
  - input mode, 21, 22
  - open command, 22
  - prompt, 22
  - quit command, 27
  - quitting a session, 24
  - remote connection, 24
  - remote Unix host, setting up, 17
  - status command, 27
- TELNET, 3
- telnetd server, 39
- terminal

- characteristics for user sessions, setting, 18
- creating a definition for the PC console, 17
- setting the type on Unix, 25
- tests
  - network, 43
- TFTP (Trivial File Transfer Protocol)
  - file size limitations, 34
- timeout, 75
- TLI, 5
- Transmission Control Protocol, see TCP
- troubleshooting, 16

## U

- UDP (User Datagram Protocol), 74
  - testing, 45
- udp driver, 74
- ulimit command, 34
- User Datagram Protocol, see UDP

## W

- well-known ports, 66