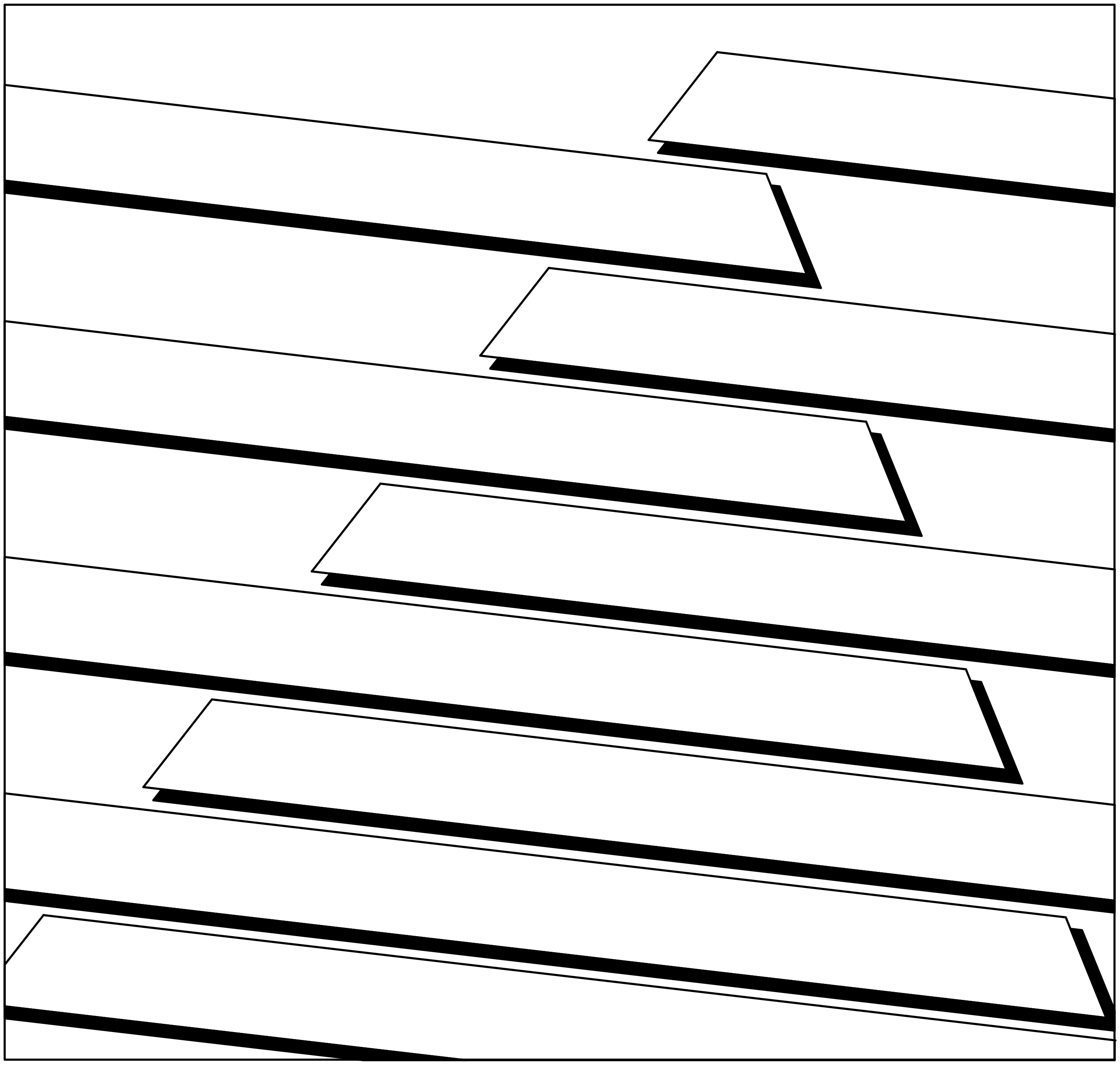
ALLEN-BRADLEY

OS-9 Internet

Software Reference Manual



Acknowledgements

OS-9/Internet was designed and implemented by Kim Kempf and Andy Nicholson. Many thanks are due to Microware and its staff for making this product possible. Special thanks to Yeongleh Lee, Daryl Jarman, Kim Kempf, Pam Weber, and Peter Dibble for their help for this update; the information and proofreading are greatly appreciated.

Copyright and Revision History

This manual reflects Version 1.4 of OS-9/Internet. Version 1.4 of OS-9/Internet is to be used with Version 2.3 or greater of the OS-9 Operating System.

Publication Editor:	Ellen Grant, Walden Miller
Revision:	B
Publication Date:	July, 1992
Product Number:	INT-68-NA-68-MO

Disclaimer

Reproduction Notice

Trademarks

Microware Systems Corporation • 1900 N.W. 114th Street
Des Moines, Iowa 50325-7077 • Phone: 515/224-1929

Chapter 1

OS-9®/Internet Overview

OS-9/Internet allows you to communicate with other computer systems connected by Internet from your OS-9 system. Once connected, you can send and receive data from other systems and log on to other systems.

This chapter introduces you to the basics of networking and provides you with a working knowledge of Internet.

If you are already familiar with networking, skip this chapter and go on to the following chapters:

This chapter covers the following topics:

- Basic networking terminology
- Available network protocols
- Network addressing
- Files used in networking
- Available header files

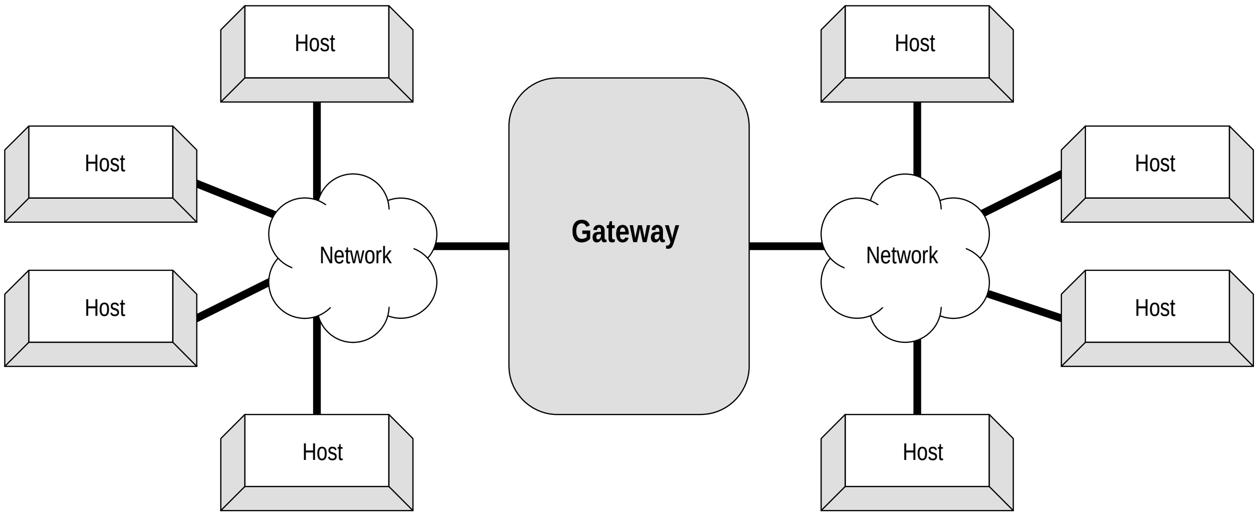
Basic Networking Terminology

A computer **network** is the hardware and software used to allow computers to communicate with each other. Each computer system connected to the network is a **host**. Hosts can be, and usually are, located at different sites. Hosts can also be of different types. For example, you may have your OS-9 system connected to a network that consists of other OS-9 systems and/or UNIX, VAX/VMS, or other systems.

An **internet** is the connection of two or more networks that allows computers on one network to communicate with computers on another network. An internet is sometimes referred to as the **internetwork**.

Networks are connected to each other by **gateways**. Gateways are computers dedicated to connecting two or more networks.

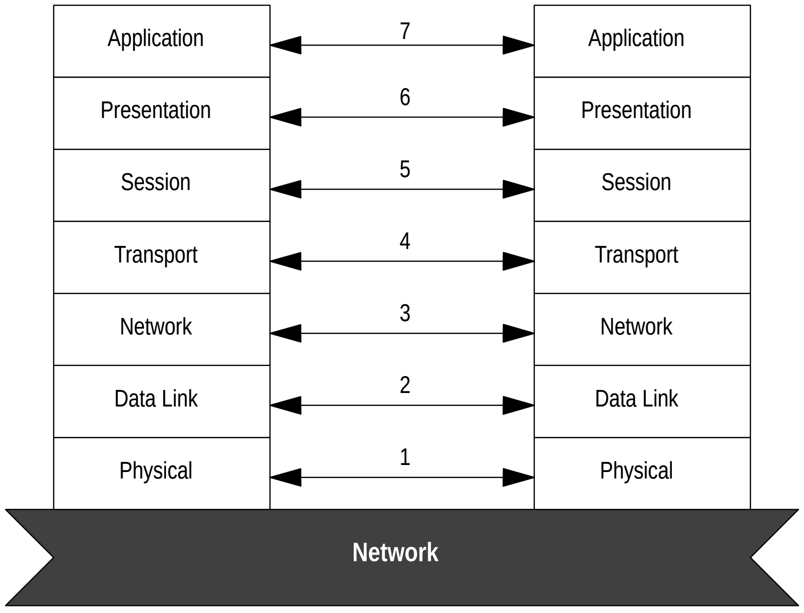
A gateway routes messages from one network to another network.



OSI Model for Networks

The International Organization for Standardization (ISO) created a model, known as the Open Systems Interconnection model, or **OSI model**, as a conceptual framework for developing protocol standards. A protocol is a set of rules or conventions that allow modularity and portability.

The OSI model contains seven conceptual layers organized as follows:



These layers are defined as follows:

- **Physical Hardware Connection**
  The physical layer specifies the physical interconnection including the electrical characteristics of voltage and current. It is the lowest layer of the architecture.

- **Data Link**
  The data link layer is the hardware interface layer. Because the raw hardware delivers only a stream of bits, this layer defines the format of frames and specifies how two machines recognize frame boundaries. Because transmission errors can destroy data, this layer includes error detection in the form of a frame checksum. You can think of this as a window to the physical wires and communication to particular Ethernet hardware (such as the AM7990 LANCE device).

- **Network**
  The network layer specifies the format of a particular network and contains the functionality that completes the definition of the interaction between the host and the network. It defines the basic unit of transfer across the network and includes the concepts of destination addressing and routing. The IP protocol is an example.

- **Transport**
  The transport layer provides end-to-end reliability by having the destination host communicate with the source host. This layer double checks to make sure that no machine in the middle has failed. The TCP protocol is an example.

- **Session**
  The session layer keeps the connection open while an application is running.

- **Presentation**
  The presentation layer provides the interface between the network and the application. The presentation layer is concerned with the representation of data being exchanged. It converts the application data into some standard form by using encoding rules.

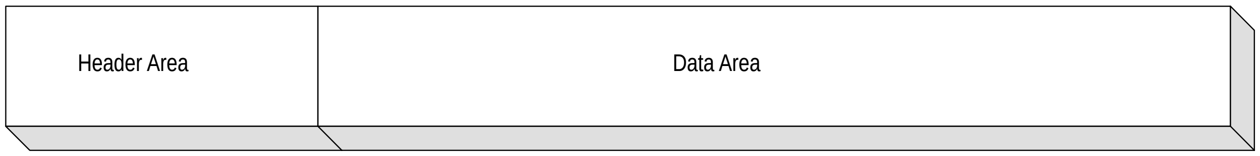
- **Application**
  The application layer communicates directly with the application processes and provides all services directly required by the application process.

Datagrams

When information is passed from one host to another, either on the same network or across gateways, the data is called a **packet**. Packets are the actual physical data to transfer across the network layer. A **datagram** is a specific type of packet and is the basic unit of information passed on a network. The Internet calls this unit the Internet datagram or IP datagram.

A datagram is divided into a header area and a data area. The datagram header contains the source and the destination Internet Protocol address and a type field identifying the datagram's contents.

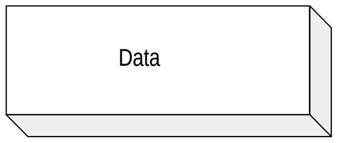
Figure 1.1
Illustration of a Basic Datagram



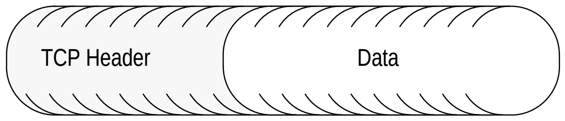
The datagram size depends on the network's **maximum transfer unit** (MTU). Because each network may have a different MTU, Internet divides large datagrams into smaller **fragments** when the datagram needs to pass through a network that has a small MTU. The process of dividing datagrams into fragments is known as **fragmentation**.

Fragmentation usually occurs at a gateway somewhere along the path between the datagram source and its final destination. The gateway receives a datagram from a network with a large MTU and must route it over a network for which the MTU is smaller than the datagram size. The size of the fragment must always be a multiple of eight and is chosen so each fragment can be shipped across the underlying network in a single **frame**. A frame is passed across the data link layer and contains an **encapsulated datagram**.

The addition of information to the datagram is called **encapsulation**. Datagrams are encapsulated with information as they pass through layers of the network. The following illustrates this concept:

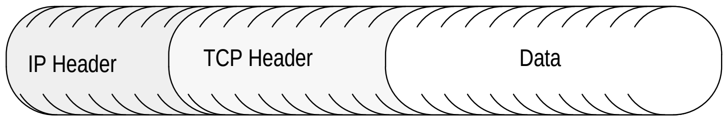


Initial Packet of Data to Transfer

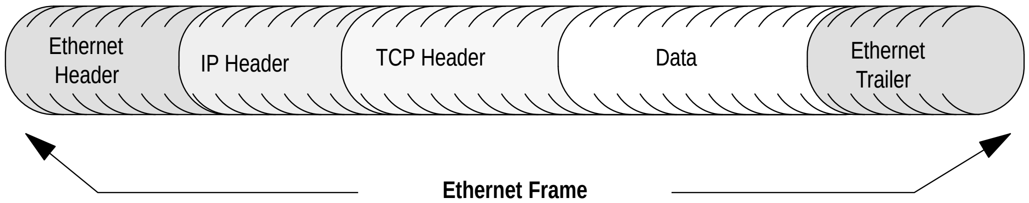


Protocol = TCP

Initial Packet with
16–Bit TCP Source Port Number
16–Bit TCP Destination Port Number



Protocol = IP
Internet 32–Bit Source Address
Internet 32–Bit Destination Address



Ethernet 48–Bit Source Address
Ethernet 48–Bit Destination Address

Fragments are reassembled to produce a complete copy of the original datagram before it can be processed at the destination. However, the datagram may remain fragmented until it reaches its ultimate destination.

Client and Service Processes

The terms client and server appear frequently in networking documentation. A **server** process provides a specific service accessible over the network. A **client** process is any process that wishes to use a service provided by a server.

Protocols

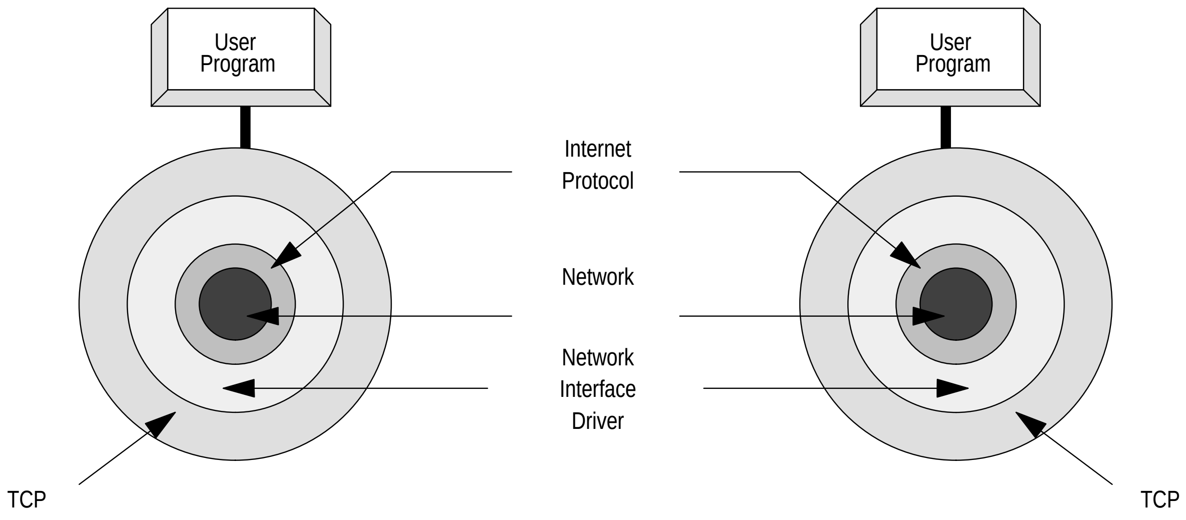
When client and server processes communicate, both processes must follow a set of rules and conventions. These rules are known as a **protocol**. Without protocols, hosts could not communicate with each other.

The protocols that you need to be familiar with when using OS-9/Internet are:

- Internet Protocol (IP)
- Transmission Control Protocol (TCP)
- User Datagram Protocol (UDP)

Internet Protocol (IP)

The Internet Protocol (IP) is the Internet datagram delivery protocol. IP is a lower-level protocol located above the network interface drivers and below the higher-level protocols such as the User Datagram Protocol (UDP) and the Transmission Control Protocol (TCP). IP is the protocol that provides packet delivery service for higher level protocols such as TCP and UDP. Programs may use IP through the higher-level protocols such as UDP and TCP.



Due to the IP layer orientation, datagrams flow through the IP layer in two directions:

- from the network IP to user processes
- from user processes down to the network

The IP layer provides for a checksum of the header portion, but not the data portion of the datagram. IP computes the checksum value and sets it when datagrams are sent. The checksum is checked when datagrams are received.

A checksum is a small, integer value used for detecting errors when data is transmitted from one machine to another.

The IP layer supports fragmentation and reassembly. If the datagram is larger than the MTU of the network interface, datagrams are fragmented on output. Fragments of received datagrams are dropped from the reassembly queues if the complete datagram is not reconstructed within a short time period.

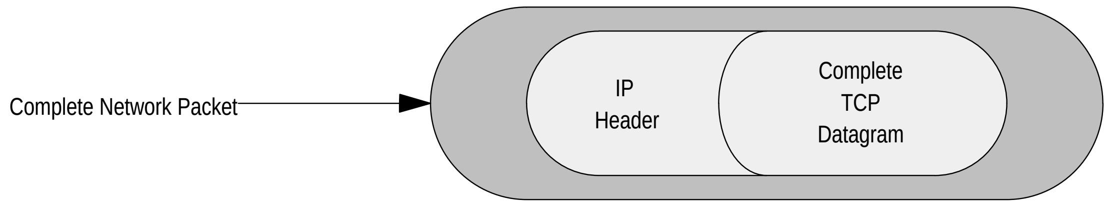
If an error is discovered while a datagram is in the network interface driver layer, the error is passed to the user process.

Transmission Control Protocol (TCP)

The Transmission Control Protocol (TCP) is layered on top of the IP layer. It is a standard transport level protocol that allows a process on one machine to send a stream of data to a process on another machine. TCP provides reliable, flow controlled, orderly, two-way transmission of data between connected processes. You can also shut down one direction of flow across a TCP connection, leaving a one-way (simplex) connection.

Software implementing TCP usually resides in the operating system and uses IP to transmit information across the underlying Internet. TCP assumes that the underlying datagram service is unreliable. Therefore, it performs a checksum of all data to help implement reliability. TCP uses IP's host level addressing and adds its one per-host collection of port addresses. The endpoints of a TCP connection are identified by the combination of an IP address and a TCP port number.

The TCP packets are encapsulated into the IP datagrams:



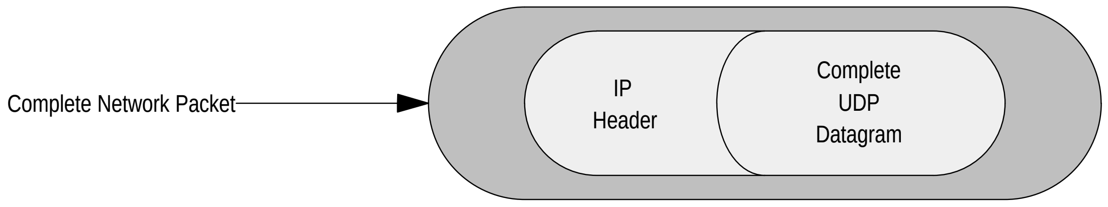
User Datagram Protocol (UDP)

The User Datagram Protocol (UDP) is also layered on top of the IP layer. UDP is a simple, unreliable datagram protocol that allows an application on one machine to send a datagram to an application on another machine using IP to deliver datagrams. Conceptually, the important difference between UDP datagrams and IP datagrams is that UDP includes a protocol port number, allowing the sender to distinguish among multiple application programs on the remote machine.

Like TCP, UDP uses a port number along with an IP address to identify the endpoint of communication.

UDP datagrams are not reliable. They can be lost or discarded in a variety of ways, including a failure of the underlying communication mechanism. UDP implements a checksum over the data portion of the packet. If the checksum of a received packet is incorrect, the packet is dropped without sending an error message to the application. Each UDP socket is provided with a queue for receiving packets. This queue has a limited capacity, and any datagrams that arrive once the capacity of the queue is reached are silently discarded.

The UDP packets are also encapsulated into the IP datagrams:



Internet Addresses

Regardless of which protocol you use to send messages across a network, each host is assigned a 32-bit Internet address, or **IP address**.

An IP address consists of two portions: a network portion, netid, and a host portion, hostid.

IP addresses are usually represented visually as four decimal numbers, where each decimal digit encodes one byte of the 32-bit IP address. This is referred to as **dot notation**. IP addresses specified using the dot notation use one of the following forms:

- a.b.c.d
- a.b.c
- a.b
- a

When four parts are specified, each is interpreted as a byte of data and assigned, from left to right, to the four bytes of an IP address. When an IP address is viewed as a 32-bit integer quantity, VAX bytes are ordered from right to left (d.c.b.a).

When a three part address is specified, the last part is interpreted as a 16-bit quantity and placed in the right most two bytes of the network address.

When a two part address is supplied, the last part is interpreted as a 24-bit quantity and placed in the right most three bytes of the network address.

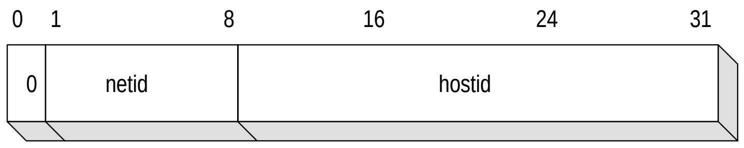
When only one part of the address is specified, the value is stored directly in the network address without any byte rearrangement.

Each integer is chosen carefully to make routing efficient. Basically, an IP address identifies a network to which a host is attached and the specific host attached to the network.

Networks are separated into several classes. The three primary classes of IP addresses are:

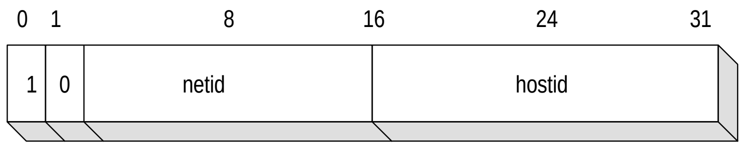
Class A
Used for networks that have more than 65536 hosts. Seven bits are allocated to the netid and 24 bits to the hostid. For example, the 32-bit hexadecimal value 0x0102ff04 is equal to 1.2.255.4. This represents a Class A address with a netid of 1 and a hostid of 0x02ff04.

A two part address format is convenient for specifying Class A network addresses as net.host.

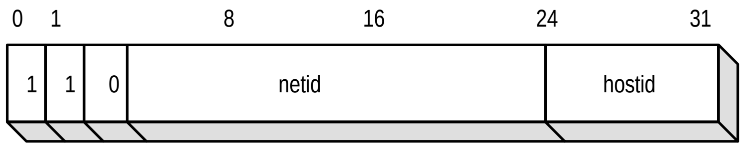


Class B
Used for intermediate-sized networks with between 256 and 65536 hosts. Fourteen bits are allocated to the netid and 16 bits to the hostid. An example Class B decimal address is 128.100.0.5.

A three part address format is convenient for specifying Class B network addresses as 128.net.host.



Class C
Used for networks with fewer than 256 hosts. Twenty-one bits are allocated to the netid and only 8 bits to the hostid. An example Class C decimal address is 192.100.2.10.



Using C language specifications, numbers supplied as parts in a dot notation may be interpreted as:

The numbers designated as:	Contain:
Decimal	no leading character(s).
Octal	a leading 0.
Hexadecimal	a leading 0x or 0X.

Specifying IP addresses in dot notation allows you to determine the network class from the three high-order bits. The IP address has been defined to allow you to extract either the hostid or the netid. Gateways base routing on the netid and depend on such efficient extraction.

Port Numbers

In addition to IP addresses, **port numbers** are used. Port numbers distinguish which process on one system is communicating with which process on another system. Port numbers are selected by the user and the system when the Internet sockets are opened and bound. Port numbers are unsigned word values, so the maximum number of ports in the system is 65535.

Port numbers less than 1024 are reserved. Select port numbers greater than 1024 for your services.

Important: Sockets are discussed in Chapter 3, Sockets.

The following is an example showing the use of port numbers. This example uses ftp, a utility used for transferring files between systems. ftp is discussed in Chapter 5, Transferring Files with FTP.

In this example, a user on a client OS-9 system named delta is using a service, ftp, from a server OS-9 system named gamma. You can think of the connection as a quintuple association between the following:

- protocol
- local host IP address
- local host port number
- foreign host IP address
- foreign host port number

As shown in the services file, ftp has a standard port number of 21. Any requests to port 21 will reach the ftp daemon. The ftp daemon (the server) will bind to port number 21 and wait for connections from the client.

Important: Binding sockets is discussed in Chapter 3, Sockets.

To initiate the ftp (or client) request, the client system, delta, connects to port number 21 on the server system, gamma. A port number on delta is needed; however, the specific port number used is unimportant. The ISP system produces a random port number for the client system in order to complete the quintuple association.

Important: Connecting sockets is discussed in Chapter 3, Sockets.

In this case, the association is as follows:

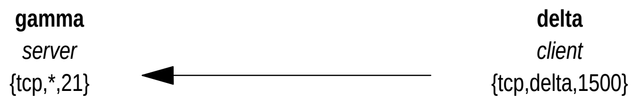
$$\underbrace{\text{\{tcp, 128.10.0.3, 21,}}_{\substack{\text{gamma} \\ \textit{server}}} \quad \underbrace{\text{tcp, 128.10.0.7, 1500\}}}_{\substack{\text{delta} \\ \textit{client}}}$$

This association shows the following information:

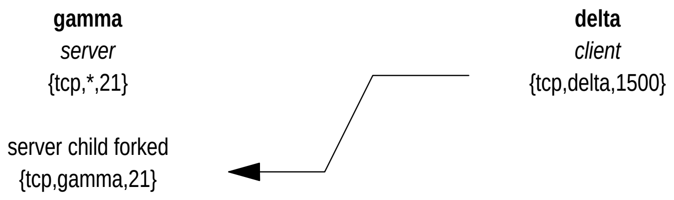
- TCP is the protocol used.
- The local host IP address is 128.10.0.7.
- The local host port number is 1500.
- The foreign host IP address is 128.10.0.3.
- The foreign host port number is 21.

The following shows graphically how ftp and the quintuple association work.

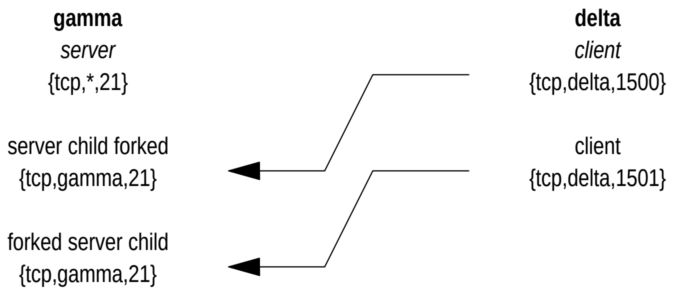
The gamma system has ftpd bound to port number 21 and is waiting for a connection. The delta system initiates a connection to gamma system port 21.



Once the connection is established, the quintuple association is complete. Then, ftpd forks the daemon child, ftpdc, to handle the ftp connection. The appropriate paths to the socket are duplicated by ftpd and passed to ftpdc when forked. ftpd closes the appropriate socket path and waits for another connection:



If another user wishes to ftp a file from gamma to delta, the following association occurs:



Important: The client as a system simply chose port number 1501.

With this, there are two unique associations:

{tcp,gamma,21,delta,1500}
{tcp,gamma,21,delta,1501}

To view these associations, use the inetstat utility. Refer to Chapter 8, OS9/Internet Utilities, for information on inetstat.

Networking Files

Four files are necessary to provide the network with pertinent information:

The file:	Is a list of:
hosts	hosts known to your system. At a minimum, you need to add an entry for each of your hosts (including the host you are using) to the file.
networks	networks analogous to hosts. Update this file for your environment.
protocols	protocols used by the user-level Internet software.
services	services used by the user-level Internet software.

Each of the data files is normally created from the official data base maintained at the Network Information Control Center (NIC). However, local changes may be required to bring it up to date regarding unofficial aliases, unknown hosts, networks, and/or services.

Each file contains single line entries consisting of a number of fields and (optionally) comments. Fields are separated by any number of blanks and/or tab characters. A pound sign (#) indicates the beginning of a comment. The comment includes all characters up to the end of the line.

The hosts File

The hosts file contains information regarding the known hosts on the DARPA Internet. For each host, a single line entry should be present. Each entry contains the following:

- internet address
- official host nam
- aliases (optional)

Network addresses are specified in the conventional "." notation using the inet_addr() routine from the Internet address manipulation library. Host names can contain any printable character other than a field delimiter, newline, or comment character.

The following example hosts entry consists of an address, name, and comment:

```
192.1.1.1 balin #documentation
```

Important: You can arbitrarily choose the Internet addresses for a LAN not connected to other networks. If you use OS-9/Internet to connect to an existing LAN or the Internet, consult your local network administration conventions to determine the proper network address.

The networks File

The networks file contains information regarding the known networks which comprise the DARPA Internet. A single line entry should be present for each network. Each entry consists of the following information:

- official network name
- network number
- aliases (optional)

Network numbers are specified in the conventional “.” notation using the inet_network() routine from the Internet address manipulation library. Network names can contain any printable character other than a field delimiter, newline, or comment character.

The following example networks entry consists of a name, number, alias, and comment:

```
arpanet 10 arpa #just a comment
```

The protocols File

The protocols file contains information regarding the known protocols used in the DARPA Internet. A single line entry should be present for each protocol. Each entry contains the following information:

- official protocol name
- protocol number
- aliases (optional)

Protocol names can contain any printable character other than a field delimiter, newline, or comment character.

The following example protocols entry consists of a name, number, alias, and comment:

```
udp 17 UDP # user datagram protocol
```

The services File

The services file contains information regarding known services available to the DARPA Internet. A service is a reserved port number for a specific application. For example, ftp is a service reserved at port 21. Each service also specifies the protocol it uses. Because each network can have a unique services file, networks can offer different services.

A single line entry should be present in the services file for each service. Each entry contains the following information:

- official service name
- port number at which the service reside
- official protocol name
- aliases (optional)

Service names can contain any printable character other than a field delimiter, newline, or comment character.

The port number and protocol name are considered a single item; a slash character (/) separates the port number and protocol name (for example, 512/tcp).

The following example services entry consists of a service name, port number, protocol name, alias, and comment:

```
shell 515/tcp cmd #no passwords used
```

To create a service, you need to select a port number greater than 1024 (port numbers less than 1024 are reserved), a protocol, and a name and add this information to the services file.

Three C calls are available for using services. Refer to the descriptions of getservent(), getservbyport(), and getservbyname() in Chapter 9, Socket/Network C Libraries, for more information.

Header Files

Three header files are associated with Berkeley sockets:

Header File:	Description:
in.h	Contains the main structure used in all Internet applications.
inetdb.h	Contains structures on the Internet database.
socket.h	Contains structures which are imbedded in the Internet system, as well as many macros such as AF_INET.

Important: Chapter 3, Sockets, contains more detailed information about sockets.

The main internet application structure is sockaddr_in which is defined in the in.h header file. The structure is defined as follows:

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

struct in_addr  {
    u_long s_addr;
};
```

The hostent structure is in the netdb.h header file. It is used to get address information about any host in the inetdb database:

```
struct hostent  {
    char *h_name;       /* pointer to host name */
    char **h_aliases;   /* pointer to the pointer to the alias for the host */
    int  h_addrtype;    /* host address type */
    int  h_length;      /* length of host */
    char *h_addr;       /* pointer to the address of the host */
};
```

OS-9/Internet Operation

A short overview is provided here for basic understanding of Internet modules.

OS-9/Internet consists of four major software components:

- the socket manager (SOCKMAN)
- the protocol handlers (TCP/UDP/IP
- the mbuf facility (F$Mbuf)
- the interface manager (IFMAN)

The socket manager provides program-level access to the network systems. The socket abstraction was designed as a part of the Berkeley Standard Distribution of UNIX (BSD4.x). This abstraction was designed to support multiple protocols and address families under one data-driven interface. The socket layer (SOCKMAN) handles all interactions with the user program through the socklib.l library. SOCKMAN also provides access to the protocol modules that handle the network-specific functions of the communication.

Important: Refer to chapter 3, Sockets, for more information about sockets.

SOCKMAN calls the protocol handlers on behalf of the user programs. SOCKMAN automatically binds and calls the protocol modules. SOCKMAN provides a calling interface and standard services (such as timers) to allow future protocols to be developed and integrated into the system without affecting existing protocols. Finally, SOCKMAN (with the cooperation of IFMAN) provides a list of active device drivers which are called by the lowest level of the protocol routines.

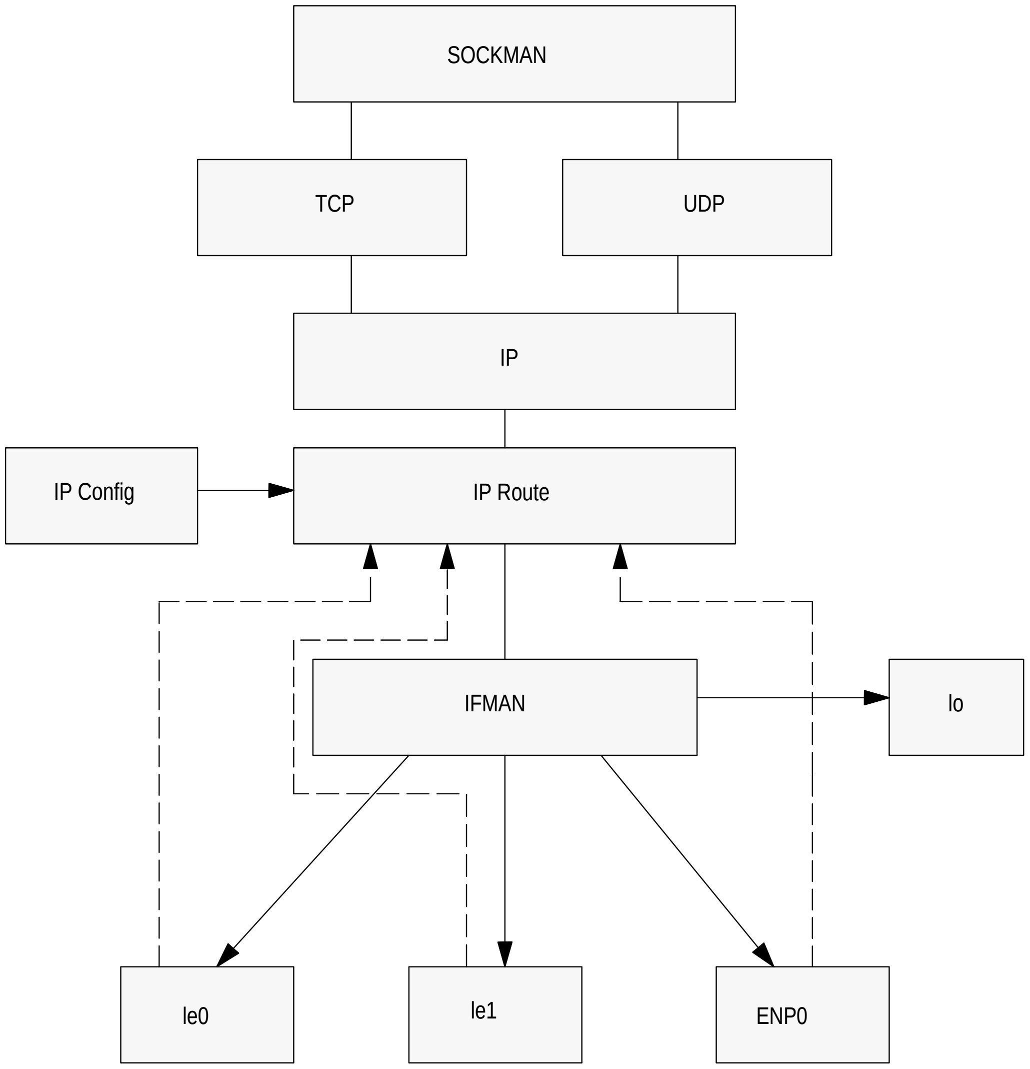
Important: The protocol modules are actually OS-9 subroutine modules.

The Internet system uses the mbuf facility to dynamically allocate the memory it needs. This mbuf memory pool is allocated from the kernel when the Internet system is started. The Internet system then allocates memory exclusively from this mbuf memory pool. This makes memory requests for the Internet system faster than allocating memory from the kernel.

The interface manager (IFMAN) maintains information about configured network link-level interfaces in a hardware-independent manner. IFMAN is unusual compared to traditional OS-9 file managers in that it is a “passive” entity. IFMAN simply maintains a list of data structures that describe the characteristics and state of the network interfaces.

The data structure for the particular device contains all the appropriate information for protocol modules directly calling the driver. This is important because some network data is not intended for any process on the target machine but rather the network software itself (for example, routing or broadcast data).

The following illustrates the OS-9/Internet system:



Chapter 2

Installing OS-9/Internet

Introduction

This chapter covers the following topics:

- hardware set up
- installing Internet
- making device descriptors for the VME/147 and VME/167 system
- making device descriptors for CMC and ENPLLD systems
- the ipconfig data module
- getting Internet up and running on VME/147 and VME/167 systems
- OS-9/Internet implementation notes

Hardware Set Up

Before installing OS-9/Internet on your system, read the notes appropriate to your system:

Notes to VME/147 Users

The VME/147 is pre-configured to use with Ethernet. However, make sure that the Ethernet address is in the battery-backed RAM (BBRAM).

This installation procedure applies to all versions of the VME/147 CPU except the RF/-SRF (Reduced Feature) versions. –RF/–SRF versions do not have an on-board Ethernet interface.

The Ethernet address consists of six bytes:

- The first three bytes are the manufacturer's ID (Motorola ID = 08 00 3E).
- The last three bytes are the serial number of the specific VME/147 system (2x xx xx). These bytes should match the last three bytes of the Ethernet Station ID printed on the label attached to the rear of the front panel.

The Ethernet driver combines the three bytes found in BBRAM at address FFFE 0778 with 08 00 3E to form the six-byte Ethernet address.

Important: This is the physical hardware Ethernet address which is not the same as the Internet address.

You can use Motorola’s 147 debugger, the ROM debugger, or sysdbg to confirm that the Ethernet address is in the BBRAM. The MVME/147 User Manual contains a discussion of the Ethernet ID.

After you have confirmed/set the BBRAM ID, connect your Ethernet cable to the Ethernet connector on the MVME/712 module. Confirm that the Ethernet transceiver power LED, located on the VME/712 module, is lit.

Notes to VME/167 Users

The VME/167 is pre-configured to use with Ethernet. However, make sure that the Ethernet address is in the battery-backed RAM (BBRAM).

The Ethernet address consists of six bytes:

- The first three bytes are the manufacturer’s ID (Motorola ID = 08 00 3E).
- The last three bytes are the serial number of the specific VME/167 system. These six bytes are found on a label on the back of the front panel of the CPU board. Ensure that these six bytes match the bytes in BBRAM at address FFFC 1F2C.

You can use Motorola’s 167 debugger, the ROM debugger, or sysdbg to confirm that the Ethernet address is in the BBRAM. The MVME/167 User Manual contains a discussion of the Ethernet ID.

After you have confirmed/set the BBRAM ID, connect your Ethernet cable to the Ethernet connector on the MVME/712 module. Confirm that the Ethernet transceiver power LED, located on the VME/712 module, is lit.

Notes to VME/374 Users

The vme374 driver is known to work with the Motorola VME/374 Multi-Protocol Ethernet Module using the standard default “common environment” ROMs that come with the board. The driver automatically provides the VME/374 with firmware code; no ROM change is required.

Setting the Ethernet Address
The VME/374 has NVRAM used to store the station Ethernet address. This is preset. You do not need to make changes unless you determine the Ethernet the board is using is not correct. You can connect an RS-232 terminal to the P2 connector on the VME/374. On power up with the standard ROMs, a configuration menu can reset the Ethernet address. This procedure is described in the Motorola manual for the board. Alternately, you can place the Ethernet address in the me0.a device descriptor and that address will override the board setting. A label on the inside of the front panel gives the proper Ethernet address for your particular board.

VME/374 Jumper Settings

Check all jumper options on the VME/374 to make sure they are appropriate for your system configuration. The factory jumper settings should be satisfactory for most systems. Change the BG/BR jumpers if your system uses a level other than level 3 for bus request/grant. There are no IRQ jumpers. The descriptor specifies the level and vector the VME/374 uses. The default setting is IRQ level 4 and vector 206.

The port address for the VME/374 is shipped as A32 space at 0xffd00000 and the enclosed descriptor is also set to this. To change to A24 or a different A32 address, you must change the me0.a file and remake the descriptor to match the hardware setting.

The VME/374 decodes to a lmeg window (0x0 – 0xfffff). In A32 space, the VME/374 can be addressed in the range:

```
0xf8000000
0xfff00000
```

J7 sets the module base address. To change to A24 space:

- Remove the jumper on pins 1 - 2 on J7.
- Remove the jumpers on pins 9 - 10, 11 - 12, and 13 - 14 on J6.
- Move all the jumpers on J8 to the opposite row of pins.
- Set the base address on J7.

The VME/374 can appear in any lmeg window in A24 space.

Considerations for Use with a VME/165 CPU Module

The Motorola VME/165 CPU module does not provide a D16 master access window which is required to access the VME/374. If you are using the VME/374 with a VME/165 CPU, you must use the included driver module, vme374_dl6. This driver uses only word-length memory accesses and properly operates on the VME/165. The result is a bigger and slower driver than the normal vme374 driver. However, this is required on the VME/165 and it does not appear to affect performance.

Direct Memory Access (DMA) Considerations

As shipped, the me0 descriptor is set to cause the driver to enable DMA. The VME/374 firmware needs to know the location of the host CPU’s local memory in the VMEbus address space to properly translate the host addresses to bus addresses. The descriptor is set assuming the host CPU’s local memory appears at bus address 0x00000000. If your CPU’s local memory appears at 0x00400000 on the VMEbus, change vme374_hmoff value in the descriptor (me0.d) as follows:

```
*
* The following is the VMEbus slave access address for the main CPU’s
* memory.
 dc.l  0x00400000  u_int  vme374_hmoff; /* host memory (DMA) offset */
```

If for some reason you cannot or do not want to use DMA, change the use_dma value in the descriptor:

```
  dc.w l       u_short    use_dma;   /* 0=host copy, 1=374 dma copy */
```

When set to 0, the host copies the data from the VME/374's shared memory. Note that the copy is *not* performed within the IRQ service routine so it does not affect IRQ latency. The driver firmware buffers up to 128 incoming packets in the VME/374's RAM before any are lost.

Depending on the type of allowed slave access, you may need to change the address modifier used by the VME/374 when accessing the host CPU's memory. By default, the descriptor is set to use A32 supervisor access as specified by the vme_am value:

```
dc.w 0x0d u_short vme_am;  /* address modifier code for VMEbus DMA */
```

If you change the VME/374 to respond as an A24 slave, you may have to change vme_am to 0x3d to cause the VME/374 DMA to use A24 super access.

Other Values in the Descriptor
The only other value that you may adjust is the maximum number of mbufs to allocate for DMA receive. In DMA mode, the driver pre-allocates mbufs and the VME/374 firmware copies the received Ethernet frame directly into the mbuf. This further speeds the received packet processing speed. Each mbuf reserved for DMA consumes about 1600 bytes of mbuf allocation. Therefore, be sure that the SysMbuf allocation is sufficient to handle the amount of mbuf you wish to use.

```
dc.w  u_short rcv_max;     /* max recv mbufs to use for DMA */
```

Notes to CMC/ENP101 Users

The enp10i driver is known to work with the CMC enp10i Ethernet Node Processor. This card must be using the K1 kernel version 4.7. The ROMs on the enp10i should read:

E10LLD 4.0
KI 4.7

CMC enp10i Jumper Settings
Check all the jumpers on the enp10i to make sure they are appropriate for your system configuration. The factory jumper settings should be satisfactory for most system.

The port address for the enp10i is shipped as a slave in A24 or VME standard address space at location 0xde0000. The supplied descriptor is set up for address 0xde1000 due to address considerations for the enp10i. If the address changes, this must be reflected within the eni0 descriptor. If jumpered to A32 or VME extended address space, you must adjust the "0d" address modifier within the eni0.d file. The port address change can be reflected within the if_devices file. Refer to the section "Making Device Descriptors for CMC and ENPLLD Systems" for making the eni0 descriptor.

With many newer VME CPU boards, specific address regions within their memory map reflect A24 or standard VME address space. Make sure the port address within the descriptor reflects this properly.

To verify the port address, use a debugger (either rombug, sysdbg, debug, or srcdbg) to access the card. The following is a dump of the card using a debugger:

```
debug: d4 de1000
0x00DE1000 – 00000004 00F809E2 00F80198 00F70E00 ...F.x.b.x...w‘.
0x00DE1010 – 00000000 00000000 00000000 00000000 ................
0x00DE1020 – 00000000 00000000 00000000 00000000 ................
0x00DE1030 – 00000000 00000000 00000000 00000000 ................
```

Verify that the first long word is 00000004. This verifies the existance of the enp10i card.

Changing the enp10i's memory verifies the address modifiers within the eni0 descriptor. Try changing the memory using the following debugger commands:

```
debug: cw de1020                    16 bit read/write test
0x00de1020 : 0000 .
debug: cl de1020                    32 bit read/write test
0x00de1020 : 00000000 .
```

For the above change memory tests, the 16-bit access is the only one required to work, for the driver will only do 16-bit accesses to the enp10i board. If the 16-bit access test is successful (that is, no bus errors occur), the drivers can access the enp10i.

DMA Considerations
As shipped, the driver will perform DMA. The enp10i driver does not translate the host addresses to the bus addresses, they must be identical. The enp10i card itself will only DMA in A24 or VME standard address space. Verify that memory is available and that the enp10i can access this memory.

Installing Internet

To install OS-9/Internet, obtain the following information from your network administrator:

- this node’s name, Internet address, and broadcast address
- names and Internet addresses of nodes and networks with which this node will communicate

On some OS-9 DevPaks, ISP is included. If your system already has an ISP directory, skip to Step 3.

1. Create an ISP Directory on the Hard Drive. Use the makdir utility to create an ISP directory:

```
makdir /h0/ISP
```

2. Copy the Internet Software from the Floppy Diskettes to your Hard Drive. Use the dsave utility to copy the Internet floppy diskettes into the ISP directory. Use a buffer size appropriate for your installation.

```
chd /d0
dsave –eb=20 /h0/ISP
```

The buffer size of 20 is used as an example

3. Edit the if_devices File. Change your current data directory to /h0/ISP/DRIVERS:

```
chd /h0/isp/drivers
```

Edit the if_devices file to include your Internet and broadcast addresses. Also, verify the I/O address and interrupt vector in the if_devices file.

Refer to the appropriate section on making device descriptors (Making Device Descriptors for the VME/147 and VME/167 Systems or Making Device Descriptors for CMC and ENPLLD Systems) for more information.

Important: On most systems, especially DevPaks with ISP included, the only items to change in this file are the Internet and broadcast addresses.

4. Edit the hosts File. Change your current data directory to /h0/ISP/ETC:

```
chd /h0/ISP/ETC
```

Edit the hosts file to include this host’s name and Internet address, as well as other host’s names and Internet addresses.

5. Edit the networks File. Edit the networks file to include all networks with which this node needs to communicate. Save the networks file.

6. Create the inetdb Data Module

The idbgen utility creates a data mocule named inetdb from the hosts, networks, protocols, and services files. The output data module is left in the current directory. It must be resident in the module directory for the Internet utilities to work. This allows embedded, diskless systems to use Internet and have access to the information contained in the four files.

Load idbgen if it is not currently in memory:

```
load –d /h0/ISP/CMDS/idgben
```

Run idbgen:

```
idbgen
```

7. Edit the socket.a File

Change your current data directory to /h0/ISP/SOCKDESC:

```
chd /h0/ISP/SOCKDESC
```

Edit the socket.a file. Go to the end of the file and replace the string myhostname in the net_name field with your host name:

```
net_name dc.b “myhostname”,0,0,0,0,0,0,0,0,0
```

Remake the socket descriptor using the supplied makefile:

```
make –u socket
```

8. Edit the ipconfig.a File

Change your current data directory to /h0/ISP/IPCONFIG:

```
chd /h0/ISP/IPCONFIG
```

Edit the ipconfig.a file. It should be similar to the following:

```
* Internet Configuration
 dc.l 0                          flags
* 0x0001                         gateway flag
 dc.b 0,0,0,0                    default Internet destination
 dc.w host_rts
 dc.w net_rts
 align

host_rts
 dc.b 0,0,0,0                    must terminate routing lists
 dc.b 0,0,0,0

net_rts
 dc.b 0,0,0,0                    must terminate routing lists
 dc.b 0,0,0,0
```

The default Internet destination field is the IP address of the network's gateway. If you have a stand-alone network, verify that this field is set to the following:

```
dc.b 0,0,0,0
```

If this system is a gateway, you can set up static routing tables in the host_rts and net_rts fields. If you are running routed, you do not need to set the gateway flag.

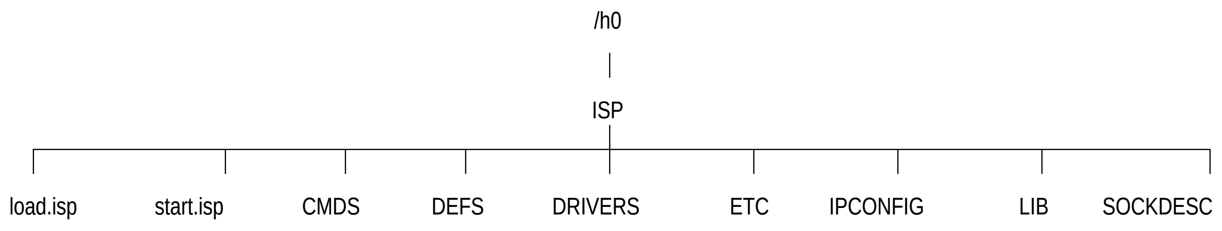
Save the ipconfig.a file.

Remake the ipconfig module using the supplied makefile:

```
make –u ipconfig
```

Important: Refer to the section entitled The Ipconfig Data Module if you need more information.

Important: The files in the distribution package assume the following file and directory organization. They will not assemble and link correctly if the organization is different.



Making Device Descriptors for VME/147 and VME/167 Systems

The device descriptor contains the parameters that affect the operation of the drivers. The ifgen utility uses the if_devices file as a template for creating the device descriptor. The driver is shipped with all the parameters properly set for the VME/147 or the VME/167, except the Internet address, which is site specific.

For the VME/147, the device descriptor is le0_147 and the driver is am7990.

For the VME/167, the device descriptor is ie0_ie167 and the driver is i82596.

If you have an existing IP Ethernet network, the administrator of that system can help you choose an Internet address that is appropriate for your site. If you are initially setting up a network between just a few systems, you can simply use the network number example that already appears in the descriptor. The example Internet address is a ***class C*** address.

Important: Never assign a host number with all zeros or ones (binary) as this is reserved for broadcast use.

The bdaddr field must be set to the Internet address that is recognized as broadcast for this host. Usually this is a matter of substituting the inetaddr with 255 or 0 as the host, as shown in the following examples of the if_devices file.

For the VME/147:

```
le0_147 uses am7990 at 0xFFFE1800 vector 68 level 5 poll
0 mtu 1500\
submask 0\
flags notrailers, broadcast\
inetaddr      192.52.109.1 \
bdaddr        192.52.109.255
```

For the VME/167:

```
ie0  uses ie167 at 0xFFF46000 vector 0x67 level 5 poll 0
mtu 1500\
submask 0\
flags notrailers, broadcast\
inetaddr      192.52.109.1 \
bdaddr        192.52.109.255
```

The if_devices file is located in the ISP/DRIVERS directory. You must remake the device descriptors to include the Internet address for the system. The following is a summary of the procedure to make the descriptor:

```
chd  ISP/DRIVERS
umacs if_devices                 or use any editor you choose
load –d /h0/isp/cmds/ifgen
ifgen <if_devices
```

Next, change directories and create the object code for the descriptor.

For the VME/147:

```
chd am7990
make –u
```

For the VME/167:

```
chd ie167
make –u
```

To check if le0 or ie0 have been updated, use one of the following commands:

```
dir –e ../../cmds/le0_147
```

or

```
dir –e ../../cmds/ie0_167
```

Next, add the host’s name and appropriate Internet address to the hosts file, update the networks file, and use the idbgen utility to update inetdb:

```
chd ../../etc
<update hosts>
<update networks>
load –d ../cmds/idbgen
idbgen
```

Use the following command to check that inetdb has been updated:

```
dir –e inetdb
```

Start the Internet system for device packs by executing start.isp:

```
chd ..
start.isp
```

Making Device Descriptors for CMC and ENPLLD Systems

The device descriptor contains the parameters that affect the operation of the drivers. The ifgen utility uses the if_devices file as a template for creating the device descriptor. The driver is shipped with all the parameters properly set for the enp10i or the enp100l, except the Internet address, which is site specific.

For the enp10i, the device descriptor is eni0 and the driver is enp10i.

For the enp100l, the device descriptor is cnp0 and the driver is enp100i.

If you have an existing IP Ethernet network, the administrator of that system can help you choose an Internet address that is appropriate for your site. If you are initially setting up a network between just a few systems, you can simply use the network number example that already appears in the descriptor. The example Internet address is a **class C** address.

Important: Never assign a host a number with all zeros or ones (binary) as this is reserved for broadcast use.

The bdaddr field must be set to the Internet address that is recognized as broadcast for this host. Usually this is a matter of substituting the inetaddr with 255 or 0 as the host, as shown in the following examples of the if_devices file.

To make the device descriptors, first change your current data directory to ISP/drivers:

```
chd isp/drivers
```

Update the if_devices file. Place the appropriate Internet address in the following field:

```
inetaddr xxx.xxx.xxx.xxx
bdaddr   xxx.xxx.xxx.255
```

The last digit of the bdaddr field should be 255 or 0.

Load the ifgen device:

```
load –d ../cmds/ifgen
```

The if_devices file is located in the ISP/DRIVERS directory. You must remake the device descriptors to include the Internet address for the system. The following summarizes the procedure to make the descriptor:

```
ifgen <if_devices
chd ENPLLD/ENP10i                    or ENPLLD/ENP100i
make
```

To check if cnp0 or eni0 have been updated, use one of the following commands:

```
dir –e ../../../cmds/cnp0
```

or

```
dir –e ../../../cmds/eni0
```

Next, add the host’s name and appropriate Internet address to the hosts file, update the networks file, and use the idbgen utility to update inetdb:

```
chd ../../../etc
<update hosts>
<update networks>
load –d ../cmds/idbgen
idbgen
```

Use the following command to check that inetdb has been updated:

```
dir –e inetdb
```

If you are using a 68000 or 68010 CPU, change SYSMbuf_020 to SysMbuf_010 in the load.enp100i, load.enp10i, and load.both files:

```
chd ..
<make changes>
```

Start the Internet system in the following way:

- Execute start.eni if the system is using enp10i.
- Execute start.gate only if system is working as router (or gateway).

The Ipconfig Data Module

`ipconfig` contains routing information for the ip protocol module. The ipconfig, as shipped, works on a network with no gateway access requirements. ISP Version 1.3 (and greater) includes a routed program to automatically modify the routing tables based on broadcasts from other routed programs on gateway machines. Generally, you should run the routed program rather than modifying the ipconfig table entries.

On a system that is not acting as a gateway, routed updates the IP routing tables to allow access to other networks based on the gateway broadcasts.

On a system that is acting as a gateway, routed broadcasts its routing tables on all connected networks. Thus, the network gateway information is determined dynamically from the network, rather than fixed in the static ipconfig table.

Although you do not need to alter the ipconfig module for OS-9/Internet to work, you may wish to alter it, depending on your system.

If you do not run routed and you attempt to communicate with a host on a different network, Internet will refuse to attempt the connection, as it knows of no way to reach that host. You can change this by altering the ipconfig module.

If there is a default host to send all non-routable packets to, change the default host entry in ipconfig.a to that host.

You may specify host-specific routing by adding entries to the host-specific routing table.

You may specify network routing by adding entries to the network routing table.

Important: Host-specific and network routing tables must be terminated with a null entry.

This is the routing information as OS-9/Internet is shipped:

```
* Internet Configuration
 dc.l 0                                flags
* 0x0001                               gateway flag
 dc.b 0,0,0,0                          default Internet destination
 dc.w host_rts
 dc.w net_rts
 align

host_rts
 dc.b 0,0,0,0                          must terminate routing lists
 dc.b 0,0,0,0

net_rts
 dc.b 0,0,0,0                          must terminate routing lists
 dc.b 0,0,0,0
```

For the purpose of the following example, assume your machine is on network 192.9.200.x.

To route all packets destined for networks 192.9.100.x and 192.9.101.x through a host at 192.9.200.34, use network routing.

To route all packets for host 192.8.50.24 through host 192.9.200.33, use host-specific routing.

All other packets should go to your network’s general gateway at 192.9.200.32.

In this example, change your ipconfig.a file as follows:

```
* Internet Configuration
 dc.l 1                              flags
* 0x0001                             gateway flag
 dc.b 192,9,200,32                   default Internet destination
 dc.w host_rts
 dc.w net_rts
 align

host_rts
 dc.b 192,8,50,24                    all packets for this guy go to
 dc.b 192,9,200,33                   this guy

 dc.b 0,0,0,0                        must terminate routing lists
 dc.b 0,0,0,0

net_rts
 dc.b 192,9,100,0                    route this network through here
 dc.b 192,9,200,34

 dc.b 192,9,101,0                    this one too...
 dc.b 192,9,200,34

 dc.b 0,0,0,0                        must terminate routing lists
 dc.b 0,0,0,0
```

Then, remake a new ipconfig using the make utility:

```
chd /h0/ISP/CMDS          change to directory containing ipconfig file
attr –w ipconfig          make it possible to write to ipconfig
chd ../IPCONFIG           change to directory containing ipconfig source
umacs ipconfig.a          make changes
make                      make new ipconfig module
```

Getting Internet Up and Running on VME/147 and VME/167 Systems

Two procedure files are provided as examples of loading and starting the ISP system. Use the test procedure described here the first time you run ISP.

1. Run the load.isp procedure file to load all the modules into memory:

```
$:  chd /h0/isp           or whichever directory you are using for ISP
$:  load.isp
```

2. Initialize the network memory handler. For example:

```
$:  mfree
```

Current total free RAM: 1786.25 K-bytes

```
$:  mbinstall
$:  mfree
```

Current total free RAM: 1658.25 K-bytes

The free memory decrease is due to the allocation to ISP.

3. Run routed (or ispstart) to open a socket and initialize the entire ISP system:

```
$:  routed<>>>/nil&     or ispstart&
$:   procs –e

 Id    PId    Grp.Usr    Prior  MemSiz    Sig S     CPU Time   Age Module & I/O
  2    0      0.0        128    15.50k    0 w        0.04  0:01 tsmon <>>>term
  3    2      0.0        128     4.00k    0 w        0.47  0:01 shell <>>>term
  7    3      0.0        128    20.00k    0 s        0.02  0:00 routed <>>>/nil
  8    0      0.0        128     2.00k    0 e        0.00  0:00 ifman
  9    0      0.0        128     2.00k    0 s        0.00  0:00 sockman
 10    3      0.0        128    18.25k    0 *        0.10  0:00 procs <>>>term
```

Ensure that the routed (or ispstart), ifman, and sockman processes appear.

4. Check the Ethernet hardware.

For VME/147 users, check the Ethernet hardware as follows:

```
$ lestat /le0

this=003f7130  next=00243350  prev=00242eb0  static=003f7cb0 size=00000208
name=le0  driver=am7990  mtu=1500  flags=0022
af=2  port=0  ipaddr=192.9.200.55
Ethernet address = 8:0:3e:20:3f:6f
busy=0  running =1
in=2  out=1  inerr=0  outerr=0  coll=0
unkirq=0  recv=0  irecv=3  fram=0  oflo=0  crc=0  rbuf=0 miss=0  bogus=0
xirq=1  trys=1  xmit=1  more=0  one=0  defer=0  tbuf=0
uflo=0  1col=0  1car=0  retry=0  babl=0  enq=0  tailirq=0  seen=0
```

Ensure that the displayed Ethernet address is reasonable for your system.

After initialization, the driver sends an ARP request for its own address as a test of the driver transmit and receive capability. You can see this attempt in the xmit/ recv counters of the lestat display. Then, telnet to a known, working Internet host. Check the last line of the lestat display for the error/attempt status.

The last three lines of the display provide important information about the health of the driver and LANCE interface. In general:

This line is the receive status:

```
unkirq=0  recv=0  irecv=3  fram=0 oflo=0 crc=0 rbuf =0 miss=0 bogus=0
```

This line is the transmit status:

```
xirq=1  trys=1  xmit=1  more=0  one=0  defer=0  tbuf=0
```

This line is the error status:

```
uflo=0  lcol=0 lcar=0 retry=0 babl=0  enq=0  tailirq=0  seen=0
```

The receive and transmit status lines contain error counters as defined by the LANCE device. These are not errors; they are conditions that arise as a normal part of Ethernet operations. However, if these values increase greatly in value over a short period of time a problem may exist.

The error status line contains counters for unexpected conditions that affect the ability to communicate on the Ethernet.

Unexpected increases of the value in lcar usually indicate a transceiver cable problem. Try reseating or replacing the transceiver cable if this error persists.

Unexpected increases in lcol and babl usually indicate a problem with the Ethernet cable. Check the hardware for open cables, shorts, or bad terminators.

For VME/167 users, check the Ethernet hardware as follows:

```
$ iestat /ie0

this=01eb9830 next=01eba370 prev=01ec2480 static=01eb9460 size=00000208
name=ie0 driver=ie167 mtu=1500 flags=0022
af=2 port=0 ipaddr=192.9.200.40
Ethernet address = 8:0:3e:21:10:c0
in=2744 out=2770 inerr=0 outerr=0 coll=0

unkirq=0 irqs=5517
rirq=2734 irecv=2744 rmiss=0 rloop=5
xirq=2770 trys=2770 enq=0
v_addr=fff46000 v_port=fff46000 v_ca=fff46004 v_irq=fff46000
v_shram=1e03e20 size=0x18910 (100624)
Selftest spinloops=676 res1=6c335394 res2=00000000
SCP=1e03e60 ISCP=1e03e70 SCB=1e03e80 CB=1e03eb0 (32) RFD=1e04330 (32)
TBD=1e10530 (32) next=1e17250 last=1e1c730
SCB cmd=0000 status=0040 t_on=175 t_off=25 CUS:Idle RUS:Ready
cmds issued=2774 cmds complete=2774 scbloops=0 dropped=0
cmds linked=0 cu starts=2774 cu idle=2775 needxirqstart=0 xirqstarts=0

Recv errors: crc=0 align=0 resource=0
             dmaover=0 rcvcoll=0 short=32
Xmit errors: lcar=0 lcol=0 lcts=0 udma=0
Xmit status: mtry=0 defer=106 heartbeat=2736 collisions=131
```

Ensure that the displayed Ethernet and IP addresses are reasonable for your system.

After initialization, the driver sends an ARP request for its own IP address as a test of the driver transmit and receive capability. This is shown in the in/out counters of the iestat display. Refer to the following discussion if any values in the lines marked Recv errors: or Xmit errors: are non-zero. Then, telnet to a known, working Internet host. Check the iestat again for error/attempt results.

The second set of lines are provided for those familiar with the operation of the i82596 and the software driver. These items are pointers to i82596 shared memory data structures, statistical counters, and the last known i82596 chip status.

Some of the values are described here:

- `t_on` and `t_off` i82596 bus throttle ON/OFF timer values. These items (respectively) set the maximum time the i82596 can remain on the local bus and the minimum time it must stay off the local bus.

  On the VME/167, t_on and t_off are determined by sC, where s is the desired time in µs and C is the CPU clock rate in MHz. Thus, for a t_on time of 7µs on a 25MHz CPU, use 175 (7 multiplied by 25).

- `rmiss` indicates the number of packets that the i82596 could not receive because no space was available in the chip receive queue. If rmiss is non-zero, the i82596 is receiving packets at a rate higher than the CPU can process them. Increasing the chip’s receive queue will reduce the packet loss. Change the max_rfd value in the ie0.d device descriptor to adjust the receive queue.

- `dropped` indicates the number of packets that the driver has prepared for the chip to transmit, but were discarded because the chip command queue is full. This usually happens when the i82596 is busy due to packets being received or due to transmit deferral of previous packets. Change the max_cbl value in the ie0.d device descriptor to adjust the command queue.

- `cmds` linked indicates the number of i82596 commands that the driver linked into the chip’s command queue while the command queue was active. If cmds linked is non-zero, the CPU is providing transmit packets at a rate higher than the i82596 can transmit them. This is generally good.

- The cmds issued and cmds complete values must always be the same.

- `needxirqstart` and xirqstarts count the number of times the driver tried to link a packet in the i82596 command queue while the chip was working on the last entry of the queue. The driver xmit irq routine must restart the command queue because the i82596 is about to go idle. The needxirqstart and xirqstarts values must always match.
- `cu starts` and `cu idle` indicate (respectively) the number of times the driver started the command unit and the number of “command unit idle” interrupts issued by the chip. cu starts + 1 must always equal cu idle.
- `lcol`, `lcal`, and `lcts` with non-zero values usually indicate a network hardware problem. Check or replace the transceiver cable, the MAU (media access unit), or backbone cable.

5. If the ISP software seems to be operating from these tests, start Internet by typing the following commands:

`chd /h0/ISP`	change to the ISP directory
`start.isp`	call load.isp

To start Internet on your system automatically when the system is booted, edit the startup file in the root directory as follows:

`chd /h0`	change to the root directory
`umacs startup`	edit startup file

Add the following line to your startup file:

```
chd ISP; start.isp
```

After adding this line, re-boot the system and verify that all OS-9/Internet functions are available to test the new startup file. The Internet system is now ready for startup. Two procedure files are provided to facilitate starting the system:

`load.isp`	load all necessary Internet files
`start.isp`	start Internet system

OS-9/Internet Implementation Notes

OS-9/Internet uses the BSD4.x IPC (interprocess communication) socket facilities. Not all the services or features of the BSD IPC are provided, but most are present. Some facilities cannot be implemented on OS-9. Consequently, alternatives are supplied.

The following items in this implementation are significantly different from BSD IPC:

- Sockets can be bound to the AF_INET domain.
- Only SOCK_STREAM (TCP) and SOCK_DGRAM (UDP) socket types are supported.
- The select() function is difficult to implement on OS-9. The _ss_sevent() function is provided in sockman and pkman to allow a process to wait until data is available on multiple paths.
- Network database functions (such as gethostname()) access the inetdb data module rather than the data files in /etc. This allows the entire implementation to exist in ROM. Changes to these files require recreating the inetdb module and loading the new copy into memory. Otherwise, the functions and their BSD counterparts behave identically.
- The function synopses list the errno values received when the function fails. The errors listed are only the more interesting ones. Be aware that other standard OS9 errors can be returned such as E_PERMIT, E_PTHFUL, and E_MEMFUL.
- You can use normal OS-9 read() and write() functions to perform I/O on stream sockets. However, the bytecount given for read() is the maximum number of bytes to read during the call, not the number of bytes to return (as defined by other OS9 file managers). For example, if 24 bytes of data are received, read(p,buf,100) returns the 24 bytes instead of waiting for 76 more bytes. This is consistent with UNIX I/O programming and makes such applications easier to port.
- `sockman` does not support the readln() and writeln() functions.
- Non-blocking I/O and out-of-band data are not yet fully supported

Chapter 3

Transferring Files with FTP

OS-9/Internet allows you to copy files from one system to another using the ftp utility. `ftp` is the user interface to the ARPANET standard File Transfer Protocol. You can use it to log into a remote system, look at the directories on the remote machine, and transfer files to and from a remote network site.

This chapter covers the following topics:

- connecting to a remote host with ftp
- file naming conventions
- available ftp commands
- locating files on the remote host
- copying files from the remote host to your local system
- copying files from your local system to the remote host
- exiting ftp

More information on ftp is provided in the utility description in Chapter 6, OS-9/Internet Utilities.

Connecting to a remote Host with FTP

The syntax for ftp is as follows:

```
ftp [<host>] [<opts>]
```

`host` is the network site (remote host) to which you want to connect. When you enter host on the command line, ftp immediately attempts to establish a connection to an FTP server on that host. For example, if you want to establish connection with a remote site called iggy, enter the following command:

```
$ ftp iggy
```

After pressing **[Return]**, you would see something similar to the following:

```
Connected to iggy.
220 iggy OS-9 ftp server V1.0 ready
Name (iggy:ellen):
Name (iggy:ellen): ellen
Password (iggy:ellen): <<not shown for security>>
331 password required for ellen
230 user ellen logged in
Connected to iggy.
Mode: stream       Type: ascii        Form: non-print    Structure: file
Verbose: on        Bell: off          Prompting: on      Globbing: on
Hash mark printing: off               Use of PORT commands: on
ftp>
```

You are now in the command interpreter. Notice the last four lines. These lines are a **current status report** and the ftp prompt (`ftp>`). The status report displays information about the current ftp modes:

Mode:	**Description:**
Mode	Specifies the transfer mode. Currently, stream is the only valid mode.
Type	Specifies the representation type. The representation type tells ftp the type of information with which it will deal. The following values are valid: ascii — Specifies network ASCII. This is the default type. binary — Specifies image data.
Form	Specifies the file transfer form. Currently, non-print is the only valid Form.
Structure	Specifies the file structure. Currently, file is the only valid Structure.
Verbose	When this option is on, all responses from the FTP server are displayed. In addition, when a file transfer completes, statistics regarding the efficiency of the transfer are reported. When this option is off, the information is not displayed.
Bell	When this option is on, a bell announces the completion of file transfers. When off, ftp is silent when the file transfer completes.
Prompt	When this option is on, you receive prompts during multiple file transfers to selectively retrieve or store files. When this option is off, you do not receive prompts during multiple file transfers.
Glob	Globbing refers to the expansion of wildcards for remote file names. If this option is on, wildcards used in remote file names are expanded for commands such as mdelete, mget, and mput. If globbing is off, file names are taken literally.
Hash mark printing	Specifies whether a hash mark (#) should be printed for each buffer transferred.

If you do not enter the host on the command line, ftp enters its command interpreter and displays the current status report of the ftp modes. This report is the same as if you had entered a host name.

To connect to a host from the command interpreter, you need to use the open command. For example:

```
ftp> open iggy
Connected to iggy.
220 iggy OS–9 ftp server V1.0 ready
Name (iggy:ellen): ellen
Password (iggy:ellen ):
331 password required for ellen
230 user ellen logged in
ftp>
```

File Naming Conventions

Before transferring files with ftp, you should be aware of the file naming conventions. `ftp` processes the local files that are specified as parameters to ftp commands according to the following rules:

- If the first character of the file name is an exclamation point (!), the remainder of the parameter is interpreted as a shell command. ftp forks a shell with the parameter supplied and reads (writes) from the standard output (standard input) of that shell. If the shell command includes spaces, you must quote the parameter.
- Failing these checks, if globbing (wildcard expansion) is enabled, local file names are expanded.

Important: Remote file names are not processed. They are passed just as they are typed, except for the mdelete, mdir, mget, and mls commands. For these commands, the remote file names are expanded according to the rules of the remote host's server.

Available FTP Commands

The following ftp commands are available. More information about these commands is provided in the ftp utility description in Chapter 6, OS-9/Internet Utilities.

Command:	Description:
$ [<command>]	Runs as a shell command on the local machine.
append [<local> [<remote>]]	Appends local to a file on the remote machine.
ascii	Sets the file transfer type to network ASCII (default).
bell	Announces the completion of file transfers with a bell.
binary	Sets the file transfer type to support binary image transfer.
bye/quit	Terminates the ftp session and exits.
cd/chd	Changes the current data directory on the remote system.
close	Terminates the remote session and returns to the ftp command interpreter.
connect	Connects to a remote ftp.
debug [<value>]	Toggles socket level debugging mode.
delete [<remote>]	Deletes a file on the remote system.
dir/ls [<remote dir> [<local file>]]	Displays a remote directory listing.
form	Sets the file transfer form.
get/recv <remote> [<local>]	Copies remote to the local system.
glob	Toggles wildcard expansion (globbing) of remote files names for mdelete, mget, and mput.
hash	Toggles printing hash marks (#) for each buffer transferred.
help/?	Prints a help message.
lcd/lchd [<directory>]	Changes the current local directory.
mdelete [<remote>]	Deletes multiple files on the remote system.
mdir/mls [<remote dir> <local file>]	Redirects the display of remote directories to a local file.
mget [<remote>]	Copies the remote files to the current local directory.
makdir/mkdir [<remote dir>]	Creates a directory on the remote system.
mode [<name>]	Sets the transfer mode to name.
mput [<local>]	Expands wildcards in the local list and copies each file in the resulting list to the current remote directory.
open [<host> [<port>]]	Establishes a connection to the specified host FTP server.
pd/pwd	Prints the name of the current remote directory pathlist.
prompt	Toggles interactive prompting for commands that involve multiple file transfers.
put/send [<local> [<remote>]]	Copies a local file to the remote machine.
quote [<params>]	Sends the specified parameters to the remote FTP server.
remotehelp/rhelp	Requests help from the remote server.
rename [<old> <new>]	Renames the remote file old to have the name new.
rmdir [<remote dir>]	Deletes a directory on the remote machine.
sendport	Toggles the use of PORT commands.
status	Shows the current status of ftp.
struct [<name>]	Sets the file structure to name.
type [<name>]	Sets the representation type to name.
user [<user name> [<password>] [<account>]]	Identifies yourself to the remote FTP server.
verbose	Toggles verbose mode.

Locating Files on the Remote Host

Once in the command interpreter, you can use any of the available ftp commands. For example, you can move around the directory structure of the remote host, copy files to or from the remote host, change ftp parameters, and make new directories on the remote host.

Because ftp is generally used to transfer files between systems, you need to be able to locate files on the remote host.

You can use either dir or ls to display the directories on the remote host.

For example:

```
ftp> dir
200 PORT command ok
150 Opening data connection for dir –ea  (192.9.200.58,1046) (0 bytes).
                            Directory of . 11:44:44
 Owner    Last modified  Attributes Sector Bytecount Name
-------   -------------  ---------- ------ --------- ----
  1.78    92/04/03 1306   –––wr–wr     104       248 .login
  0.0     92/02/22 2007   d–ewrewr      F4        96 PROJ
  1.78    92/03/12 0841   ––––––wr     124     56984 mbox
  1.78    92/03/12 0840   ––––––wr     10C      9939 srcfile
226 Transfer complete
477 bytes received in 0.19 seconds (2.45 Kbytes/s)
```

To display the directory PROJ, enter **dir proj** at the ftp prompt:

```
ftp> dir proj
200 PORT command ok
150 Opening data connection for dir –ea proj (192.9.200.58,1047) (0 bytes).

                          Directory of proj 11:46:14
 Owner    Last modified  Attributes Sector Bytecount Name
-------   -------------  ---------- ------ --------- ----
  1.78    92/02/22 2007   ––ewrewr      F8       128 mine
226 Transfer complete
234 bytes received in 0.11 seconds (2.08 Kbytes/s)
```

If you want to change your current data directory on the remote system, enter **cd** or **chd** and the directory name. For example:

```
ftp> chd proj
200 CWD command ok
```

Now when you display the current data directory with the dir command, the directory PROJ is displayed. Using cd or chd without parameters returns you to the original directory.

If you lose track of where you are on the remote system, you can enter **pwd** for a Unix system, or **pd** for an OS-9 system:

```
ftp> pd
251 "/h0/DOC" is current directory
```

Copying Files from the Remote Host to Your Local System

The get command allows you to transfer files from the remote host to your local system. For example, to copy the file srcfile to your current directory on your local system, enter:

```
ftp> get srcfile
200 PORT command ok
150 Opening data connection for sbf.driver
(192.9.200.58,1051) (9939 bytes).
setting srcfile to 9939 bytes
226 Transfer complete
10149 bytes received in 0.39 seconds (25.41 Kbytes/s)
ftp>
```

`get` also allows you to give the file a different name on the local system. For example, if you want to call the file srcfile2 on your local system, enter the command:

```
ftp> get srcfile srcfile2
```

You can also use the recv command to copy files from the remote host to your local system. `recv` has been provided for your convenience.

If you want to check that the file was copied to your local system, use the ftp command $ [<command>]. This command allows you to run the command as a shell command. This means that to display your current directory on your local machine, you would enter:

```
ftp> $ dir
                            Directory of . 11:57:12
NOTES           PROGRAMS        TEXT            animate.c       arrayex.c
arrayex.r       aschart.c       aschart.r       balance         balance_sheet
race.c          race.r          race1.c         srcfile2        stpit.c
stpit.r         tour.c          tour.r          train.c         train.r
ftp>
```

Copying Multiple Files

You may copy more than one file at a time using the mget command, and you can use wildcards to specify the files. `ftp` expands the wildcards according to the normal wildcard file rules:

- An asterisk (*) matches any group of zero or more characters.
- A question mark (?) matches any single character.

For example, to copy all the files from the remote host to your local system's current data directory, you would type **`mget *`**. If prompting is on, mget prompts you before copying each file. This allows you to selectively copy only certain files.

```
ftp> mget *
mget mbox? y
mget srcfile? n
mget PROJ/mine? n
ftp>
```

In this example, the file mbox was copied to your local system's current data directory. `srcfile` and PROJ/mine were not copied. Notice that when you specify an asterisk (*), mget also copies files in subdirectories located in the current directory. Therefore, mget may copy more files than you expect.

Copying Files from Your Local System to the Remote Host

The put command allows you to copy files from your local system to a remote system. For example, to copy the aschart.c to the remote system, enter:

```
ftp> put aschart.c
200 PORT command ok
150 Opening data connection for aschart.c
(192.9.200.58,1057).
226 Transfer complete
227 bytes sent in 0.01 seconds (22.17 Kbytes/s)
```

`put` allows you to rename the file that you are copying. For example, if you want the file aschart.c to be called aschartnew.c on the remote system, enter the command:

```
ftp> put aschart.c aschartnew.c
```

Now when you display the remote directory, the file aschartnew.c is displayed:

```
ftp> dir
200 PORT command ok
150 Opening data connection for dir –ea  (192.9.200.58,1058) (0 bytes).
                            Directory of . 10:52:50
 Owner    Last modified  Attributes Sector Bytecount Name
-------   -------------  ---------- ------ --------- ----
  1.78    92/04/03 1306   –––wr–wr     104       248 .login
  0.0     92/02/22 2007   d–ewrewr      F4        96 PROJ
  1.78    92/02/22 1052   ––––––wr   78030       212 aschartnew.c
  1.78    92/03/12 0841   ––––––wr     124     56984 mbox
  1.78    92/03/12 0840   ––––––wr     10C      9939 srcfile
226 Transfer complete
541 bytes received in 0.21 seconds (2.52 Kbytes/s)
ftp>
```

You can also use the send command to copy files from your local system to the remote host. `send` has been provided for your convenience.

Copying Multiple Files

You may copy more than one file at a time using the mput command, and you can use wildcards to specify the files. `ftp` expands the wildcards according to the normal wildcard file rules. For example, to copy all the files from your local system's current data directory that end in .c, you would type **`mput *.c`**. If prompting is on, mput prompts before copying each file. This allows you to selectively copy only certain files.

```
ftp> mput *.c
mput race.c? y
200 PORT command ok
150 Opening data connection for race.c (192.9.200.58,1059).
226 Transfer complete
2993 bytes sent in 0.06 seconds (48.71 Kbytes/s)
mput animate.c? y
200 PORT command ok
150 Opening data connection for animate.c (192.9.200.58,1060).
226 Transfer complete
723 bytes sent in 0.01 seconds (70.61 Kbytes/s)
mput tour.c? n
mput train.c? y
200 PORT command ok
150 Opening data connection for train.c (192.9.200.58,1061).
226 Transfer complete
904 bytes sent in 0.01 seconds (88.28 Kbytes/s)
.
.
.
ftp>
```

In this example, the file tour.c is not copied to the remote system. If prompting had not been turned on, all files ending with .c would have been copied to the remote system.

Exiting FTP

When you finish your ftp session and want to return to your local system, enter the quit command at the prompt:

```
ftp> quit
200 Goodbye
$
```

You are returned to your local system.

The bye command is also available for your convenience. It works the same as quit. Use whichever command is easier for you to remember.

Chapter 4

Using Telnet

Establishing a Socket

You can establish sockets in several ways. The sequence required to establish connected sockets and unconnected sockets are different. Further, connected sockets are established differently on the client and server sides.

By using telnet, you can establish a connection to a login server at another site. telnet uses the TELNET protocol to allow you to login to another machine, called a **remote host**. Once you login to the remote host, your terminal appears to be connected to that machine, and any keys you enter are automatically passed to the remote host.

This chapter covers the following topics:

- beginning a telnet session
- available telnet commands
- capturing information from a telnet session
- ending a telnet session

Beginning a Telnet Session

To establish a connection with a remote host, use the telnet command. The syntax for telnet is as follows:

```
    telnet [<opts>] [<hostname> [<portnum>]]
```

`telnet` supports the following options:

Option:	Description:
-?	Displays the on-line help message.
-d	Turns on socket level debugging.
-o	Shows options processing.

If you call telnet without parameters, you enter telnet's command mode. This is indicated by the prompt:

```
    telnet>
```

In command mode, telnet accepts and executes commands.

If you call telnet with parameters, telnet performs an open command with those parameters.

Once a connection has been opened, telnet enters input mode. In this mode, text typed is sent to the remote host. To issue telnet commands while in input mode, precede them with the telnet escape character, control-right bracket (<control>]).

While in command mode, the normal terminal editing conventions are available.

Available Telnet Commands

The following commands are available in telnet's command mode. More detail concerning each command is available in the telnet utility description in Chapter 6, OS-9/Internet Utilities.

Command:	Description:
capture [<param>]	Captures all I/O of a telnet session to a specified file. Four parameters are currently available: <file> specifies the file in which to write the I/O on turns on capture mode. off turns off capture mode close closes the capture file
close	Closes the current connection and returns to telnet command mode.
display	Displays current telnet operating parameters.
mode	Tries to enter line-by-line or character-at-a-time mode.
open <host>	Opens a connection to the specified host.
quit	Closes any open telnet connections and exits telnet.
send [<chars>]	Transmits special characters to telnet: ao Abort Output ayt 'Are You There' brk Break ec Erase Character el Erase Line escape Current Escape Character ga 'Go Ahead' Sequence ip Interrupt Process nop 'No Operation' synch 'Synch Operation' Command
set <param>	Sets telnet operating parameters by setting local characters to specific telnet character functions. The following parameters are supported: echo Character to toggle local echoing on/off. erase <local> Sets the telnet erase character. flushoutput <char> Sets the telnet flushout character.interrupt <char> Sets the telnet interrupt character.kill <char> Sets the telnet kill character. quit <char> Sets the telnet quit character.
status	Shows the current status of telnet.
toggle <param>	Toggles telnet operating parameters: crmod Toggles the mapping of received carriage returns.localchars Toggles the effects of the set using the set command. debug Toggles debugging mode. netdata Toggles the printing of hexadecimal network data in debugging mode. options Toggles the viewing of option processing in debugging mode.
z/$	Suspends the current telnet session and forks a shell.
? [<command>]	Prints the help display.

Connecting to Another Host from Comand Mode

To connect to a remote host from the command mode, use open:

```
     telnet> open iggy
```

open opens a connection to a specified host. If you do not specify a host, telnet prompts you for the host name. The host name may be the name of a host or an Internet address specified in the dot notation. You can also specify a port number for the telnet connection. If you do not specify a port number, telnet attempts to contact a TELNET server at the default port.

Once you connect to a remote host, you can log into the system as normal.

Displaying the Current Telnet Operating Parameters

Telnet allows you to display the current operating parameters that you have specified. To do this, use the display command:

```
telnet> display
won’t map carriage return on output
won’t recognize certain control characters
won’t turn on socket level debugging
won’t print hexadecimal representation of network traffic
won’t show options processing

[^E]    Echo
[^]]    Escape
[^X]    Erase
[^K]    Flush output
[^Y]    Interrupt
[^T]    Kill
[^R]    Quit

telnet>
```

The output from display shows what parameters are currently operating. To change any of the parameters, use the toggle command to change the first five parameters. The set command changes the remaining parameters.

For example, to view the options processing in debugging mode, you would enter:

```
     telnet> toggle options
     will show options processing

     telnet>
```

After executing this command, the `current operating parameters will look like the following:`

```
telnet> display
won’t map carriage return on output
won’t recognize certain control characters
won’t turn on socket level debugging
won’t print hexadecimal representation of network traffic
will show options processing

[^E]    Echo
[^]]    Escape
[^X]    Erase
[^K]    Flush output
[^Y]    Interrupt
[^T]    Kill
[^R]    Quit

telnet>
```

The set command works in the same manner.

Displaying the Current Status of Telnet

You can also display the current status of a telnet session. To do this, use the status command:

```
     telnet> status
     No connection.
     Escape character is ‘^]’.
     capture closed.

     telnet>
```

This is useful if you cannot remember whether you have any existing connections and to see whether you are capturing the session.

Capturing Information from a Telnet Session

You can capture the I/O of a telnet session with the capture command. When you use capture, you capture the I/O in a specified file. For example, to place the capture in the file newtest, enter the following:

```
     telnet> capture newtest
     capture to file newtest.

     telnet>
```

This command creates and opens a file called newtest and turns on the capture mode. If the specified file already exists, an error is returned.

Once the capture mode is turned on, you can begin your session by issuing an open command to a remote host. You can display directories on your screen, check the processes running on the system, list files to your screen, etc. and your commands and the system’s replies will all be captured in the file on your local system. When you return to your local system, you can list the contents of the captured file:

```
$ list newtest
OS-9/68020 V2.4.x82   Iggy_vme147 - 68030   91/11/08 16:24:00

User name?: ellen
Password:

Process #25 logged on    91/11/08 16:24:03
Welcome!

$ dir
                            Directory of . 16:24:16
ELLEN           WORKSTUFF         mbox            srcfile
$ procs
 Id PId Grp.Usr  Prior  MemSiz Sig S    CPU Time   Age Module & I/O
 25  26   1.78    128    8.00k   0 w        0.45  0:00 shell <>>>pks00
 30  25   1.78    128   24.00k   0 *        0.05  0:00 procs <>>>pks00
$ mdir
    Module Directory at 16:24:30

kernel      syscache    ssm         init        tk147
rtclock     rbf         scsi147     rb5400      d0
rbvccs      h0          scf         sc8x30      term
$: logout
$
```

Ending a Telnet Session

You can capture the I/O of a telnet session with the capture command. When you use capture, you capture the I/O in a specified file. For example, to place the capture in the file newtest, enter the following:

To end a telnet session from the command mode, use the quit command:

```
telnet> quit
$
```

You are returned to your local system.

To end a telnet session from the remote host, simply logout:

```
iggy$: logout
Connection closed by foreign host.
$
```

Again, you are returned to your local system.

Chapter 5

Using the BOOTP Server

OS-9 Bootstrap Protocol (BOOTP)

The OS-9 Bootstrap Protocol (BOOTP) allows you to boot from the network. OS-9 BOOTP clients require a bootp server on the connected network to support the BOOTP protocol as specified in RFC-951 (Croft/Gilmore) and TFTP as specified in RFC-906 (Finlayson). It also requires the server to support the BOOTP Vendor Information Extensions described in RFC-1048 and RFC-1084 (Reynolds).

The OS-9 BootP server is based on the Carnegie Mellon University implementation. Microware does not provide or support the bootp server for UNIX or other operating systems. Contact the University Computer Center at Carnegie Mellon for the availability of the BootP server on other operating systems.

This chapter covers the following topics:

- overview of the BOOTP server
- OS-9 BOOTP server utilities
- setting up the bootptab configuration file

Overview

BOOTP is a client-server protocol. The OS-9 system being booted is the client and is implemented as a standard OS-9 CBOOT boot driver in ROM. The client system makes requests to a server system on the network. The server may or may not be an OS-9 system. The client requests the server to identify the following:

- the client’s Internet (IP) address
- the name (pathlist).
- the size of an OS-9 bootfile

The server subsequently transfers the bootfile across the network back to the client using the TFTP protocol.

The OS-9 ROM boot code starts the OS-9 network boot option (BOOTP) either through the menu selection <le> or automatically with no operator intervention. The client broadcasts the BOOTP request on the network containing the client’s hardware address (Ethernet address) retrieved from SRAM. A server responds with the following information:

- the client’s IP address.
- the server’s IP address
- a path to the bootfile
- the size of the bootfile

The client then sends a TFTP request for its bootfile to the server. The responding server calls the TFTP service to transfer the bootfile to the client. The BOOTP client reads the OS9 bootfile as it is transferred across the network and copies it into local RAM in the same manner as other boot device drivers.

After the file is successfully read in by the client, BOOTP returns to the C booting subsystem to complete the bootstrap and passes control to the OS-9 kernel.

ISP network drivers can use the IP address determined by BOOTP. This eliminates the need for otherwise identical LAN driver device descriptors to carry different IP addresses.

OS-9 BOOTP Server Utilities

A BOOTP server includes the startbootp procedure, a bootptab configuration file, and the following utility programs:

Name:	**Description:**	
bootpd	Responds to BOOTP client requests with BOOTP server responses.	
	bootptest	A simple program to test bootpd server response.
	hostname	Sets and/or prints the hostname string in SOCKMAN.
tftpd	Responds to tftp read requests and forks tftpdc to handle the transfer.	
	tftpdc	Reads a bootfile for a client using the TFTP protocol.

These utilities are discussed in the next chapter. The utility programs are located in the /h0/ISP/CMDS directory, and the startbootp procedure file and bootptab configuration file are located in the /h0/TFTPBOOT directory.

The startbootp procedure starts the OS-9 bootp server and the associated utilities. The following is an example startbootp file:

```
-t
* start BootP servers
setenv PORT /term
setenv PATH /h0/CMDS:/h0/ISP/CMDS
hostname delta
tftpd -D=/h0/TFTPBOOT<>>>/nil &
bootpd /h0/TFTPBOOT/bootptab<>>>/nil&
```

You can call startbootp from the OS-9 startup file, if appropriate, by adding the following lines to your startup file:

```
$ chd /h0/TFTPBOOT
$ startbootp
```

The OS9boot.hostname file should have the public read permissions set for tftpd to access. Use the following command to turn on the public read permissions for the OS9Boot file:

```
attr -pr os9boot.*
```

The OS9 bootpd server is derived from the Version 2.1 bootpd source code. This source code contains the following notice:

```
/*
 * Copyright (c) 1988 by Carnegie Mellon.
 *
 * Permission to use, copy, modify, and distribute this program for any
 * purpose and without fee is hereby granted, provided that this copyright
 * and permission notice appear on all copies and supporting documentation,
 * the name of Carnegie Mellon not be used in advertising or publicity
 * pertaining to distribution of the program without specific prior
 * permission, and notice be given in supporting documentation that copying
 * and distribution is by permission of Carnegie Mellon and Stanford
 * University. Carnegie Mellon makes no representations about the
 * suitability of this software for any purpose. It is provided “as is”
 * without express or implied warranty.
 *
 *
 * Copyright (c) 1986, 1987 Regents of the University of California.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms are permitted
 * provided that this notice is preserved and that due credit is given
 * to the University of California at Berkeley. The name of the University
 * may not be used to endorse or promote products derived from this
 * software without specific prior written permission. This software
 * is provided as is’’ without express or implied warranty.
 */
```

Setting Up the Bootptab Configuration File

When bootpd is first started, it performs the following functions:

- reads a configuration file to build an internal database of clients and desired boot responses for each
- listens for BOOTP boot requests on UDP socket port 67 (bootps)
- checks the file time stamp on the configuration file before processing a boot request. If the file time stamp has changed since the last check, it re-builds the client database

The configuration file has a format similar to that of termcap in which two character case-sensitive tag symbols represent host parameters. These parameter declarations are separated by colons (:). The general format is as follows:

```
hostname:tg=value...:tg=value...:tg=value:
```

hostname is the actual name of a BOOTP client and tg is a two-character tag symbol. Most tags must be followed by an equal sign and a value. Some tags may also appear in a boolean form with no value (:tg:).

`bootpd` recognizes the following tags:

Tag:	Description:
bf	Bootfile
bs	Bootfile size in 512-octet (byte) blocks
ha	Host hardware address
hd	Bootfile home directory
hn	Send hostname
ht	Host hardware type
ip	Host IP address
sm	Host subnet mask
tc	Table continuation (points to similar “template” entry)
vm	Vendor magic cookie selector

There is also a generic tag, Tn, where n is an RFC-1048 vendor field tag number. This can allow you to immediately use future extensions to RFC-1048 without first modifying bootpd. You can represent generic data as either a stream of hexadecimal numbers or as a quoted string of ASCII characters. The length of the generic data is automatically determined and inserted into the proper field(s) of the RFC-1048-style bootp reply.

The ip and sm tags each expect a single IP address. All IP addresses are specified in standard Internet dot notation and may use decimal, octal, or hexadecimal numbers (octal numbers begin with 0, hexadecimal numbers begin with 0x or 0X).

Specifying the Type of Hardware

The ht tag specifies the hardware type code as:

- an unsigned decimal, octal, or hexadecimal integer
- ethernet or ether for 10Mb Ethernet

Specifying the Hardware Address

The ha tag takes a hardware address. The hardware address must be specified in hexadecimal. You may include optional periods and/or a leading 0x for readability. The ha tag must be preceded by the ht tag (either explicitly or implicitly; see tc).

Specifying the Host Name, Home Directory, and Bootfile

The host name, home directory, and bootfile are ASCII strings which may be optionally surrounded by double quotes (“”). The client’s request and the values of the hd and bf symbols determine how the server fills in the bootfile field of the bootp reply packet.

- If the client specifies an absolute path name and that file exists on the server machine, that path name is returned in the reply packet.
- If the file cannot be found, the request is discarded and no reply is sent.
- If the client specifies a relative path name, a full path name is formed by prepending the value of the hd tag and testing for the file’s existence.
- If the hd tag is not supplied in the configuration file or if the resulting bootfile cannot be found, the request is discarded. Because OS-9 BOOTP clients normally supply os9boot as the bootfile name, the relative path name case is used.

Clients which specify null boot files always elicit a reply from the server. The exact reply depends on the hd and bf tags.

- If the bf tag specifies an absolute path name and the file exists, that path name is returned in the reply packet.
- If the hd and bf tags together specify an accessible file, that file name is returned in the reply.
- If a complete file name cannot be determined or the file does not exist, the reply contains a zeroed-out bootfile field.

In each case, existence of the file means that, in addition to actually being present, the file must have its public read access bit set. tftpd requires this to permit the file transfer. Set the hd tag to /h0/TFTPBOOT or to the same directory as given on the tftpd command line.

All file names are first tried as filename.hostname and then simply as filename. This provides for individual per-host bootfiles.

The following table further illustrates the interaction between hd, bf, and the bootfile name received in the BOOTP request:

Homedir specified:	Bootfile specified:	Client's file specification:	Action:
No	No	Null	Send null file name
No	No	Relative	Discard request
No	Yes	Null	Send if absolute else discard request
No	Yes	Relative	Discard request
Yes	No	Null	Send null file name
Yes	No	Relative	Lookup with .host
Yes	Yes	Null	Send home/boot or bootfile
Yes	Yes	Relative	Lookup with .host

Specifying the Size of the Bootfile

The bootfile size, bs, may be either a decimal, octal, or hexadecimal integer specifying the size of the bootfile in 512-octet blocks, or the keyword auto. Specifying auto causes the server to automatically set the bootfile size to the actual size of the named bootfile at each request. Specifying the bs symbol as a boolean has the same effect as specifying auto as its value. OS9 BOOTP clients require bs or bs=auto.

Sending a Host Name

The hn tag is strictly a boolean tag. It does not take the usual equal sign and value. Its presence indicates that the host name should be sent to RFC-1048 clients. bootpd attempts to send the entire host name as it is specified in the configuration file; if this will not fit into the reply packet, the name is truncated to just the host field (up to the first period, if present) and then tried. In no case is an arbitrarily-truncated host name sent. If nothing reasonable will fit, nothing is sent.

Sharing Common Values Between Tags

Often, many host entries share common values for certain tags (such as name servers). Rather than repeatedly specifying these tags, you can list a full specification for one host entry and shared by others using the tc (table continuation) tag. The template entry is often a dummy host which does not actually exist and never sends bootp requests. This feature is similar to the tc feature of termcap for similar terminals.

Important: `bootpd` allows the tc tag symbol to appear anywhere in the host entry, unlike termcap which requires it to be the last tag.

Information explicitly specified for a host always overrides information implied by a tc tag symbol, regardless of its location within the entry. The tc tag may be the host name or IP address of any host entry previously listed in the configuration file.

Sometimes you need to delete a specific tag after it has been inferred with tc. To delete the tag, use the construction tag@. This removes the effect of tag. For example, to completely undo the host directory specification, use :hd@: at an appropriate place in the configuration entry. After removal with @, you can reset a tag using tc.

Blank lines and lines beginning with a pound sign (#) are ignored in the configuration file. Host entries are separated from one another by newlines. You can extend a single host entry over multiple lines if the lines end with a backslash (\). You can also have lines longer than 80 characters.

Tags may appear in any order, with the following exceptions:

- The host name must be the very first field in an entry.
- The hardware type must precede the hardware address.

Individual host entries must not exceed 1024 characters.

An Example Bootptab File

An example /h0/TFTPBOOT/bootptab file follows:

```
# First, we define a global entry which specifies the stuff every host uses.
#
# the bs tag is required for OS-9 BOOTP clients
# bf is set (to anything) to cause the bootfile.hostname lookup action
#
global.dummy:sm=255.255.255.0:hd=/h0/TFTPBOOT:bs:
#
# individual hosts
#
boop:   tc=global.dummy:ht=ethernet:ha=08003E205284:ip=192.52.109.96:
vite:   tc=global.dummy:ht=ethernet:ha=08003e20c300:ip=192.52.109.57:
boesky: tc=global.dummy:ht=ethernet:ha=08003E202eae:ip=192.52.109.61:
```

Chapter 6

OS-9/Internet Utilities

Fourteen utilities are provided for use with the OS-9/Internet software:

Utility:	Description:
arpstat	ARP Table Information arpstat reports or changes the ARP table information.
bootptest	Test Bootpd Server bootptest tests the bootpd server.
ftp	File Transfer Protocol ftp transfers files to and from remote systems. There are many ftp commands for file manipulation between systems.
hostname	Host Name for Bootpd hostname prints or sets the host name for bootpd.
idbgen	Internet Database Generation idbgen builds the Internet data module from the four data files: hosts, networks, protocols, and services. idbgen must be run each time any of these files are updated.
idbdump	Internet Database Display idbdump displays the current entries of the Internet data module.
ifgen	IF Device Descriptor Generation ifgen generates the necessary assembler code for all IF device descriptors provided for in the if_devices file.
iifstat	IF Device Status ifstat generates an IF device status report.
inetstat	Report TCP status inetstat prints information about open TCP sockets and the status of TCP.
ipstat	IP Information ipstat reports or changes IP information.
ispstart	Internet Startup ispstart starts up the Internet system.
lestat	Lance Device Status lestat generates a LANCE device status report.
mbinstall	F$MBuf Installation mbinstall installs the F$MBuf system call.
telnet	Telnet User Interface telnet provides the user interface for communication between systems on the Internet system. telnet provides the ability to log on to remote systems.

Five daemon server programs and three connection handlers are also provided:

Daemon:	Description:
bootpd	Bootp Server Daemon
ftpd	FTP Server Daemon
ftpdc	FTP Server Connection Handler
routed	Route Daemon
telnetd	Telnet Server Daemon
telnetdc	Telnet Server Connection Handler
tftpd	TFTP Server Daemon
tftpdc	TFTP Server Connection Handler

arpstat

Report/Change ARP Table Information

Syntax

```
arpstat [<opt>]
```

Description

`arpstat` reports or changes the ARP (Address Resolution Protocol) table information. By default, all entries in the ARP table are shown. The IP address, Ethernet address, a timer value, and a flag value are given for each entry.

Options

Option:	**Description:**
-?	Displays the description, options, and command syntax for arpstat.
-a	Shows all entries in the ARP table. This is the default.
-d[=]<hostname>	Deletes the entry in the ARP table for <hostname>.
-n	Shows numbers rather than names.
-s[=]<hostname> <physaddr> [temp]	Adds <hostname> with the physical address specified by <physaddr>. temp specifies that the entry should be added temporarily. The entry will be deleted after 20 minutes. If the word temp is not added, the entry will remain until removed with the -d option.

bootpd

Respond to BOOTP Request From Booting System

Syntax

```
bootpd [<opts>] {<configfile>}
```

Description

`bootpd` is the server daemon that handles client BOOTP requests. bootpd must be run as super user.

The -d option causes bootpd to display request activity which is useful to diagnose BOOTP client request problems. Each additional -d (up to three) appearing on the command line gives more debugging messages.

Each time a client request is received, bootpd checks to see if the configfile has been updated since the last request. This allows you to change configfile without restarting bootpd.

`bootpd` is normally run in an ISP startup file as follows:

```
bootpd /h0/etc/bootptab <>>>/nil&
```

`bootpd` looks in inetdb (using getservbyname()) to find the port numbers it should use. Two entries are extracted:

Entry:	Description:
bootps	the bootp server listening port
bootpc	the destination port used to reply to clients

If the port numbers cannot be determined this way, the port numbers are assumed to be 67 for the server and 68 for the client.

Options

Option:	Description:
-?	Displays the syntax, options, and command description of bootpd.
-d	Log debug information to <stderr>.

bootptest

Test for BOOTP Server Response

Syntax

```
     bootptest {<opts>}
```

Description

bootptest sends a BOOTP request to the network and waits for a response from a BOOTP server. If a response is received, bootptest attempts to read the bootfile from the server. bootptest provides a way to test a BOOTP server setup without using an actual diskless client.

The -h, -e, and -n options are not really options as all must appear on the command line. `-h` accepts a name which is converted to an IP address using gethostbyname().

If no name is available for a host, the IP address can be given in dotted decimal notation.

You can broadcast by specifying 0 or 255 as the host portion of the IP network address. This solicits a response from any BOOTP server on the named network.

A real BOOTP client uses all ones (255.255.255.255) for the IP address when it boots because it does not yet know its IP address. An IP address of all ones is received as a broadcast by any IP host with a socket bound to the bootps port (UDP 67). `bootpd` uses the contents of the BOOTP message to tell from where the broadcast came.

The bootptab configuration file on the bootpd server must specify an entry for the system on which bootptest is running. bootptest cannot do a proxy test for another host because the bootpd server directs the BOOTP response to the intended client’s IP address, not the IP address from which bootptest is running.

The most useful test is a simple assurance test that bootpd is properly running on the server system. Run bootptest naming loopback or the host’s own hostname and see if a response is received from bootpd. Use the -d option in bootpd to display log messages.

Options

Option:	Description:
-?	Displays the syntax, options, and command description for bootptest.
-e=<etheradr>	In colon notation.
-f=<filename>	Copy bootfile into <filename>.
-h=<hostname>	Target server IP address (name or dotted decimal).
-n=<filename>	Bootfile name for bootp server.

Example

The following is an example of bootptest:

```
bootptest -h=192.52.109.255 –e=8:0:3E:20:52:84 –n=os9boot
```

ftp

File Transfer Manipulation/Remote Internet Site Communication

Syntax

```
     ftp [<opts>] [<host>]
```

Description

`ftp` is the user interface to the ARPANET standard file transfer protocol. ftp transfers files to and from a remote network site.

If <host> is specified on the command line, ftp immediately attempts to establish a connection to an FTP server on that host.

If <host> is not specified, ftp enters its command interpreter. A current status report of the ftp modes is displayed and ftp waits for instructions. When waiting for commands, ftp displays the prompt `ftp`. For example:

```
ftp
Not connected.
Mode: stream   Type: ascii   Form: non–print Structure: file
Verbose: on    Bell: off     Prompting: on   Globbing: on
Hash mark printing: off      Use of PORT commands: off
ftp>
```

Important: These fields are discussed in Chapter 3, Transferring Files with FTP.

File Naming Conventions

Local files specified as parameters to ftp commands are processed according to the following rules.

- If the first character of the file name is an exclamation point (!), the remainder of the parameter is interpreted as a shell command. ftp then forks a shell with the supplied parameter and reads (writes) from the standard output (standard input) of that shell. If the shell command includes spaces, the parameter must be quoted; for example, “! ls –lt”. A useful example of this mechanism is: “dir !more”.
- Failing these checks, if **globbing** (wildcard expansion) is enabled, local file names are expanded.

Important: `ftp` does not process remote file names. They are passed just as they are typed, except for the mdelete, mdir, mget, and mls commands. The file names passed to these commands are expanded according to the rules of the remote host's server. Expansion is accomplished by sending ls to the server.

File Transfer Protocols

The FTP specification specifies many parameters which may affect a file transfer. OS-9/ISP currently supports the following type, mode, and structure parameters.

`type` may be:

- ascii specifies network ASCII
- image or binary specify image

Currently, the only mode supported is stream.

Currently, the only structure supported is file.

Commands

Once in the command interpreter, the following ftp commands are available:

Command:	Description:
$ [<command>]	Runs as a shell command on the local machine. If <command> is not specified, it starts an interactive shell.
append [<local> [<remote>]]	Appends <local> to a file on the remote machine. If you do not specify <remote>, the <local> name is used in naming the remote file. If you specify neither <local> nor <remote>, ftp prompts for the appropriate information. File transfer uses the current settings for transfer type, structure, and mode.
ascii	Sets the file transfer type to network ASCII. This is the default.
bell	Announces the completion of file transfers with a bell.
binary	Sets the file transfer type to support binary image transfer.
bye	Terminates the ftp session and exits (same as quit).
cd	Changes the current directory on the remote system (same as chd).
chd	Changes the current directory on the remote system (same as cd).
close	Terminates the remote session and returns to the ftp command interpreter.
connect	Connects to remote ftp.
debug [<value>]	Toggles socket level debugging mode. When debugging is on, ftp prints each command sent to the remote machine, preceded by the string —> NOTE: <value> is currently accepted as a parameter, but is not implemented in this release.delete [<remote>]

Command:	**Description:**
delete [<remote>]	Deletes a file on the remote machine. If <remote> is not specified, ftp prompts for the file name.
dir [<remote> [<local>]]	Displays a remote directory listing. If <remote> is not specified, dir displays the current remote directory. If <local> is specified, dir redirects the directory listing to the specified file instead of to standard output.
form	Sets the file transfer form. non–print is the only form currently supported.
get <remote> [<local>]	Copies the file <remote> to the local system. If <local> is not specified, get copies <remote> to a file with the same name. get uses the current settings for transfer type, structure, and mode while transferring the file. This command is the same as recv.
glob	Toggles globbing (wildcard expansion) of remote file names for mdelete, mget, and mput. If globbing is turned off, file names are taken literally.
hash	Toggles printing # (hash marks) for each buffer transferred.
help [<command>]	Prints a help message about a command if specified (same as ?). If no <command> is specified, help displays a list of available commands.
lcd [<directory>]	Changes the current local directory (same as lchd). If no directory is specified, ftp prompts for the directory name.
lchd [<directory>]	Changes the current local directory (same as lcd). If no directory is specified, ftp prompts for the directory name.
ls [<remote> [<local>]]	Displays an abbreviated directory from the remote system. If <remote> is not specified, ls displays the current remote directory. If <local> is specified, ls redirects the directory listing to the specified file instead of to standard output.
mdelete [<remote>]	Deletes multiple files on the remote system. You can specify <remote> using file name wildcards (for example, ch*.doc).
mdir [<remote> <local>]	Redirects the display of the contents of remote directories to a local file. You can specify <remote> using file name wildcards (for example, chap*.doc). If <local> and <remote> are not specified, ftp prompts for the appropriate information.
mget [<remote>]	Copies the remote files to the current local directory. You can specify <remote> using file name wildcards (for example, chap*.doc).
makdir [<remote>]	Makes a directory on the remote system (same as mkdir). If <remote> is not specified, ftp prompts for the directory name.
mkdir [<remote>]	Makes a directory on the remote system (same as makdir). If <remote> is not specified, ftp prompts for the directory name.
mls [<remote> [<local>]]	Redirects the display of remote file/directory listings to the file <local>. You can specify <remote> using file name wildcards (for example, chap*.doc). If <remote> and <local> are not specified, ftp prompts for the appropriate information.
mode [<mode>]	Sets the transfer mode to <mode>. Currently, only stream is accepted as <mode>.
mput [<local>]	Expands wildcards in the list of <local> and copies each file in the resulting list to the current remote directory.
open [<host> [<port>]]	Establishes a connection to the specified host FTP server. An optional port number may be supplied, in which case ftp attempts to contact an FTP server at that port. If the auto–login option is ON (default), ftp also attempts to automatically log the user in to the FTP server. If <port> is not specified, ftp prompts for the host name.
pd	Prints the name of the current remote directory pathlist (same as pwd).

Command:	Description:
prompt	Toggles interactive prompting for commands that involve multiple file transfers. Interactive prompting occurs during multiple file transfers to allow you to selectively retrieve or store files. By default, prompting is turned ON. If prompting is turned off, any mget or mput transfers all files, and any mdelete deletes all files.
put [<local> [<remote>]]	Copies a local file to the remote machine. If <local> is not specified on the command line, ftp prompts for both the local and remote file names. If <local> is specified on the command line and <remote> is not specified, ftp uses the local file name in naming the remote file. File transfer uses the current settings for transfer type, structure, and mode. This command is the same as send.
pwd	Prints the name of the current remote directory pathlist (same as pd).
quit	Terminates the ftp session and exits (same as bye).
quote [<params>]	Sends the specified parameters, verbatim, to the remote FTP server. A single FTP reply code is expected in return. If no parameters are specified on the command line, ftp prompts for them.
recv <remote> [<local>]	Copies a remote file to the local system. If <local> is not specified, recv copies <remote> to a file with the same name. recv uses the current settings for transfer type, structure, and mode while transferring the file. This command is the same as get.
remotehelp [<command>]	Requests help from the remote FTP server (same as rhelp). If specified, <command> is supplied to the server as well.
rename [<old> <new>]	Renames the remote file <old> to have the name <new>. If <old> and <new> are not specified on the command line, ftp prompts for the appropriate information.
rhelp [<command>]	Requests help from the remote FTP server (same as remotehelp). If specified, <command> is supplied to the server as well.
rmdir [<remote>]	Deletes a directory on the remote machine. If <remote> is not specified on the command line, ftp prompts for the directory name.
send [<local> [<remote>]]	Copies <local> to the remote machine. If <local> is not specified, ftp prompts for both the local and remote file names. If <local> is specified and <remote> is not, send uses the local file name in naming the remote file. File transfer uses the current settings for transfer type, structure, and mode. This command is the same as put.
sendport	Toggles the use of PORT commands. By default, ftp attempts to use a PORT command when establishing a connection for each data transfer. If the PORT command fails, ftp uses the default data port. When PORT commands are disabled, no attempt is made to use PORT commands for each data transfer. This is useful for certain FTP implementations which ignore PORT commands, but incorrectly indicate they have been accepted. NOTE: This command does not have any effect. It is accepted as a legal command, but not implemented. PORT commands are *always* used in the current implementation of ftp.
status	Shows the current status of ftp.
struct [<struct>]	Sets the file structure to <struct>. Currently, the only valid <struct> is file.

Command:	**Description:**
type [<type>]	Sets the representation type to <type>. The valid values for <type> are: ascii for network ASCII. binary or image for image. tenex for local byte size with a byte size of eight (used to talk to TENEX machines). If <type> is not specified, the current type is printed. The default type is network ASCII.
user [<user> [<password>] [<account>]]	Identifies you to the remote FTP server. If <user> is not specified, ftp prompts for it. If the password and/or the account field is not specified and the server requires it, ftp prompts for it after disabling local echo. Unless ftp is called with auto–login disabled, this process is performed automatically on initial connection to the FTP server.
verbose	Toggles verbose mode. In verbose mode, all responses from the FTP server are displayed. In addition, if verbose mode is on, when a file transfer completes, it reports statistics regarding the efficiency of the transfer. By default, if commands are coming from a terminal, verbose mode is ON; otherwise, verbose mode is OFF.
? [<command>]	Prints a help message about a command if specified (same as help). If no <command> is specified, ? displays a list of commands available.

Important: You may use quotation marks (“”) around command parameters having embedded spaces.

Options

You may specify options at the command line or to the command interpreter.

Option:	**Description:**
–?	Displays the description, options, and command syntax for ftp.
–d	Turns on debugging mode.
–g	Does not expand wildcard file name expansion (that is, globbing).
–n	Does not attempt auto–login upon initial connection. If auto–login is enabled, ftp uses the login name on the local machine as the user identity on the remote machine, prompts for a password, and optionally, an account with which to login.
–s	Does not set the file size on received data. By default, ftp attempts to pre–extend the file when getting a file. Often, a remote server includes the file size in the response string when it opens a data connection. ftp recognizes a byte specification with the form (xxxx bytes), if the response code is 150 or 125. If the file size is not included or a different response code is used, ftp does not attempt to pre–extend the file.
–v	Turns off verbose mode. Verbose mode shows all responses from the remote server and reports on data transfer statistics.

ftpd

Incomng FTP Server Daemon

Syntax

```
ftpd [<opt>] <redirections>&
```

Description

`ftpd` is the incoming ftp daemon process. It must be running to handle incoming ftp connection requests. `ftpd` forks the ftpdc communications handler each time a connection to the ftp service is made.

`ftpd` has two available options.

Option:	Description:
-d	Prints debugging information to the standard error path
-l	Prints user login information to the standard error path

To save this information for later use, redirect the standard error path to an appropriate file on the command line:

```
ftpd –d </nil >>>–/h0/ISP/ftpd.debug&
ftpd –l </nil >>>–/h0/ISP/ftpd.login&
```

If neither option is used, redirect the standard error path to the null driver along with the standard input/output paths:

```
ftpd <>>>/nil
```

Important: End the command line with an ampersand (&) to place ftpd in the background (for example, ftpd /nil&). Only supers users can run ftpd.

Options

Option:	Description:
–?	Displays the description, options, and command syntax for ftpd.
–d	Prints debugging information to standard error.
–l	Prints login information to standard error.

ftpdc

FTP Server Communications Handler

Syntax

The ftpd utility must fork this utility.

Description

`ftpdc` is the incoming communications handler for ftp. Each time an ftp service connection is made, ftpd calls this process. Do not fork this process by any other method.

The two ftpd options are passed directly to ftpdc along with the standard error path. Consequently, this process is completely transparent.

hostname

Display or Set Internet Name of Host

Syntax

```
    hostname [<name>]
```

Description

`hostname` prints or sets the string returned by the socket library gethostname() function. By default, gethostname() returns the net_name string appearing in the /socket device descriptor. You can use hostname to override the default to return any string up to 32 characters. bootpd requires gethostname() to return the proper string that names the host on which it is running. If the name is properly set in the socket descriptor, you do not need to run hostname.

ibdgen

Generate Network Database Module

Syntax

```
idbgen [<opts>]
```

Description

`idbgen` generates an OS-9 data module (inetdb) from the four network database files: hosts, networks, protocols, and services.

Any time a change is made to any of these files, you must use idbgen to generate a new data module.

By default, idbgen looks for the network database files in the current directory. However, you may use the –d option to specify the directory containing the files.

Options

Option:	Description:
–d=<str>	Specifies the directory to find the network database files.
–r=<num>	Sets the module revision to <num>.
–x	Places the module in the execution directory.

idbdumb

Display Internet Database Entries

Syntax

```
idbdump [<opts>]
```

Description

`idbdump` displays a formatted listing of the entries of the Internet database currently loaded in memory. If no options are specified, idbdump displays each entry in the database. Specific options display the appropriate type of database entry. For example, the command idbdump –n displays only the network entries:

```
idbdump –n
module id: 1
Network Entries:

loopback     127
sun–ether    192.9.200 sunether ethernet localnet
sun–oldether 125       sunoldether
arpanet 10
```

Options

Option:	Description:
–?	Displays the description, options, and command syntax for idbdump.
–h	Displays the hosts entries.
–n	Displays the networks entries.
–p	Displays the protocols entries.
–s	Displays the services entries.

ifgen

IF Device Descriptor Generation

Syntax

```
ifgen –z=<file>
ifgen <if_devices
```

Description

`ifgen` uses the contents of the if_devices file when creating assembly language descriptor files. You may specify the if_devices file using the -z option or by redirecting ifgen's standard input. The if_devices file contains simple instructions describing the contents of the descriptors.

Device Descriptor Templates

`ifgen` uses templates to create assembly language descriptor files. ifgen replaces the following with the information found in the if_devices file.

Option:	Description:
%A	Internet address
%B	Broadcast/destination address
%D	Directory for device driver
%F	Interface flags
%I	Device descriptor name
%L	Interrupt level
%M	Manual transmission unit of device
%N	Device driver name
%P	Port address
%S	Subnet mask
%T	Polling priority
%V	Interrupt vector
%i	Internet name
%l	Link level address

The following is a sample device descriptor template:

```
*
* Template for ifdgen descriptor generation utility
*
 nam %I device descriptor module

 use ../desc/defsfile

TypeLang set (Devic<<8)+0
Attr_Rev set (ReEnt<<8)+0
 psect %I,TypeLang,Attr_Rev,1,0,0
 dc.l %P port address
 dc.b %V auto–vector trap assignment
 dc.b %L IRQ hardware interrupt level
 dc.b %T exclusive polling table priority
 dc.b Updat_ device mode capabilities
 dc.w FM_name file manager name offset
 dc.w Dvr_name device driver name offset
 dc.w 0 DevCon
 dc.w 0,0,0,0 reserved
 dc.w 2 option byte count

OptTbl dc.b 0x9 DT_INET
 dc.b 0 pad
OptLen equ *–OptTbl

* Internet address
 dc.l bname   ; offset to interface name
 dc.w %M      ; maximum transmission unit
 dc.w %F      ; interface flags
 dc.w sockaddr_in          ; struct sockaddr
 dc.l %S      ; subnet mask
 dc.w bdaddr  ; broadcast/dest address
 dc.w linkaddr             ; link–level (Ethernet) addresss
 dc.l 0
 dc.l 0
 dc.l 0
 dc.l 0
 dc.l 0
*
* driver–specific stuff can be placed here
*
 use %I.d

 align
net_name dc.b “%i”,0
 align
linkaddr dc.b %l
 align
bdaddr dc.b %B
 align
sockaddr_in dc.w 2 AF_INET address family
 dc.w 0 port
 dc.b %A direct address (internet)
socksiz equ *–sockaddr_in
 ifgt 16–socksiz           padd to at least 16 bytes
 rept 16–socksiz
 dc.b 0
 endr
 endc
 align
FM_name dc.b “ifman”,0  file manager
 align
Dvr_name dc.b “%N”,0 device driver
 ends
```

The if_devices File

The if_devices file contains specific entries for each descriptor to make. Each entry consists of a number of fields. Each field is indicated by a leading keyword:

Keyword:	Template Flag:	Function:
inetaddr	%A	Internet address
bdaddr	%B	Broadcast/destination address
dvrdir	%D	Directory for device driver
fkags	%F	Interface flags
name	%I	Device descriptor name
level	%L	Interrupt level
mtu	%M	Manual transmission unit of device
uses	%N	Device driver name
at	%P	Port address
submask	%S	Subnet mask
poll	%T	Polling priority
vector	%V	Interrupt vector
iname	%i	Internet name
laddr	%l	Link level address

Important: The exception to the keyword value entry format is the device descriptor name. If this field appears first in an if_devices entry, you may omit the name keyword.

The value following each keyword replaces the template flag when the descriptor is remade using ifgen.

More than one value may follow the interface flags keyword. The following keywords indicate the values:

Keyword:	Template value:	Function:
notrailers	IFF_NOTRAILERS	Do not accept trailer packets
debug	IFF_DEBUG	Debug mode
broadcast	IFF_BROADCAST	Allow broadcast packets
point	IFF_PT_TO_PT	Point to point link
take_nofwd	IFF_TAKE_NFWD	Accept “no forward” packets
noarp	IFF_NOARP	No ARP packets

You may continue any entry on the following line by placing a backslash (\) at the end of the line to continue. Continue all entries longer than 256 characters on another line.

If a field is not included in the if_devices file, the default value of zero is used for numeric fields.

For example, the following if_devices file describes three descriptors.

```
#
# parameter driver for ifdev device descriptors
#
lo0 uses ifloop at 0 mtu 65535 flags notrailers,take_nofwd \
             inetaddr 127.0.0.1
lo1 uses ifloop at 1 mtu 65535 flags notrailers,take_nofwd \
             inetaddr 126.0.0.1
le0.147 uses am7990 at 0xFFFE1800 vector 68 level 5 poll 0 mtu 1500 \
             flags notrailers,broadcast \
             inetaddr 192.9.200.1
```

To work correctly, ifgen relies on a specific directory structure. The uses field specifies the device driver name and the directory in which to make the device descriptor. The Desc directory must contain the device descriptor template. `Ifgen` must be run from the parent directory of Desc. For example, the sample if_devices file minimally assumes the following directory structure:

```
Desc         Am7990       Ifloop
```

Running ifgen from the parent of these directories creates the desired descriptors in the proper directories.

Options

Option:	Description:
–z=<file>	Specifies the pathlist of the if_devices file to use.

ifstat

IF Device Driver Status

Syntax

```
ifstat <if device>
```

Description

`ifstat` prints information about the status of an IF device.

Options

Option:	Description:
–?	Displays the description, options, and command syntax for ifstat.
–a	Shows information of all interface devices.
–l	Shows information of local interface device /lo0.

Example

```
$ ifstat –l

this=002bcfe0 next=00354070 prev=002b82e0 static=00354040 size=00000208
name=lo0 driver=ifloop mtu=–1 flags=00a0
af=2 port=0 addr=7F000001
nada

$ ifstat –a

ifdev(0).if_next = 002bcfe0
ifdev(0).if_prev = 00354070
ifdev(0).if_mtu = 1500
ifdev(0).if_flags = 0022
ifdev(0).if_subnet = ffffff00
ifdev(0).if_addr = 192.9.200.3
ifdev(0).if_addr.family_type = 2
ifdev(0).if_addr.port = 0
ifdev(0).if_devnam = le0
ifdev(0).if_dvrnam = am7990

ifdev(1).if_next = 00354070
ifdev(1).if_prev = 002b82e0
ifdev(1).if_mtu = –1
ifdev(1).if_flags = 00a0
ifdev(1).if_subnet =          0
ifdev(1).if_addr = 127.0.0.1
ifdev(1).if_addr.family_type = 2
ifdev(1).if_addr.port = 0
ifdev(1).if_devnam = lo0
ifdev(1).if_dvrnam = ifloop
```

inetstat

Report TCP Status

Syntax

```
inetstat
```

Description

`inetstat` prints an Internet status report. By default, the following information is displayed for each active connection:

- the path number
- the protocol used
- the local address
- the foreign address
- the state of the connection
- a queue
- the process ID

Options

Option:	Description:
–?	Displays the description, options, and command syntax for inetstat.
–a	Prints information about all open sockets.
–n	Shows the numbers rather than the Internet names.
–t	Prints TCP statistics.

Example

```
$ inetstat

Active connections
 Path Proto Local Address       Foreign Address     State           Q   Pid
  185 tcp   vite.telnet         yme.1872            established     0     0
    0 tcp   vite.telnet         wahz.3641           time_wait       0     0
   71 tcp   vite.telnet         bobo.1035           established     0     0
```

ipstat

Report/Change IP Information

Syntax

```
ipstat [<opt>]
```

Description

`ipstat` reports or changes IP information.

Options

Option:	Description:
–?	Displays the description, options, and command syntax for ipstat.
–a <–n|–h|–f> <Destination> <gateway>	Adds an entry to the host/net/default routing list.
–c <–n|–h|–f> <Destination> <gateway>	Changes to gateway if entry with matched Destination on host/net routing list.
–d <–n|–h|f> <Destination> <gateway>	Deletes an entry from the host/net/default routing list.
–f	Prints the default information.
–h	Prints the host routing list.
–i	Prints the IP static information.
–n	Prints the net routing list.

Example

```
$ ipstat –i

Ip statistic information:
--------------------------------------------------------
ipm_inpackets(# of packets received) = 602290
ipm_outpackets(# of packets sent) = 647851
ipm_complete(complete datagrams received) = 602286
ipm_badcheck(datagram received with bad checksum) = 0
ipm_runt(tiny datagram that were tossed) = 0
ipm_version(unacceptable version number) = 0
ipm_hlen(header length too short) = 0
ipm_orphan(datagrams that no one wanted) = 0
ipm_route(datagrams routed through this machine) = 0
ipm_qfull(datagrams dropped because recv q was full) = 0
ipm_raw(datagram that went to raw socket) = 0
ipm_ulp(datagrams that went to ulp’s) = 602225
ipm_icmp(icmp datagrams) = 61
ipm_nomem(no memory for processing packet) = 0
ipm_gwflag = 0
--------------------------------------------------------
```

ispstart

Internet Startup

Syntax

```
ispstart&
```

Description

`ispstart` starts the Internet system. There are no options, parameters, or output. An error is returned if ispstart cannot start the Internet system.

lestat

LANCE Driver Status

Syntax

```
lestat <LANCE device>
```

Description

`lestat` prints information about the status of a LANCE device.

Example

```
$ lestat /le0
this=00344180 next=003c9840 prev=003eb6f0 static=00344210 size=00000144
name=le0 mtu=1500 flags=0022
af=2 port=0 ipaddr=192.9.200.57
in=11280 out=5398 inerr=1 outerr=0 coll=0
unkirq=0 recv=11280 irecv=11280 fram=0 oflo=0 crc=0 rbuf=0 miss=1 bogus=0
xirq=5398 trys=5398 xmit=5398 more=5 one=4 defer=53 tbuf=0
uflo=0 lcol=0 lcar=0 retry=0 babl=0 enq=0 tailirq=0 seen=0
```

The last three lines of the display provide important information about the health of the driver and LANCE interface. In general:

- This line is the receive status:

```
unkirq=0 recv=0 irecv=3 fram=0 oflo=0 crc=0 rbuf=0 miss=0 bogus=0
```

- This line is the transmit status:

```
xirq=1  trys=1  xmit=1  more=0  one=0  defer=0  tbuf=0
```

- This line is the error status:

```
uflo=0  lcol=0 lcar=0 retry=0 babl=0  enq=0  tailirq=0  seen=0
```

The receive and transmit status lines contain error counters as defined by the LANCE device. These are not really errors, but conditions that arise as a normal part of Ethernet operations. Only worry about these values if they increase greatly in value over a short period of time.

The last line (error status) contains counters for conditions that are unexpected and affect the ability to communicate on the Ethernet. Unexpected increases of the lcar value usually indicate a transceiver cable problem. Try reseating or replacing the transceiver cable if this error persists. Unexpected increases in lcol and babl usually indicate a problem with the Ethernet cable itself. Check the hardware for open cables, shorts, or bad terminators if this condition persists.

mbinstall

Mbuf Installation Utility

Syntax

```
mbinstall
```

Description

`mbinstall` installs the F$MBuf system call. Before mbinstall executes, you must load the Sysmbuf module in memory.

routed

Routing Daemon

Syntax

```
     routed [<opts>] <redirections>&
```

Description

`routed` is a routing daemon which is supposed to be running in the background at boot time to maintain the routing list.

`routed` uses UDP socket port 520 to listen and for response. About every minute, routed checks to see if more than one Ethernet interface device is active. If more than one is active, routed works as an Internet router and supplies routing information to directly connected networks about every 30 seconds. This routing information, in the RIP data structure, helps the hosts on the directly connected networks to update their routing table. Even though a host has only one active interface device and does not work as a router, routed helps to listen to the routing information being broadcasted on the network and keep the routing list up to date.

About every 90 seconds, routed checks the routing list. If an entry has not been updated for 3 minutes, the entry’s hopcount field is set to infinity and marked for deletion. If the host is a router, this deletion information is broadcasted to let other hosts update their routing table. This delays the actual deletions for about 60 seconds to make sure the delete message was passed throughout the network.

The -d option is for debugging purposes. `routed` supports some debug information when routed –d is running. The debug information is printed on the screen where routed is executed without an ampersand (&) (not in background). `routed` displays the ifdev information currently in iflist, the RIP information routed received, the add, delete, and update routing list information, and the time stamp.

The -n option allows routed not to be a router (gateway) even though more than one Ethernet interface device is active.

`routed` only maintains active entries. Passive entries added manually or received from ipconfig are not changed.

Important: The command line should end in an ampersand (&) in order to place the utility in the background. The super user must run routed.

Options

Option:	Description:
–?	Displays the description, options, and command syntax for routed.
–d	Prints debug information.
–n	Specifies that the daemon does not want to be a gateway.

telnet

Provide Internet Communication Interface

Syntax

```
    telnet [<opts>] [<hostname> [<portnum>]]
```

Description

`telnet` communicates with another host using the TELNET protocol. If you execute telnet without parameters, you enter command mode. This is indicated by the prompt `telnet`. In this mode, telnet accepts and executes the commands listed below. If executed with parameters, telnet performs an open command (see below) with those parameters.

Once a connection is opened, telnet enters input mode. In this mode, typed text is sent to the remote host. To issue telnet commands from input mode, precede them with the telnet escape character. The escape character is initially set to control-right-bracket (^]), but you can redefine it. In command mode, normal terminal editing conventions are available.

Commands

The following commands are available. You only need to type enough of each command to uniquely identify it.

Command:	Description:
capture [<param>]	Captures all I/O of a telnet session to a specified file. Currently, capture supports four parameters: <file> Specifies a new file in which to write the I/O of a telnet session. If <file> already exists, capture returns an error. When <file> is specified, capture creates and opens the file, and turns on capture mode. on Turns on capture mode; begins to write I/O to the current specified capture file. off Turns off capture mode; stops writing I/O to the specified capture file. NOTE: This does not close the file. close Closes the capture file.
close	Closes the current connection and returns to telnet command mode.
display	Displays the current telnet operating parameters. See toggle.
mode	Tries to enter line–by–line or character–at–a–time mode.
open <host>	Opens a connection to the specified <host>. If <host> is not specified, telnet prompts for the host name. <host> may be a host name or an Internet address specified in dot notation. If the port number is not specified, telnet attempts to contact a TELNET server at the default port.
quit	Closes any open telnet connection and exits telnet.
send [<chars>]	Transmits special characters to telnet: ao Abort Output ayt ‘Are You There’ brk Break ec Erase Character el Erase Line escape Current Escape Character ga ‘Go Ahead’ Sequence ip Interrupt Process nop ‘No Operation’ synch ‘Synch Operation’ Command
set <param>	Sets telnet operating parameters by setting local characters to specific telnet character functions. Once set, the local character sends the respective character function to the telnet utility. Specify control characters as a caret (^) followed by a single letter. For example, control–X is ^X. set supports the following parameters: echo Sets character to toggle local echoing on and off. erase <local char> Sets the telnet erase character. flushoutput <local char> Sets the telnet flushout character. This character sends an Abort Output. interrupt <local char> Sets the telnet interrupt character. This character sends an Interrupt Process. kill <local char> Sets the telnet kill character. This character sends an Erase Line. quit <local char> Sets the telnet quit character. This character sends a Break.
status	Shows the current telnet status. This includes the connected peer and the state of debugging.

Command:	Description:
toggle <param>	Toggles telnet operating parameters: crmod Toggles the mapping of received carriage returns. When crmod is enabled, any carriage return characters received from the remote host are mapped into a carriage return and a line feed. This mode does not affect the characters you type, only those received. This mode is not very useful, but it is required for some hosts that ask the user to perform local echoing. localchars Toggles the effects of the set using the set command. debug Toggles debugging mode. When debug is on, it opens connections with socket level debugging on. Turning debug on does not affect existing connections. netdata Toggles the printing of hexadecimal network data in debugging mode. options Toggles the viewing of options processing in debugging mode. Displays options sent by telnet as SENT; displays options received from the telnet server as RCVD.
z	Suspends the current telnet session and forks a shell.
$	Suspends the current telnet session and forks a shell.
? [<command>]	Prints the help display. If <command> is specified, telnet prints information about the specified command. Otherwise, it displays a general help message.

Options

Option:	Description:
–?	Displays the description, option, and command syntax for telnet.
–d	Turns on socket level debugging.
–o	Shows options processing.

telnetd

Incoming Telnet Server Daemon

Syntax

```
telnetd [<opts>] <redirections>&
```

Description

`telnetd` is the incoming telnet daemon process. It must be running to handle incoming telnet connection requests. `telnetd` forks the telnetdc communications handler each time a connection to the telnet service is made.

Two options are available.

- -d prints debugging information to the standard error path
- -l prints user login information to the standard error path

To save this information for later use, redirect the standard error path to an appropriate file on the command line:

```
telnetd –d </nil >>>–/h0/ISP/telnetd.debug&
telnetd –l </nil >>>–/h0/ISP/telnetd.log&
```

If neither option is used, redirect the standard error path to the null driver (along with the standard in/out paths):

```
telnetd <>>>/nil
```

Important: End the command line with an ampersand (&) to place the utility in the background. Only super users can run telnetd.

Options

Option:	Description:
–?	Displays the description, options, and command syntax for telnetd.
–d	Prints the debug information to standard error.
– l	Prints the login information to standard error.

telnetdc

Telnet Server Communications Handler

Syntax

The telnetd utility must fork this utility.

Description

`telnetdc` is the incoming communications handler for telnet. Each time a telnet service connection is made, telnetd calls this process. Do not fork this process by any other method.

The two telnetd options are passed directly to telnetdc along with the standard error path. Consequently, it is completely transparent to the user.

tftpd

Respond to tftpd Boot Requests

Syntax

```
tftpd [<opts>] {<dirname> [<opts>]}
```

Description

`tftpd` is the server daemon that handles the client TFTP requests. Once a BOOTP client has received the BOOTP response, it knows the name of its bootfile. The client then issues a TFTP "read file request" back to the same server machine from which it received the BOOTP response. tftpd forks tftpdc to perform the actual file transfer.

`tftpd` in any system is a security problem because the TFTP protocol does not provide any way to validate or restrict a transfer request; there is no login procedure. To provide some level of security, tftpd only transfers files from a single directory. You can specify this directory on the tftpd command line. The default is /h0/TFTPBOOT.

Important: Bootfiles are assumed to be located in /h0/TFTPBOOT or <dirname>.

Options

Option:	Description:
–?	Displays the syntax, options, and command description of bootpd.
–d	Log debug information to <stderr>.
–D	Directory that tftpd is allowed to access.

tftpdc

Respond to tftpd Boot Requests

Syntax

`tftpdc` is called by tftpd.

Description

`tftpdc` is intended to be run only by tftpd and therefore has no command line options.

Chapter 7

Socket/Network C Libraries

The OS-9/Internet Library

The OS-9/Internet library provides functions for retrieving information from the Internet data files (hosts, protocols, networks, and services) and for Internet address manipulation. The network data files are accessed from the inetdb data module.

Each data access function links to inetdb and returns a structure pointing to the appropriate entry from the data files. Because the structure contains pointers instead of actual data, your programs should never modify the data returned by the data access functions. Programs must also remain linked to inetdb when using values returned by the functions.

There are two methods of linking to inetdb:

- a call to sethostent(), setnetent(), setservent(), or setprotoent() explicitly links to inetdb
- a call to any of the Internet get functions implicitly links to inetdb

Once a process becomes linked to inetdb, these functions will not re-link (increase the module link count) to the data module. Use the set functions to assure that the module is properly linked when necessary.

To unlink a process from inetdb, use one of the end functions.

For example, the following program accesses the data module and prints the host entries:

```
listhosts()
{
    struct hostent *host

    sethostent();          /* link to inetdb */

    while (host = gethostent()) {
        print_host_entry(host) }

    endhostent();          /* unlink from inetdb */
}
```

The following functions comprise the netdb.l library:

endhostent()	endnetent()	endprotoent()	endservent()
gethostent()	gethostbyaddr()	gethostbyname()	getnetent()
getnetbyaddr()	getnetbyname()	getprotoent()	getprotobyn
me()	getprotobynumber()	getservent()	getservbyna
e()	getservbyport()	inet_addr()	inet_lnaof()
inet_makeaddr()	inetof()	inet_network()	inet_ntoa()
sethostent()	setpeerent()	setprotoent()	setservent()

The OS-9/Internet Socket Library

The socket library provides a BSD4.3-like socket interface. The following functions comprise the socklib.l library:

_ss_sevent()	accept()	bind()	connect()
gethostname()	getpeername()	getsockname()	getsockopt()
listen()	recv()	recvfrom()	send()
sendto()	setsockopt()	shutdown()	socket()

_ss_sevent

Set Event on Socket

Syntax

```
_ss_sevent(path, io_event)
int path,        /* open path */
    io_event;  /* event ID returned from _ev_link() or _ev_creat() */
```

Description

`_ss_sevent()` registers an event with the appropriate driver. The event is incremented when the driver has data ready.

`path` is a path opened to a socket (sockman), a pseudo keyboard master, or slave device (pkman).

`io`_event is the event ID returned from a call to either _ev_link() or _ev_creat().

You can issue a _ss_sevent() on multiple paths to detect the presence of data ready on multiple paths at once. The driver executes _ev_setr(io_event, 1, 0x8000) when data is available for reading. A process executes _ev_wait(io_event,1,32767) to wait for the data.

`_ss_sevent()` returns -1 if the driver cannot register the event.

Examples

```
if ((io_event = _ev_creat(0,–1,1,“myevent”) == –1) {
    fprintf(stderr, “can’t create event\n”);
    exit(errno);
}
if (_ss_sevent(net_path,io_event) == –1 ||
_ss_sevent(pkb_path,io_event) == –1) {
     fprintf(stderr, “can’t do _ss_sevent()\n”);
     exit(errno);
}

while (1) {
    if (poll_net()) break;
    if (poll_pkb()) break;
    if (_ev_wait(io_event, 1, 32767) == –1) {
        fprintf(stderr,“ev_wait failed\n”);
        exit(errno);
    }
}
```

accept()

Accept Connection on Socket

Syntax

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

int s, ns;
struct sockaddr addr;
int size, addrlen;

size = sizeof(struct sockaddr_in);
ns = accept(s, &addr, &addrlen);
```

Description

`accept()` takes the first connection on the queue of pending connections and creates a new socket with the same properties as socket s. It allocates and returns a new descriptor, ns, for the socket.

`ns` reads and writes data to and from the socket which connected to this socket. It is not used to accept more connections. The original socket remains open for accepting further connections.

`s` is the original socket. It was created by socket(), bound to an address with bind(), and is listening for connections after a listen().

`addr` is a pointer that returns the address of the peer as known to the communications layer. `addr` is returned in AF_INET format.

`addrlen` is a pointer to a value-result parameter used to pass the amount of space pointed to by addr. It returns the actual length in bytes of the address returned in AF_INET format.

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 is present.

If the socket is marked non-blocking and no pending connections are present on the queue, accept() returns an error.

`accept()` is used with connection-based socket types, currently with type SOCK_STREAM.

`accept()` returns -1 on error. If successful, it returns a non-negative integer that is a descriptor for the accepted socket.

Errors

If unsuccessful, accept() may return any of the following errors:

Error:	Description:
E_ILLFNC	The socket must be listening to call accept().
EOPNOTSUPP	The referenced socket type or option is not supported.
EWOULDBLOCK	The socket is non-blocking and no connections are present to be accepted.

See Also

`bind()`, `connect()`, `listen()`, `select()`, and `socket()`

bind()

Bind Name to Socket

Syntax

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

bind(s, name, namelen)
int s;                      /* socket */
struct sockaddr *name;      /* pointer to the socket address */
int namelen;                /* length of assigned name */
```

Description

`bind()` assigns a name to an unnamed socket. When socket() creates a socket, it exists in a name space (address family) but has no assigned name. bind() requests that the name pointed to by name be assigned to the socket.

`s` specifies the path number of the socket.

`name` is a pointer to the socket address.

`namelen` specifies the length of the assigned name.

`bind()` returns 0 if successful. Otherwise, it returns –1 with the appropriate error code placed in the global variable errno.

Errors

If unsuccessful, bind() may return any of the following errors:

Error:	**Description:**
E_BMODE	Not possible to bind this socket.
E_ILLFNC	Socket already bound.
EADDRNOTAVAIL	The specified address (name) is not available on the local machine.
EADDRINUSE	The specified address (name) is already in use.
EINVAL	The socket is already bound to an address (name).
EPERMIT	The requested address (name) is protected, and the current user has inadequate permission to access it.
ESOCKNOSUPPORT	This socket type is not supported.

See Also

`connect()`, `getsockname()`, `listen()`, and `socket()`

connect()

Initiate Connection on Socket

Syntax

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

connect(s, name, namelen)
int s;                /* socket */
struct sockaddr *name; /* pointer to the socket address */
int namelen;          /* length of assigned name */
```

Description

`connect()` connects to a listening socket.

`s` specifies the path number of the socket to connect. If s is of type SOCK_STREAM, connect() attempts to connect to another socket.

`name` is a pointer to the other socket.

`namelen` specifies the length of the assigned name.

A successful connection returns 0. Otherwise, it returns –1 with the appropriate error code in errno.

Errors

If unsuccessful, connect() may return any of the following errors:

Error:	Description:
EADDRNOTAVAIL	The specified address is not available from this machine.
EAFNOSUPPORT	Addresses in the specified address family cannot be used with this socket.
EISCONN	The socket is already connected.
ETIMEDOUT	Connection establishment timed out without establishing a connection.
ECONNREFUSED	The attempt to connect was forcefully rejected.
ENETUNREACH	The network is not reachable from this host.
EADDRINUSE	The address is already in use.
EWOULDBLOCK	The socket is non-blocking and the connection cannot be completed immediately.

See Also

`accept()`, `getsockname()`, `select()`, and `socket()`

endhostent()

Unlink from Network Database

Syntax

```
#include <socket.h>
#include <netdb.h>

endhostent()
```

Description

`endhostent()` indicates that the process is finished using the inetdb data module. The link count of inetdb is decremented.

See Also

`gethostent()`, `gethostbyaddr()`, `gethostbyname()`, and `sethostent()`

endnetent()

Unlink from Network Database

Syntax

```
#include <socket.h>
#include <netdb.h>

endnetent()
```

Description

`endnetent()` indicates that the process is finished using the inetdb data module. The link count of inetdb is decremented.

See Also

`getprotoent()`, `getprotobyaddr()`, `getprotobyname()`,
and `setprotoent()`

endprotoent()

Unlink from Network Database

Syntax

```
#include <socket.h>
#include <netdb.h>

endprotoent()
```

Description

`endprotoent()` indicates that the process is finished using the inetdb data module. The link count of inetdb is decremented.

See Also

`getprotoent()`, `getprotobyaddr()`, `getprotobyname()`, and `setprotoent()`

endservent()

Unlink from Network Database

Syntax

```
#include <socket.h>
#include <netdb.h>

endservent()
```

Description

`endservent()` indicates that the process is finished using the inetdb data module. The link count of inetdb is decremented.

See Also

`getservent()`, `getservbyaddr()`, `getservbyname()`, and `setservent()`

gethostbyaddr()

Get Network Host Entry

Syntax

```
#include <socket.h>
#include <netdb.h>

struct hostent *gethostbyaddr(addr, len, type)
char *addr;                /* pointer to address of host to get */
int len,                   /* length of address */
    type;                  /* type of address */
```

Description

`gethostbyaddr()` sequentially searches from the beginning of the hosts entries of inetdb until a matching host address is found, or until EOF is encountered. Host addresses are supplied in network order. A null pointer (0) returns on EOF or error.

`addr` is a pointer to the Internet address of the host to get.

`len` specifies the length of the address in bytes.

`type` specifies the AF_INET address type.

`gethostbyaddr()` returns a pointer to a structure containing the fields of a hosts entry in the inetdb data module. hostent has the following structure:

```
struct   hostent {
    char *h_name;          /* official name of host */
    char **h_aliases;      /* alias list */
    int  h_addrtype;       /* address type */
    int  h_length;         /* length of address */
    char *h_addr;          /* address */
};
```

The fields are defined as:

Name:	Description:
h_name	A pointer to the official name of the host.
h_aliases	A pointer to a pointer to a zero terminated array of alternate names for the host.
h_addrtype	The type of address being returned. Currently, this is AF_INET.
h_length	The address length in bytes.
h_addr	A pointer to the network address for the host. Host addresses are returned in network byte order.

Important: `gethostbyaddr()` implicitly links to inetdb, if the calling process has not previously linked to the data module.

Important: All information is contained in a static area. You must copy the information to save it. Only the Internet address format is understood.

See Also

`endhostent()`, `gethostbyname()`, `gethostent()`, and `sethostent()`

gethostbyname()

Get Network Host Entry

Syntax

```
#include <socket.h>
#include <netdb.h>

struct hostent *gethostbyname(name)
char *name;                /* pointer to the name of the host */
```

Description

`gethostbyname()` sequentially searches from the beginning of the hosts entries of inetdb until a matching host name or alias is found, or until EOF is encountered. A null pointer (0) returns on EOF or error.

`name` is a pointer to the name of the host.

`gethostbyname()` returns a pointer to a structure containing the fields of a hosts entry in the inetdb data module. `hostent` has the following structure:

```
struct hostent {
    char *h_name;          /* official name of host */
    char **h_aliases;      /* alias list */
    int  h_addrtype;       /* address type */
    int  h_length;         /* length of address */
    char *h_addr;          /* address */
};
```

The fields are defined as:

Name:	Description:
h_name	A pointer to the official name of the host.
h_aliases	A pointer to a pointer to a zero terminated array of alternate names for the host.
h_addrtype	The type of address being returned. Currently, this is AF_INET.
h_length	The address length in bytes.
h_addr	A pointer to the network address for the host. Host addresses are returned in network byte order.

Important: `gethostbyname()` implicitly links to inetdb if the calling process has not previously linked to the data module.

Important: All information is contained in a static area. You must copy the information to save it. Only the Internet address format is understood.

See Also

`endhostent()`, `gethostbyaddr()`, `gethostent()`, and `sethostent()`

gethostent()

Get Network Host Entry

Syntax

```
#include <socket.h>
#include <netdb.h>

struct hostent *gethostent()
```

Description

`gethostent()` reads the next host entry from inetdb. gethostent() returns a pointer to a structure containing the fields of a hosts entry in the inetdb data module. hostent has the following structure:

```
struct hostent {
    char *h_name;          /* official name of host */
    char **h_aliases;      /* alias list */
    int  h_addrtype;       /* address type */
    int  h_length;         /* length of address */
    char *h_addr;          /* address */
};
```

The fields are defined as:

Name:	Description:
h_name	A pointer to the official name of the host.
h_aliases	A pointer to a pointer to a zero terminated array of alternate names for the host.
h_addrtype	The type of address being returned. Currently, this is AF_INET.
h_length	The address length in bytes.
h_addr	A pointer to the network address for the host. Host addresses are returned in network byte order.

Important: `gethostent()` implicitly links to inetdb if the calling process has not previously linked to the data module. `sethostent()` explicitly links the calling process to inetdb. `endhostent()` unlinks the calling process from inetdb.

Important: All information is contained in a static area. You must copy the information to save it. Only the Internet address format is understood.

See Also

`endhostent()`, `gethostbyaddr()`, `gethostbyname()`, and `sethostent()`

gethostname()

Get Name of Current Host

Syntax

```
gethostname(name, namelen)
char *name;                /* standard host name */
int  namelen;              /* size of name array */
```

Description

`gethostname()` returns the standard host name for the current processor. The returned name is null-terminated unless insufficient space is provided.

`name` is a pointer to the standard host name.

`namelen` specifies the size of the name array.

If successful, gethostname() returns a value of 0. Otherwise, it returns 1 and places the appropriate error code in the global variable errno.

Important: Host names are limited to 31 characters. `gethostname()` returns the host name from the device descriptor. `sethostname()` currently does nothing.

getnetbyaddr()

Get Network Entry

Syntax

```
#include <netdb.h>

struct netent *getnetbyaddr(net, type)
long net;                   /* net number */
int  type;                  /* net number type */
```

Description

`getnetbyaddr()` sequentially searches from the beginning of the networks entries of inetdb until a matching net address and type is found, or until EOF is encountered. A null pointer (0) returns on EOF or error.

`net` is the network number.

`type` is the network number type.

`getnetbyaddr()` returns a pointer to a structure containing the fields of a networks entry in the inetdb data module. `netent` has the following structure:

```
struct netent {
    char *n_name;           /* official name of net */
    char **n_aliases;       /* alias list */
    int  n_addrtype;        /* net number type */
    long n_net;             /* net number */
};
```

The members of this structure are:

Name:	Description:
n_name	A pointer to the official name of the network.
n_aliases	A pointer to a pointer to a zero terminated list of alternate names for the network.
n_addrtype	The type of the network number returned. Currently, this is only AF_INET.
n_net	The network number. Network numbers are returned in machine byte order.

Important: `getnetbyaddr()` implicitly links to inetdb if the calling process has not previously linked to the data module.

Important: All information is contained in a static area. You must copy the information to save it. Only Internet network numbers are understood.

See Also

`endnetent()`, `getnetbyname()`, `getnetent()`, and `setnetent()`

getnetbyname()

Get Network Entry

Syntax

```
struct netent *getnetbyname(name)
char *name;    /* network name */
```

Description

`getnetbyname()` sequentially searches from the beginning of networks entries of inetdb until a matching name or alias is found, or until EOF is encountered. A null pointer (0) returns on EOF or error.

`name` is a pointer to the network name.

`getnetbyname()` returns a pointer to a structure containing the fields of a networks entry in the inetdb data module. netent has the following structure:

```
struct netent {
    char *n_name;           /* official name of net */
    char **n_aliases;       /* alias list */
    int  n_addrtype;        /* net number type */
    long n_net;             /* net number */
};
```

The members of this structure are:

Name:	Description:
n_name	A pointer to the official name of the network.
n_aliases	A pointer to a pointer to a zero terminated list of alternate names for the network.
n_addrtype	The type of the network number returned. Currently, this is AF_INET.
n_net	The network number. Network numbers are returned in machine byte order.

Important: `getnetbyname()` implicitly links to inetdb if the calling process has not previously linked to the data module.

Important: All information is contained in a static area. You must copy the information to save it. Only Internet network numbers are understood.

See Also

`endnetent()`, `getnetbyaddr()`, `getnetent()`, and `setnetent()`

getnetent()

Get Network Entry

Syntax

```
#include <netdb.h>

struct netent *getnetent()
```

Description

`getnetent()` reads the nextinetdb network entry. It returns a null pointer (0) on EOF or error.

`getnetent()` returns a pointer to a structure containing the fields of a networks entry in the inetdb data module. netent has the following structure:

```
struct netent {
    char *n_name;          /* official name of net */
    char **n_aliases;      /* alias list */
    int  n_addrtype;       /* net number type */
    long n_net;            /* net number */
};
```

The members of this structure are:

Name:	Description:
n_name	A pointer to the official name of the network.
n_aliases	A pointer to a pointer to a zero terminated list of alternate names for the network.
n_addrtype	The type of the network number returned. Currently, this is AF_INET.
n_net	The network number. Network numbers are returned in machine byte order.

Important: `getnetent()` implicitly links to inetdb if the calling process has not previously linked to the data module. `setnetent()` links to inetdb, and endnetent() unlinks from inetdb.

Important: All information is contained in a static area. You must copy the information to save it. Only Internet network numbers are understood.

See Also

`endnetent()`, `getnetbyaddr()`, `getnetbyname()`, and `setnetent()`

getpeername()

Get Name of Connected Peer

Syntax

```
getpeername(s, name, namelen)
int s;                     /* socket */
struct sockaddr *name;     /* pointer to the socket address */
int *namelen;              /* size of name */
```

Description

`getpeername()` returns the name of the remote node (peer) connected to socket s.

`s` specifies the path number of the socket.

`name` is a pointer to the socket address.

You should initialize the namelen pointer to indicate the amount of space pointed to by name. `namelen` returns the actual size of the name returned (in bytes).

If successful, getpeername() returns 0. Otherwise, it returns –1.

If unsuccessful, getpeername() may return the following error:

Error:	**Description:**
ENOTCONN	The socket is not connected.

See Also

`bind()`, `getsockname()`, and `socket()`

getprotobyname()

Get Protocol Entry

Syntax

```
#include <netdb.h>

struct protoent *getprotobyname(name)
char *name;   /* protocol name */
```

Description

`getprotobyname()` sequentially searches from the beginning of the protocols entries of inetdb until it finds a matching protocol name or alias, or until it encounters EOF. `getprotobyname()` returns a null pointer (0) on EOF or error.

`name` is a pointer to the name of the protocol.

`getprotobyname()` returns a pointer to an object containing the fields of a line in the protocols data base. `protoent` has the following structure:

```
struct protoent {
    char *p_name;          /* official name of protocol */
    char **p_aliases;      /* alias list */
    int  p_proto;          /* protocol number */
};
```

The members of this structure are:

Name:	Description:
p_name	A pointer to the official name of the protocol.
p_aliases	A pointer to a pointer to a zero terminated list of alternate names for the protocol.
p_proto	The protocol number.

Important: `getprotobyname()` implicitly links to inetdb if the calling process has not previously linked to the data module.

Important: All information is contained in a static area. You must copy the information to save it. Only the Internet protocols are understood.

See Also

`endprotoent()`, `getprotobynumber()`, `getprotoent()`,
and `setprotoent()`

getprotobynumber()

Get Protocol Entry

Syntax

```
#include <netdb.h>

struct protoent *getprotobynumber(proto)
int proto;                 /* protocol number */
```

Description

`getprotobynumber()` sequentially searches from the beginning of the protocols entries of inetdb until it finds a matching protocol number, or until it encounters EOF. `getprotobynumber()` returns a null pointer (0) on EOF or error.

`proto` specifies the protocol number.

`getprotobynumber()` returns a pointer to an object containing the fields of a line in the protocols data base. `protoent` has the following structure:

```
struct protoent {
    char *p_name;          /* official name of protocol */
    char **p_aliases;      /* alias list */
    int  p_proto;          /* protocol number */
};
```

The members of this structure are:

Name:	Description:
p_name	A pointer to the official name of the protocol.
p_aliases	A pointer to a pointer to a zero terminated list of alternate names for the protocol.
p_proto	The protocol number.

Important: `getprotobynumber()` implicitly links to inetdb if the calling process has not previously linked to the data module.

Important: All information is contained in a static area. You must copy the information to save it. Only the Internet protocols are understood.

See Also

`endprotoent()`, `getprotobyname()`, `getprotoent()`, and `setprotoent()`

getprotoent()

Get Protocol Entry

Syntax

```
#include <netdb.h>

struct protoent *getprotoent()
```

Description

`getprotoent()` reads the next protocols entry of inetdb. It returns a null pointer (0) on EOF or error.

`getprotoent()` returns a pointer to an object containing the fields of a line in the protocols data base. protoent has the following structure:

```
struct protoent {
    char *p_name;          /* official name of protocol
*/
    char **p_aliases;      /* alias list */
    int  p_proto;          /* protocol number */
};
```

The members of this structure are:

Name:	Description:
p_name	A pointer to the official name of the protocol.
p_aliases	A pointer to a pointer to a zero terminated list of alternate names for the protocol.
p_proto	The protocol number.

Important: `getprotoent()` implicitly links to inetdb if the calling process has not previously linked to the data module. `setprotoent()` explicitly links the calling process to inetdb. `endprotoent()` unlinks the calling process from inetdb.

Important: All information is contained in a static area. You must copy the information to save it. Only the Internet protocols are understood.

See Also

`endprotoent()`, `getprotobyname()`, `getprotobynumber()`, and `setprotoent()`

getservbyname()

Get Service Entry

Syntax

```
#include <netdb.h>

struct servent *getservbyname(name, proto)
char *name,                /* name of service */
     *proto;               /* name of protocol */
```

Description

getservbyname() sequentially searches from the beginning of the services entries of inetdb until it finds a matching protocol name or alias, or until it encounters EOF. If a non-null protocol name is supplied, searches must also match the protocol. getservbyname() returns a null pointer (0) on EOF or error.

name is a pointer to the name of the service.

proto is a pointer to the name of the protocol.

Important: getservbyname() implicitly links to inetdb if the calling process has not previously linked to the data module.

getservbyname() returns a pointer to an object containing the fields of a line in the services data base. servent has the following structure:

```
struct servent {
    char *s_name;          /* official name of service */

    char **s_aliases;      /* alias list */
    int  s_port;           /* port service resides at */
    char *s_proto;         /* protocol to use */
};
```

The members of this structure are:

Name:	Description:
s_name	A pointer to the official name of the service.
s_aliases	A pointer to a pointer to a zero terminated list of alternate names for the service.
s_port	The port number at which the service resides. Port numbers are returned in network byte order.
s_proto	A pointer to the name of the protocol to use when contacting the service.

Important: All information is contained in a static area. You must copy the information to save it.

See Also

endservent(), getprotoent(), getservbyport(), getservent(), and setservent()

getservbyport()

Get Service Entry

Syntax

```
#include <netdb.h>

struct servent *getservbyport(port, proto)
int port;                   /* service’s port number */
char *proto;                /* protocol name */
```

Description

`getservbyport()` sequentially searches from the beginning of the services entries of inetdb until a matching protocol port number is found, or until EOF is encountered. If a (non-null) protocol name is supplied, searches must also match the protocol. `getservbyport()` returns a null pointer (0) on EOF or error.

`port` specifies the service’s port number.

`proto` is a pointer to the protocol name.

Important: `getservbyport()` implicitly links to inetdb if the calling process has not previously linked to the data module.

`getservbyport()` returns a pointer to an object containing the fields of a line in the services data base. `servent` has the following structure:

```
struct servent {
    char *s_name;          /* official name of service */

    char **s_aliases;      /* alias list */
    int  s_port;           /* port service resides at */
    char *s_proto;         /* protocol to use */
};
```

The members of this structure are:

Name:	Description:
s_name	A pointer to the official name of the service.
s_aliases	A pointer to a pointer to a zero terminated list of alternate names for the service.
s_port	The port number at which the service resides. Port numbers are returned in network byte order.
s_proto	A pointer to the name of the protocol to use when contacting the service.

Important: All information is contained in a static area. You must copy the information to save it.

See Also

`endservent()`, `getprotoent()`, `getservbyname()`, `getservent()`, and `setservent()`

getservent()

Get Service Entry

Syntax

```
#include <netdb.h>

struct servent *getservent()
```

Description

`getservent()` reads the next services entry of inetdb. It returns a null pointer (0) on EOF or error.

`getservent()` returns a pointer to an object containing the fields of a line in the services data base. `servent` has the following structure:

```
struct servent {
    char *s_name;          /* official name of service */

    char **s_aliases;      /* alias list */
    int  s_port;           /* port service resides at */
    char *s_proto;         /* protocol to use */
};
```

The members of this structure are:

Name:	Description:
s_name	A pointer to the official name of the service.
s_aliases	A pointer to a pointer to a zero terminated list of alternate names for the service.
s_port	The port number at which the service resides. Port numbers are returned in network byte order.
s_proto	A pointer to the name of the protocol to use when contacting the service.

Important: `getservent()` implicitly links to inetdb if the calling process has not previously linked to the data module. `setservent()` links the calling process to inetdb. `endservent()` unlinks the calling process from inetdb.

Important: All information is contained in a static area. You must copy the information to save it.

See Also

`endservent()`, `getprotoent()`, `getservbyname()`, `getservbyport()`, and `setservent()`

getsockname()

Get Socket Name

Syntax

```
getsockname(s, name, namelen)
int s;                  /* socket */
struct sockaddr *name;  /* pointer to the socket address
*/
int *namelen;           /* length of name */
```

Description

`getsockname()` returns the current local node name for the specified socket.

`s` specifies the path number of the socket.

name is a pointer to the socket address.

You should initialize the namelen pointer to indicate the amount of space pointed to by name. On return, it contains the actual size of the name returned (in bytes).

`getsockname()` returns zero if the call succeeds. Otherwise, it returns –1.

If unsuccessful, getsockname() may return the following error:

Error:	Description:
ENOBUFS	Insufficient resources are available in the system to perform the operation.

See Also

`bind()`, `getpeername()`, and `socket()`

getsockopt()

Get Socket Options

Syntax

```
     #include <socket.h>

getsockopt(s, level, optname, optval, optlen)
int  s,                   /* socket */
     level,               /* options level */
     optname;             /* options name */
char *optval;             /* buffer for requested option */
int  *optlen;             /* value result parameter */
```

Description

`getsockopt()` returns options associated with a socket. Options may exist at multiple protocol levels, but they are always present at the uppermost socket level.

`s` specifies the path number of the socket.

`level` specifies the options level. When getting socket options, you must specify the level at which the option resides and the option name.

- To get options at the socket level, specify level as SOL_SOCKET.
- To get options at any other level, supply the protocol number of the appropriate protocol controlling the option.

For example, to indicate that the TCP protocol will interpret an option, set level to the protocol number of TCP (see getprotoent()).

`optname` specifies the name of the option. optname and any specified options are passed uninterpreted to the appropriate protocol module for interpretation.

`optval` is a pointer to the buffer for the requested option. optval and optlen together identify the buffer in which to return the value for the requested option(s). If no option value is to be supplied or returned, set optval to zero.

`optlen` is a pointer to a value-result parameter. optlen initially contains the size of the buffer pointed to by optval. `optlen` is modified on return to indicate the actual size of the returned value.

`socket.h` contains definitions for socket level options (see socket() or the chapter on sockets). Options at other protocol levels vary in format and name.

If successful, the call returns zero. Otherwise, it returns –1.

If unsuccessful, getsockopt() may return the following error:

Error:	**Description:**
ENOPROTOOPT	The option is unknown.

See Also

`getprotoent()`, `setsockopt()`, and `socket()`

htonl()

Convert 32-Bit Values Between Host and Network Byte Order

Syntax

```
#include <types.h>
#include <in.h>

u_long netlong;            /* network byte order representation */
u_long hostlong;           /* host byte order representation */

netlong = htonl(hostlong);
```

Description

`htonl()` converts 32-bit quantities between host and network byte order. On machines such as the 68000, htonl() is defined as a null macro in the in.h include file.

`netlong` specifies the network byte order representation.

`hostlong` specifies the host byte order representation to convert.

`htonl()` is most often used with Internet addresses and ports as returned by gethostent() and getservent().

See Also

`gethostent()`, `getservent()`, `htons()`, `ntohl()`, and `ntohs()`

htons()

Convert 16-Bit Values Between Host and Network Byte Order

Syntax

```
#include <types.h>
#include <in.h>

u_short netshort;          /* network byte order representation */
u_short hostshort;         /* host byte order representation */

netshort = htons(hostshort);
```

Description

`htons()` converts 16-bit quantities between host and network byte order. On machines such as the 68000, htons() is defined as a null macro in the in.h include file.

`netshort` specifies the network byte order representation.

`hostshort` specifies the host byte order representation to convert.

`htons()` is most often used with Internet addresses and ports as returned by gethostent() and getservent().

See Also

`gethostent()`, `getservent()`, `htonl()`, `ntohl()`, and `ntohs()`

inet_addr()

Internet Address Manipulation

Syntax

```
#include <types.h>
#include <socket.h>
#include <in.h>

unsigned long inet_addr(cp)
char *cp;    /* pointer to character string */
```

Description

`inet_addr()` interprets character strings representing numbers expressed in the Internet standard "." `notation. inet_addr()` returns numbers suitable for use as Internet addresses. Internet addresses are returned in network order (bytes ordered from left to right).

`inet_addr()` returns -1 for incorrectly formed requests.

Important: Refer to the first chapter for more information about Internet addresses.

See Also

`gethostent()`, `getnetent()`, `inet_lnaof()`, `inet_makeaddr()`, `inet_network()`, `inet_netof()`, and `inet_ntoa()`

inet_lnaof()

Internet Address Manipulation

Syntax

```
#include <types.h>
#include <socket.h>
#include <in.h>

int inet_lnaof(in)
struct in_addr in;       /* Internet host address */
```

Description

`inet_lnaof()` returns the local host address portion of an Internet host address. All local host address parts are returned as integer values.

`in` specifies the Internet host address to break apart.

See Also

`gethostent()`, `getnetent()`, `inet_addr()`, `inet_makeaddr()`, `inet_network()`, `inet_netof()`, and `inet_ntoa()`

inet_makeaddr()

Internet Address Manipulation

Syntax

```
#include <types.h>
#include <socket.h>
#include <in.h>

struct in_addr inet_makeaddr(net, lna)
int net,     /* Internet network number */
    lna;     /* local network address */
```

Description

`inet_makeaddr()` takes an Internet network number and a local network address and constructs an Internet address from it.

`net` specifies the Internet network number.

`lna` specifies the local network address.

`inet_netof()` and `inet_inaof()` break apart Internet host addresses, returning the network number and local network address part, respectively.

Important: Refer to the first chapter for more information about Internet addresses.

See Also

`gethostent()`, `getnetent()`, `inet_addr()`, `inet_lnaof()`, `inet_network()`, `inet_netof()`, and `inet_ntoa()`

inet_netof()

Internet Address Manipulation

Syntax

```
#include <types.h>
#include <socket.h>
#include <in.h>

int inet_netof(in)
struct in_addr in;    /* Internet host address */
```

Description

`inet_netof()` returns the network number portion of an Internet host address. All local network address parts are returned as integer values.

`in` specifies the Internet host address to break apart.

See Also

`gethostent()`, `getnetent()`, `inet_addr()`, `inet_lnaof()`, `inet_makeaddr()`, `inet_network()`, and `inet_ntoa()`

inet_network()

Internet Address Manipulation

Syntax

```
#include <types.h>
#include <socket.h>
#include <in.h>

unsigned long inet_network(cp)
char *cp;     /* pointer to character string */
```

Description

`inet_network()` interprets character strings representing numbers expressed in the Internet standard "." notation. `inet_network()` returns numbers suitable for use as Internet network numbers. Network numbers are returned as unsigned long values.

`cp` is a pointer to a character string.

`inet_network()` returns –1 for incorrectly formed requests.

See Also

`gethostent()`, `getnetent()`, `inet_addr()`, `inet_lnaof()`, `inet_makeaddr()`, `inet_netof()`, and `inet_ntoa()`

inet_ntoa()

Internet Address Manipulation

Syntax

```
#include <types.h>
#include <socket.h>
#include <in.h>

char *inet_ntoa(in)
struct in_addr in;   /* Internet address */
```

Description

`inet_ntoa()` takes an Internet address and returns a pointer to a string in the base 256 notation a.b.c.d described in the first chapter.

Internet addresses are returned in network order (bytes ordered from left to right).

Important: Refer to the first chapter for more information about Internet addresses.

Important: The return value from inet_ntoa() is a pointer to static information which is overwritten in each call.

See Also

`gethostent()`, `getnetent()`, `inet_addr()`, `inet_lnaof()`, `inet_makeaddr()`, `inet_network()`, and `inet_netof()`

listen()

Listen for Connections on Socket

Syntax

```
listen(s, backlog)
int s,                      /* socket */
    backlog;                /* maximum length of queue */
```

Description

To accept connections, socket() first creates a socket, specifies a backlog for incoming connections with listen(), and then accepts the connections with accept(). The listen call applies only to sockets of type SOCK_STREAM.

`s` specifies the path number of the socket.

`backlog` defines the maximum length to which the queue of pending connections may grow. If a connection request arrives with the queue full, the client receives an ECONNREFUSED error.

`listen()` returns 0 if successful. Otherwise, it returns a –1.

Errors

If unsuccessful, listen() may return one of the following:

Error:	Description:
E_ILLFNC	The socket must be bound in order to listen.
EADDRINUSE	Cannot connect to port zero or a wildcard address.
EOPNOTSUPP	The socket is not of a type that supports the operation listen.

See Also

`accept()`, `connect()`, and `socket()`

ntohl()

Convert 32 Bit Values Between Network and Host Byte Order

Syntax

```
#include <types.h>
#include <in.h>

u_long hostlong;           /* host byte order representation */
u_long netlong;            /* network byte order representation    */

hostlong = ntohl(netlong);
```

Description

`ntohl()` converts 32-bit quantities between network and host byte order. On machines such as the 68000, ntohl() is defined as a null macro in the include file in.h.

`hostlong` specifies the host byte order to be converted to from netlong.

`netlong` specifies the network byte order to convert to hostlong.

`ntohl()` is normally used with Internet addresses and ports as returned by gethostent() and getservent().

See Also

`gethostent()`, `getservent()`, `htons()`, `htonl()`, and `ntohs()`

ntohs()

Convert 32 Bit Values Between Network and Host Byte Order

Syntax

```
include <types.h>
#include <in.h>

u_short hostshort;         /* host byte order representation */
u_short netshort;          /* network byte order representation */

hostshort = ntohs(netshort);
```

Description

`ntohs()` converts 16-bit quantities between network and host byte order. On machines such as the 68000, ntohs() is defined as a null macro in the include file in.h.

`hostshort` specifies the host byte order to be converted to from netshort.

`netshort` specifies the network byte order to convert to hostshort.

`ntohs()` is normally used with Internet addresses and ports as returned by `gethostent()` and `getservent()`.

See Also

`gethostent()`, `getservent()`, `htonl()`, `ntohl()`, and `htons()`

recv()

Receive Message from Socket

Syntax

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

int cc;                    /* length of message */

cc = recv(s, buf, len, flags)
int s;                     /* socket */
char *buf;                 /* buffer into which the message is received */
int len,                   /* length of buffer */
    flags;                 /* currently not supported */
```

Description

`recv()` receives messages from a socket. Use recv() only on connected sockets (see connect()).

The message length is returned in cc. 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 (see socket()).

`s` specifies the path number of the socket.

`buf` is a pointer to the buffer into which the message is received.

`len` specifies the length of the buffer.

`flags` could be set to 0.

If no messages are available at the socket, recv() waits for a message to arrive, unless the socket is non-blocking. In this case, –1 is returned with the external variable errno set to EWOULDBLOCK.

You may use _ss_sevent() to determine when more data arrives.

Errors

If unsuccessful, recv() may return one of the following:

Error:	Description:
E_BMODE	Socket not bound.
ENOTCONN	Socket not connected.
ESHUTDOWN	Socket is marked for shutdown.
EWOULDBLOCK	The socket is marked non-blocking and the receive operation would block.

See Also

`connect()`, `getsockopt()`, `recvfrom()`, `send()`, `sendto()`, and `socket()`

recvfrom()

Receive Message from Socket

Syntax

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

int cc;                    /* length of message */
int s;                     /* socket */
char *buf;                 /* buffer into which message is received */
int len,                   /* length of buffer */
    flags;                 /* currently not supported */
struct sockaddr *from;     /* buffer specifying sender of message */
int *fromlen;              /* initialized to the size of from */

cc = recvfrom(s, buf, len, flags, from, fromlen)
```

Description

`recvfrom()` receives messages from a socket. You may use recvfrom() to receive data on a socket in an unconnected state.

The length of the message is returned in cc. If a message is too long to fit in the supplied buffer, excess bytes may be discarded.

`s` specifies the path number of the socket.

`buf` is a pointer to the buffer into which the message is received.

`len` specifies the length of the buffer.

`flags` could be set to 0.

`from` is a pointer to a buffer that specifies the sender of the message. If from is non-zero, the source address of the message is filled in.

`fromlen` is initialized to the size of the buffer associated with from and modified on return to indicate the actual size of the address stored.

If no messages are available at the socket, the call waits for a message to arrive, unless the socket is non-blocking. In this case, –1 is returned with the external variable errno set to EWOULDBLOCK.

You may use _ss_sevent() to determine when more data arrives.

Errors

If unsuccessful, recvfrom() may return one of the following:

Error:	Description:
ENOTCONN	Socket not connected.
ESHUTDOWN	Socket is marked for shutdown.
EWOULDBLOCK	The socket is marked non-blocking and the receive operation is blocked.

See Also

`getsockopt()`, `recv()`, `send()`, `sendto()`, and `socket()`

send()

Send Message from Socket

Syntax

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

int cc;                     /* length of message */

cc = send(s, msg, len, flags)
int s;                      /* a socket created with socket() */
char *msg;                  /* the message to send */
int len,                    /* length of message */
    flags;                  /* currently not supported */
```

Description

`send()` transmits a message to another socket. Use send() only when the socket is in a connected state.

`cc` specifies the length of the message.

`s` specifies the path number of a socket created with socket().

`msg` is a pointer to the message to send.

`len` specifies the length of the message.

`flags` is currently not supported. Set flags to 0.

If the message is too long to pass atomically through the underlying protocol, an error is returned and the message is not transmitted.

If no message space is available at the socket to hold the transmitted message, send() normally blocks, unless the socket has been placed in non-blocking I/O mode. You may use _ss_sevent() to determine when you can send more data.

No indication of failure to deliver is implicit in a send. Return values of –1 indicate some locally detected errors. `send()` returns the number of characters sent or –1 if an error occurred.

Errors

If unsuccessful, send() may return one of the following:

Error:	**Description:**
ENOTSOCK	The parameter s is not a socket.
EMSGSIZE	The socket requires that messages be sent atomically, and the message size made this impossible.
EWOULDBLOCK	The socket is marked non-blocking and the requested operation would block.

See Also

`recv()`, `recvfrom()`, `sendto()`, and `socket()`

sendto()

Send Message from Socket

Syntax

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

int cc;                    /* length of message */

cc = sendto(s, msg, len, flags, to, tolen)
int s;                     /* a socket created with socket() */
char *msg;                 /* message to send */
int len,                   /* length of message */
    flags;                 /* currently not supported */
struct sockaddr *to;       /* address of target */
int tolen;                 /* size of to */
```

Description

`sendto()` transmits a message to another socket. You may use sendto() with unconnected sockets.

`cc` specifies the length of the message.

`s` specifies the path number of a socket created with socket().

`msg` is a pointer to the message to send.

`len` specifies the length of the message.

`flags` could be set to 0.

`to` is a pointer to the address of the target.

`tolen` specifies the size of to.

If the message is too long to pass atomically through the underlying protocol, an error is returned and the message is not transmitted.

If no message space is available at the socket to hold the message to transmit, send() normally blocks, unless the socket has been placed in non-blocking I/O mode. Use _ss_sevent() to determine when you can send more data.

No indication of failure to deliver is implicit in a send. Return values of –1 indicate some locally detected errors. `send()` returns the number of characters sent or –1 if an error occurred.

Errors

If unsuccessful, send() may return one of the following:

Error:	Description:
EMSGSIZE	The socket requires that messages be sent atomically; the message size made this impossible.
EWOULDBLOCK	The socket is marked non-blocking and the requested operation would block.

See Also

`recv()`, `recvfrom()`, `send()`, and `socket()`

sethostent()

Set Network Host Entry

Syntax

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

sethostent(stayopen)
int stayopen;              /* flag */
```

Description

`sethostent()` links the calling process to inetdb, if necessary, and resets the pointer to the beginning of the hosts entries. `sethostent()` returns a null pointer (0) on EOF or error.

`gethostent()` reads the next hosts entry of inetdb. `endhostent()` unlinks the calling process from inetdb.

Important: All information is contained in a static area. You must copy the information to save it. Only the Internet address format is currently understood.

Important: Currently, OS-9/Internet ignores the stayopen flag. It is included for compatibility only.

See Also

`endhostent()`, `gethostbyaddr()`, `gethostbyname()`, `gethostent()`, `gethostname()`, and `sethostname()`

setnetent()

Set Network Entry

Syntax

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

setnetent(stayopen)
int stayopen;              /* flag */
```

Description

`setnetent()` links the calling process to inetdb, if necessary, and resets the pointer to the beginning of the networks entries. `setnetent()` returns a null pointer (0) on EOF or error.

`getnetent()` reads the next networks entry of inetdb. `endnetent()` unlinks the calling process from inetdb

Important: Currently, OS-9/Internet ignores the stayopen flag. It is included for compatibility only.

See Also

`endnetent()`, `getnetbyaddr()`, `getnetbyname()`, and `getnetent()`

setprotoent()

Set Network Protocol Entry

Syntax

```
#include <sys/socket.h>
#include <netdb.h>
setprotoent(stayopen)
int stayopen;              /* flag */
```

Description

`setprotoent()` links the calling process to inetdb, if necessary, and resets the pointer to the beginning of the protocol entries. `setprotoent()` returns a null pointer (0) on EOF or error.

`getprotoent()` reads the next protocols entry. `endprotoent()` unlinks the calling process from inetdb.

Important: Currently, OS-9/Internet ignores the stayopen flag. It is included for compatibility only.

See Also

`endprotoent()`, `getprotobynumber()`, `getprotobyname()`,
and `getprotoent()`

setservent()

Set Network Services Entry

Syntax

```
#include <socket.h>
#include <netdb.h>

setservent(stayopen)
int stayopen;              /* flag */
```

Description

`setservent()` links the calling process to inetdb, if necessary, and resets the pointer to the beginning of the services entries. `setservent()` returns a null pointer (0) on EOF or error.

`getservent()` reads the next services entry. `endservent()` unlinks the calling process from inetdb

Important: Currently, OS-9/Internet ignores the stayopen flag. It is included for compatibility only.

See Also

`endservent()`, `getprotoent()`, `getservbyport()`, `getservbyname()`, and `getservent()`

setsockopt()

Set Options on Sockets

Syntax

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

setsockopt(s, level, optname, optval, optlen)
int s,                    /* socket */
    level,                /* socket level */
    optname;              /* option name */
char *optval;             /* option values */
int optlen;               /* option length */
```

Description

`setsockopt()` sets options associated with a socket. Options may exist at multiple protocol levels; they are always present at the uppermost socket level.

`s` specifies the path number of the socket.

When setting socket options, you must specify the level at which the option resides and the name of the option.

- To set options at the socket level, specify level as SOL_SOCKET.
- To set options at any other level, supply the protocol number of the appropriate protocol controlling the option.

For example, to indicate that TCP protocol will interpret an option, set level to the protocol number of TCP (see getprotoent()).

`optname` and any specified options are passed uninterpreted to the appropriate protocol module for interpretation. The include file socket.h contains definitions for socket level options (see socket()). Options at other protocol levels vary in format and name.

`optval` and `optlen` access option values for setsockopt(). If no option value is supplied, optval may be supplied as 0.

`setsockopt()` returns 0 if the call succeeds. Otherwise, it returns -1.

Important: OS-9/Internet currently does not support any socket-level options.

Errors

If unsuccessful, setsockopt() may return one of the following:

Error:	Description:
EBADF	s is not a valid descriptor.
ENOTSOCK	s is a file, not a socket.
ENOPROTOOPT	The option is unknown.
EFAULT	The address pointed to by optval is not in a valid part of the process address space.

See Also

`getsockopt()`, `getprotoent()`, and `socket()`

shutdown()

Shut Down Part of Full-Duplex Connection

Syntax

```
shutdown(s,how)
int s,       /* socket */
    how;     /* receive and send permissions * /
```

Description

`shutdown()` shuts down all or part of a full-duplex connection of the socket specified by s.

`s` specifies the path number of the socket.

`how` specifies the method for receiving and sending permissions.

- If how is 0, further receives are disallowed.
- If how is 1, further sends are disallowed.
- If how is 2, further sends and receives are disallowed.

`shutdown()` returns 0 if it succeeds. Otherwise, it returns -1.

If unsuccessful, shutdown() may return the following error:

Error:	Description:
ENOTCONN	The specified socket is not connected.

See Also

`connect()` and `socket()`

socket()

Create Endpoint for Communication

Syntax

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

int s;                     /* socket */

s = socket(af, type, protocol)
int af,                    /* address format */
    type,                  /* type of socket */
    protocol;              /* protocol to use */
```

Description

`socket()` creates an endpoint for communication and returns a data number.

`s` specifies the path number of the socket.

`af` specifies an address format with which addresses specified in later operations using the socket should be interpreted. These formats are defined in the include file socket.h. The following formats are accepted:

Format:	Description:
AF_ETHER	Ethernet address family.
AF_INET	ARPA Internet address family.
AF_UNIX	Local to host address family.

The socket's indicated type specifies the semantics of communication. Currently defined types are:

Type:	Description:
SOCK_STREAM	Provides sequenced, reliable, two-way connection based byte streams with an out-of-band data transmission mechanism.
SOCK_DGRAM	Supports datagrams (connectionless, unreliable messages of a fixed (typically small) maximum length).
SOCK_RAW	Raw socket for AF_ETHER (connectionless, unreliable data transmission).

`protocol` specifies a particular protocol to use with the socket. Normally only a single protocol exists to support a particular socket type, using a given address format. However, many protocols may exist, in which case a particular protocol must be specified in this manner. The protocol number is particular to the communication domain in which communication is to take place. For more information, refer to the earlier discussion in this manual on the services and protocols database.

Important: Refer to the chapter on sockets for more information.

SOCK_STREAM Sockets

Sockets of type SOCK_STREAM are full-duplex byte streams, similar to pipes. A stream socket must be in a connected state before any data may be sent or received on it. A connect() call creates a connection to another socket.

Once connected, data may be transferred using read() and write() calls or some variant of the send() and recv() calls.

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 and calls will indicate an error with -1 returns, and with ETIMEDOUT as the specific code in the global variable errno.

SOCK_DGRAM Sockets

SOCK_DGRAM sockets allow sending of datagrams to correspondents named in send() calls. You can also receive datagrams at such a socket with recv().

Socket Level Options

Socket level options control the operation of sockets. These options are defined in socket.h. `setsockopt()` and `getsockopt()` set and get the following options, respectively:

Option:	**Description:**
SO_DEBUG	Turns on recording of debugging information. This enables debugging in the underlying protocol modules.
SO_REUSEADDR	Indicates the rules used in validating addresses supplied in a bind() call should allow re-use of local addresses.
SO_KEEPALIVE	Enables the periodic transmission of messages on a connected socket. If a connected party fails to respond to these messages, the connection is considered broken and processes using the socket are notified via a SIGPIPE signal.
SO_DONTROUTE	Indicates that outgoing messages should bypass the standard routing facilities. Instead, messages are directed to the appropriate network interface according to the network portion of the destination address.
SO_LINGER	Controls the actions taken when unsent messages are queued on a socket and a close() is performed. If the socket promises reliable delivery of data and SO_LINGER is set, the system blocks the process on the close attempt until it is able to transmit the data or until it decides it is unable to deliver the information. A time out period, referred to as the linger interval, is specified by setsockopt() when SO_LINGER is requested.
SO_DONTLINGER	Controls the actions taken when unsent messages are queued on a socket and a close() is performed. If SO_DONTLINGER is specified and a close is issued, the system processes the close in a manner which allows the process to continue as quickly as possible.

If successful, socket() returns the descriptor referencing the socket. Otherwise, it returns –1.

Important: Only the SO_REUSEADR option is supported in the current implementation of OS-9/Internet.

Errors

If unsuccessful, socket() may return one of the following:

Error:	**Description:**
EAFNOSUPPORT	The specified address family is not supported in this version of the system.
ESOCKTNOSUPPORT	The specified socket type is not supported in this address family.
EPROTONOSUPPORT	The specified protocol is not supported.
ENOBUFS	No end buffer space is available. The socket cannot be created.

See Also

`accept()`, `bind()`, `connect()`, `getsockname()`, `getsockopt()`, `listen()`, `recv()`, `send()`, and `shutdown()`

Error Codes

The following error messages can be returned by socket access to the Internet software. Some of these error codes are appropriate for the ftp and telnet programs. Some error codes are never returned in this implementation but are listed here for future compatibility.

Error number:	Description:
007:001	I/O operation would block An operation that would cause a process to block was attempted on a socket in non–blocking mode.
007:002	I/O operation now in progress An operation that takes a long time to complete (such as connect()) was attempted on a socket in non–blocking mode.
007:003	Operation already in progress An operation was attempted on a non–blocking object that already had an operation in progress.
007:004	Destination address required The attempted socket operation requires a destination address.
007:005	Message too long A message sent on a socket was larger than the internal message buffer or some other network limit. Messages should be smaller than 32768 bytes.
007:006	Protocol wrong type for socket A protocol was specified that does not support the semantics of the socket type requested. For example, an AF_INET UDP protocol as SOCK_STREAM is the wrong protocol type for the socket.
007:007	Bad protocol option A bad option or level was specified in getsockopt() or setsockopt().
007:008	Protocol not supported The requested protocol is not available or not configured for use.
007:009	Socket type not supported The requested socket type is not supported or not configured for use.
007:010	Operation not supported on socket For example, accept() on a datagram socket.
007:011	Protocol family not supported
007:012	Address family not supported by protocol
007:013	Address already in use Only one use of each address is normally permitted. Wildcard use and connectionless communication are the exceptions.

Error number:	**Description:**
007:014	Can’t assign requested address Normally results from an attempt to create a socket with an address not on this machine.
007:015	Network is down The network hardware is not accessible.
007:016	Network is unreachable The network is unreachable. Usually caused by network interface hardware that is operational, but not physically connected to the network. This error can also be caused when the network has no way to reach the destination address.
007:017	Network dropped connection on reset The host you were connected to crashed and rebooted.
007:018	Software caused connection abort A connection abort was caused by the local (host) machine.
007:019	Connection reset by peer A peer forcibly closed a connection. This normally results from a loss of the connection on the remote socket due to a time out or reboot.
007:020	No buffer space available A socket operation could not be performed because the system lacked sufficient buffer space or a queue was full.
007:021	Socket is already connected A connect() request was made on an already connected socket. Also caused by a sendto() request on a connected socket to a destination which is already connected.
007:022	Socket is not connected A request to send or receive data was rejected because the socket was not connected or no destination was given with a datagram socket.
007:023	Can’t send after socket shutdown
007:024	Too many references
007:025	Connection timed out A connect() or send() request failed because the connected peer did not properly respond after a period of time. The time out period depends on the protocol used.
007:026	Connection refused by target No connection could be established because the target machine actively refused it. This usually results from trying to connect to a service that is inactive on the target host.
007:027	Mbuf too small for mbuf operation
007:028	Socket module already attached
008:029	Path is not a socket

Appendix B

Example Programs

This appendix contains three examples. Each example includes a client program and a server program. You should use these programs to better understand how to use OS-9/Internet. These programs can also be used as templates for writing your own programs.

Example 1: Socket Operations

The following programs, tcpsend.c and tcprecv.c, illustrate some of the socket operations. tcpsend.c is the client program, and tcprecv.c is the server program. These programs work together and provide a good template for programs using TCP sockets.

There are two socket features to keep in mind when creating your own programs.

- Both the sender and the server must create sockets. The subsequent connect and accept operations create the link between the sockets.
- The operating system uses port numbers up to 1024, but applications may choose any port numbers. However, normal user application programs use port numbers greater than 5000. Because port numbers are often compiled into programs (as done in this example), you need to consider the allocation of numbers to avoid collisions. Strange things happen when a port number is shared unexpectedly.

The tcpsend.c Program

This program creates a socket, connects it to tcprecv, and sends data through it. This program is complete and can be compiled on OS-9 systems.

```
/* <<<<<tcpsend.c>>>>>>>
       Syntax to execute the program:       tcpsend host [file –]
       Send the named file or standard input through a socket to the named host.
*/

#include <stdio.h>
#include <errno.h>
#include <types.h>
#include <socket.h>
#include <in.h>
#include <netdb.h>

#define RETRY_COUNT 10
#define SOCKET_PORT 2700

struct sockaddr_in ls_addr;
struct data {
     int code, count;
     char data[512];
};

char msgbuf[2048];

main(argc, argv)
register char **argv;
{
     register int count = RETRY_COUNT, s, ifile;
     register int totbytes=0;
     register struct data *pack = (struct data *)msgbuf;
     struct hostent *host;

     /*
      *  If the first command line argument starts with
      *  a ’–’ print a help message and quit.
      */

     if (*argv[1] == ’–’) {
          fprintf(stderr, “tcpsend <host> <file>\n”);
          exit(0);
     }

     /* Convert the host name from the command line into a host structure. */
```

```
if ((host = gethostbyname(*++argv)) == NULL) {
     fprintf(stderr, “don’t know host ’%s’\n”, *argv);
     endhostent();
     exit(0);
}

/*
 *  Open a file for input. If the second command line argument is
 *  a ’–’, use standard input (path 0). If the second command line
 *  argument is not ’–’, it “must” be a string that we will use as
 *  a file name for the input file.
 */


if (*argv[1] == ’–’)
     ifile = 0;
else if ((ifile = open(*++argv, 1)) == –1) {
     fprintf(stderr, “can’t open file ’%s’\n”, *argv);
     endhostent();
     exit(errno);
}

/*
     Make a connection:
     *  Open a socket:
          Request internet domain and a
          sequenced, reliable, two–way connection.
     *  Initialize an internet–style socket address:
          –Take the family from the host type.
          –Use a port number that we think will be
           unique to this set of programs.
          –Copy the host address into the in_addr field.
     *  Connect the socket to the address.
     *  If the connect fails, keep trying up to
        RETRY_COUNT times.
*/

while (1) {
     /* Create a socket */
     if ((s = socket(AF_INET, SOCK_STREAM, 0)) == –1) {
          fprintf(stderr,“can’t open /socket\n”, errno);
          endhostent();
          exit(errno);
     }

     /* Initialize a socket address */
     ls_addr.sin_family = host–>h_addrtype;
```

```
                ls_addr.sin_port =  SOCKET_PORT;
                memcpy(&ls_addr.sin_addr.s_addr, host–>h_addr, host–>h_length);

                /* Connect the socket to the other socket specified in ls_addr. */

                if (connect(s, &ls_addr, sizeof ls_addr) == –1) {
                     fprintf(stderr, “connect failed\n”);
                     if (––count) {
                          close(s);
                          fprintf(stderr,“retry connect %d\n”, RETRY_COUNT – count);
                          sleep(1);
                          continue; /* retry */
                     }
                     endhostent();
                     exit(errno);
                }
                break;
           }

  endhostent();
      printf(“Connection established\nSending file ’%s’...”, *argv);
      fflush(stdout);

      /* Copy the input file to the socket. */
      while ((count = read(ifile, msgbuf, sizeof(msgbuf))) > 0) {
           if (write(s, msgbuf, count) != count) {
                fprintf(stderr, “socket write error\n”);
                exit(errno);
            }
            totbytes += count;
      }
      if (count != 0) {
           fprintf(stderr, “read error on file\n”);
           exit(errno);
       }

      /* Close the socket and the input file, and exit. */
      close(s);
      close(ifile);
      printf(“sent %u bytes\n”, totbytes);
 }
```

The tcprecv.c Program

This program creates a socket that accepts tcpsend’s connection and copies the data from the socket into a file. This program is complete and can be compiled on OS-9 systems.

```
/* <<<<<<tcprecv.c>>>>>>
     Syntax for executing program:    tcprecv [– file]
     Copy a file from a socket into the named file.
*/

#include <stdio.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define SOCKET_PORT     2700       /* This number must match the port number
                                      used by the sender. */

struct sockaddr_in ls_addr, to;

char msgbuf[20480];

main(argc, argv)
register char **argv;
{

     register int sx, s, count = 1, totbytes = 0, ofile;
     auto int size;

     /* If the first argument starts with a ’–’, give a help message */
     if (*argv[1] == ’–’) {
          fprintf(stderr,“tcprecv <file>\n”);
          exit(0);
     }

     /*      Create a file to hold the data that will come through the socket.
*/
if ((ofile = creat(*++argv, S_IREAD|S_IWRITE)) == –1) {
           fprintf(stderr, “can’t open file ’%s’\n”, *argv);
           exit(errno);
/*
Create a socket for internet stream protocol.
The procedure is:
          Create a socket.
          Bind it to the socket port  (the port amounts to the name).
          Listen for a connection.
          Accept the connection which creates a new socket.
      Read data from the new socket.
 */

if ((sx = socket(AF_INET, SOCK_STREAM, 0)) == –1) {
          fprintf(stderr,“can’t open /socket\n”, errno);
          exit(errno);
}

/* Bind the socket to SOCKET_PORT */
ls_addr.sin_family = AF_INET;
ls_addr.sin_port =  SOCKET_PORT;
ls_addr.sin_addr.s_addr = 0;

if (bind(sx, &ls_addr, sizeof ls_addr) == –1) {
     fprintf(stderr,“can’t bind socket\n”);
     exit(errno);
}

/* Wait for a connection attempt on the socket */
if (listen(sx, 1) < 0) {
     fprintf(“tcp_listen – failed!\n”);
     exit(errno);
}

size = sizeof(struct sockaddr_in);

/*
Accept a connection.
This function picks up the pending connection, and
creates a clone of socket sx.
The clone can be used to send and receive data.
The original socket can listen for additional
connections, but it cannot send or receive.

Accept will set the to variable to the address of
the other end of the connection.
*/

if ((s = accept(sx, &to, &size)) < 0) {
     fprintf(stderr, “can’t accept\n”);
     exit(errno);
}
close (sx);             /* We don’t expect another connection. */

printf(“connected to %d.%d.%d.%d\n”, to.sin_addr.s_addr);
```

```
while (count) {          /* While data comes from the socket */
/* Read the socket */
     if ((count = read(s, msgbuf, sizeof(msgbuf))) < 0) {
          fprintf(stderr, “can’t recv (cnt=%d)\n”, count);
               exit(errno);
          else if (count == 0) {
               break;
          else {              /* If the read got anything, write it. */
               if (write(ofile, msgbuf, count) != count) {
                    fprintf(stderr, “can’t write output\n”);
                    exit(errno);
                }
                totbytes += count;
           }
     }
     /* Close the socket and the output file. */
     close(ofile);
     close(s);
     printf(“read %d bytes\n”, totbytes);
}
```

Example 2: Beam and Target

The next two programs, beam.c and target.c, test the efficiency of the network. beam.c is the client program, and target.c is the server program.

The beam.c Program

This program sends a steady beam of packets to a host to test efficiency of the network. This program is complete and can be compiled on OS-9 systems.

```
/* <<<<<<<<<<<<<beam.c>>>>>>>>>>>>>
     Syntax for executing the program:   beam <hostname> <count>\n
*/

#include <stdio.h>
#include <types.h>
#include <machine/types.h>
#include <socket.h>
#include <in.h>
#include <netdb.h>
#include <errno.h>

#define PKT_SIZE 1024
#define START 1
#define NORMAL 2
#define END 3
#define PORT 20000

struct packet {
     long type;
     long size;
     long count;
     char buf[PKT_SIZE – 12];
};

main(argc, argv)
int argc;
char **argv;
{

     struct hostent *host;
     int s;
     struct packet pkt;
     struct sockaddr_in sockname;
     int count, msglen, i;

     /* Check to see if we have the correct number of parameters */
     if (argc < 3) {
          printf(“usage: beam <hostname> <count>\n”);
          exit(0);
     }

     /* Get the number of packets to beam */

     count = atoi(argv[2]);

     /* Get information concerning the host we are beaming to. */

     if ((host = gethostbyname(argv[1])) == (struct hostent *)0) {
           printf(“beam: can’t find entry for host %s\n”, argv[1]);
          exit(0);
     }

     /*
     Steps to sending UDP packets:
     1. Open a socket with type SOCK_DGRAM
     2. Bind socket to this process. Since we are a client, we
         don’t want a port, have the system get one for us.
     3. Sendto to the target.
  */

     if ((s = socket(AF_INET, SOCK_DGRAM, 0)) == –1) {
          exit(_errmsg(errno, “beam: socket call failed – ”));
     }

     sockname.sin_family = AF_INET;
     sockname.sin_port = 0;
     sockname.sin_addr.s_addr = INADDR_ANY;
     if (bind(s, &sockname, sizeof(sockname)) < 0) {
          exit(_errmsg(errno, “bind failed – ”));
     }

     /*
     Set up sockaddr_in for the send. From the host name we knwo
     the address, and target is already waiting on port 20000;
     */
     memcpy(&sockname.sin_addr.s_addr, host–>h_addr, host–>h_length);
     sockname.sin_port = PORT;  /* port number to beam to */
     endhostent();

     /* Set up packets for transfer and transfer them all. */

     pkt.size = PKT_SIZE;
     for (i = 0; i <= count; i++) {
          if (i == 0) {
               pkt.type = START;
          } else if (i == count) {
               pkt.type = END;
          } else {
               pkt.type = NORMAL;
          }
          pkt.count = i;
            /* Send packet to host. */
          if (sendto(s, &pkt, pkt.size, 0, &sockname, sizeof(sockname)) <        0) {
               exit(_errmsg(errno, “beam: send failed – ”));
          }
     }
}  /* end of main */
```

The target.c Program

This program receives a number of packets from a socket and checks if any were lost. This will test the efficiency of the network. This program is complete and can be compiled on OS-9.

```
/* <<<<<<<target.c>>>>>>>
     Syntax for executing the program:      target&
*/

#include <stdio.h>
#include <types.h>
#include <machine/types.h>
#include <socket.h>
#include <in.h>
#include <netdb.h>
#include <errno.h>
#include <time.h>

extern char *inet_ntoa();

#define PKT_SIZE 1024
#define START 1
#define NORMAL 2
#define END 3
#define PORT 20000

struct packet {
     long type;
     long size;
     long count;
     char buf[PKT_SIZE – 12];
};

main(argc, argv)
int argc;
char **argv;
{

     register int path1, count;
     long bytesrecv;
     time_t starttime, endtime;
     double dtime;
     double rate;
     struct packet pkt;
     struct sockaddr_in name;
     auto int namelen;
     char buf[200];
```

```
        printf(“target is active\n”);
        /*
        *  This process will read UDP or datagram sockets. The
        *  sequence to this is the following:
        *  1. Open a socket with a type of SOCK_DGRAM
        *  2. Bind socket to process on port
        *  3. recvfrom the socket
        */

        if ((path1 = socket(AF_INET, SOCK_DGRAM, 0)) == –1) {
             exit(_errmsg(errno, “target: socket failed – ”));
        }

        /*
        *  Before a bind, need to set up the sockaddr_in structure. Since
        *  this is a server process we bind to a specfic port number.
     */

        name.sin_family = AF_INET;
        name.sin_addr.s_addr = INADDR_ANY;
        name.sin_port = PORT;  /* port number */
        if (bind(path1, &name, sizeof(name)) == –1) {
             exit(_errmsg(errno, “target: bind failed – ”));
        }

        while (1) {
             /*
             *  The ’name’ variable is of a sockaddr_in structure.
             *  By passing this down to the recv from call, we can
             *  know who just sent us one of our packets.
             */          namelen = sizeof(name);
             if ((count = recvfrom(path1, &pkt, sizeof(pkt), 0, &name,
                                  &namelen)) == –1) {
                  exit(_errmsg(errno, “target: recv failed – ”));
             }

             /* Start the timming loop to see how efficient we are. */

start:       starttime = time(0);
             bytesrecv = 0;

             if (pkt.type != START) {
                  printf(“out of sequence packet received\n”);
                  continue;
             }
             printf(“Begin\n”);
```

```
          /*
          *  Loop until all the packets are received.
          *  Keep track of the number of bytes received.
          */
          do {

                namelen = sizeof(name);
                if ((count = recvfrom(path1, &pkt, sizeof(pkt), 0, &name,
                                      &namelen)) == –1) {
                     exit(_errmsg(errno, “target: recv failed – ”));
                }
           bytesrecv += pkt.size;
           } while (pkt.type == NORMAL);
           endtime = time(0);

           /*
           *  If we did not get the ending packet, start timming
           *  loop all over again.
           */

           if (pkt.type != END) {
                printf(“restarting ...\n”);
                goto start;
           }

           /* Figure and display stats */

           dtime = difftime(endtime, starttime);
           printf(“stats on exchange just completed:\n”);
           printf(“Start time : %s”, ctime(&starttime));
           printf(“End time : %s”, ctime(&endtime));
           printf(“%d bytes transferred in %.2f seconds\n”, bytesrecv,     dtime);
           if (dtime == 0.0) {
                printf(“too fast for meaningful timing\n”);
           } else {
                rate = (double)bytesrecv / dtime;
                printf(“Transfer rate = %.2f bytes/sec (%.2f Kbytes/sec)\n”,
                       rate, rate / (double)1024);
           }

     }

     close(path1);

}  /* end of main */
```

Example 3: Ethernet Raw Socket Support for ISP

The examples in this section, ethsendto and ethrecvf, test the ethernet raw socket. For simplification, flow control in the test program is replaced by tsleep(2) in ethsendto.c.

- If your receiving computer is faster than the sending computer, change this to tsleep(1) or remove it.
- If the sending computer works faster than the receiving computer, increase the tsleep() value.

In either case, an upper level protocol should offer flow control.

When using Ethernet raw sockets, the sen_type of the structure sockaddr_eth should be the same in both sender and receiver. A different number is used to handle different pairs of senders and receivers. Otherwise, you need an upper level protocol to handle the communication pair to avoid interlaying. sen_type should be in the range of 0x7f00 to 0x7fff. Do not use 0x0800, 0x0806, 0x0200, 0x1000, and 16 for sen_type, as these values are reserved for IP, ARP, PUP, TRAIL, and NTRAIL.

When the ISP system works as a router (gateway), Ethernet is sent out from the device in the last one of device_names of sockdesc/socket.a. For example, in sockdesc/socket.a device_names is dc.b “/lo0 /eni0 /le0”, /le0 is the device an ether packet sends out from a raw ether socket. However, in receive, an ether raw socket could receive AF_ETHER packet from both device drivers.

To compile ethsendto.c, use the following command:

```
cc ethsendto.c –l=/h0/lib/socklib.l –v=/h0/ISP.portpak/DEFS
```

The ethsendto.c Program

This program tests the Ethernet raw socket. Before compiling this program, you need to change dstaddr to the destination Ethernet address that you need. To get the destination Ethernet address, use the command enpstat /eni0 or the command lestat /le0 in the destination computer.

```
/*   “ethsendto.c”   */

#include <stdio.h>
#include <types.h>
#include <machine/types.h>
#include <socket.h>
#include <inet/eth.h>
#include <errno.h>

#define PKT_SIZE 1500
#define ETHERTYPE_ETH 0x7f00
#define AF_ETHER 12
#define START 1
#define NORMAL 2
#define END 3

struct packet {
     char    type;
     short   size;
     long    count;
     char    buf[PKT_SIZE – 5];
};

/* dstaddr should be changed to the destination node’s ethernet address */
static   dstaddr[6] = {0x08, 0x00, 0x3e, 0x20, 0x2e, 0xae};
static   ethaddr_any[6] = {0xff, 0xff, 0xff, 0xff, 0xff, 0xff};

main(argc, argv)
int argc;
char **argv;
{

     int s;
     struct packet pkt;
     struct sockaddr_eth sockname, dst;
     int count, msglen, i;

     if (argc < 2) {
          printf(“usage: ethsendto <count>\n”);
          exit(0);
     }

     count = atoi(argv[1]);
```

```
        if ((s = socket(AF_ETHER, SOCK_RAW, 0)) == –1) {
             exit(_errmsg(errno, “ethsendto: socket call failed – ”));
        }

        sockname.sen_family = AF_ETHER;
        sockname.sen_type = ETHERTYPE_ETH;
        for (i=0; i<6; i++)
             sockname.sen_addr.ena_addr[i] = ethaddr_any[i];

        if (bind(s, &sockname, sizeof(sockname)) < 0) {
             exit(_errmsg(errno, “bind failed – ”));
        }

        dst.sen_family = AF_ETHER;
        dst.sen_type = ETHERTYPE_ETH;
        for (i=0; i<6; i++) {
             dst.sen_addr.ena_addr[i] = dstaddr[i];
        }

        pkt.size = PKT_SIZE;
        for (i = 0; i <= count; i++) {
             if (i == 0) {
                  pkt.type = START;
             } else if (i == count) {
                  pkt.type = END;
             } else {
                  pkt.type = NORMAL;
             }
             pkt.count = i;

     /*  Don’t send too fast to make receiver lose packets.
      *  This is a raw ethernet socket, no ACK and no retransmission.
      *  ACK or retransmission should be implemented by upper
      *  level protocol if needed.
      *  If the sender is slower than the receiver, decrease the tsleep value,
      *  or take it off. You will see faster transmission speed.
      *  If the sender is faster than the receiver, increase the tsleep value.
      *  When upper level protocol is used, tsleep is no longer need.
      *  Flow control should be taken care of by upper level protocol.
      */

             tsleep(2);
             if (sendto(s, &pkt, pkt.size, 0, &dst, sizeof(dst)) < 0) {
                  exit(_errmsg(errno, “ethsendto: send failed – ”));
             }
        }
   }  /* end of main */
```

The ethrecvf.c Program

This program tests the Ethernet raw socket.

```
/*    ethrecvf.c    */

#include <stdio.h>
#include <types.h>
#include <machine/types.h>
#include <socket.h>
#include <inet/eth.h>
#include <errno.h>
#include <time.h>

#define PKT_SIZE 1500
#define AF_ETHER 12
#define ETHERTYPE_ETH 0x7f00
#define START 1
#define NORMAL 2
#define END 3

struct packet {
     char    type;
     short   size;
     long    count;
     char    buf[PKT_SIZE – 5];
};

static     ethaddr_any[6] = {0xff, 0xff, 0xff, 0xff, 0xff, 0xff};

main(argc, argv)
int argc;
char **argv;
{
     register int se1, count;
     long bytesrecv;
     time_t starttime, endtime;
     double dtime;
     double rate;
     struct packet pkt;
     struct sockaddr_eth name, from;
     auto int fromlen;
     int i;

     printf(“ethrecvf is active\n”);

     if ((se1 = socket(AF_ETHER, SOCK_RAW, 0)) == –1) {
          exit(_errmsg(errno, “ethrecvf: socket failed – ”));
```

```
        }

        name.sen_family = AF_ETHER;
        name.sen_type = ETHERTYPE_ETH;
        for (i = 0; i < 6; i++)
             name.sen_addr.ena_addr[i] = ethaddr_any[i];

        if (bind(se1, &name, sizeof(name)) == –1) {
             exit(_errmsg(errno, “ethrecvf: bind failed – ”));
        }

        from.sen_family = AF_ETHER;
        from.sen_type = ETHERTYPE_ETH;
        for (i = 0; i < 6; i++)
             from.sen_addr.ena_addr[i] = ethaddr_any[i];

        fromlen = sizeof(from);
        if ((count = recvfrom(se1, &pkt, sizeof(pkt), 0, &from,
                              &fromlen)) == –1) {
             exit(_errmsg(errno, “ethrecvf: recvfrom failed – ”));
        }

        /*
         *  Now “from” had returned the ethernet address of the sender of the packet
         *  which we received from.
         */

   start:    starttime = time(0);
             bytesrecv = 0;

             if (pkt.type != START) {
                  printf(“out of sequence packet received\n”);
                  exit(1);
             }
             printf(“Begin\n”);
             do {
                  fromlen = sizeof(from);
                  if ((count = recvfrom(se1, &pkt, sizeof(pkt), 0, &from,
                                        &fromlen)) == –1) {
                       exit(_errmsg(errno, “ethrecvf: recvfrom failed – ”));
             }


             /*
              * Now “from” had returned the ethernet address of the sender of the packet
              * which we received from. For upper level protocol, this is used to check
              * whether the packet we received is from the host we like or not.
              */
```

```
          bytesrecv += pkt.size;
     } while (pkt.type == NORMAL);
     endtime = time(0);

     if (pkt.type != END) {
          printf(“restarting ...\n”);
          goto start;
     }
     dtime = difftime(endtime, starttime);
     printf(“stats on exchange just completed:\n”);
     printf(“Start time : %s”, ctime(&starttime));
     printf(“End time : %s”, ctime(&endtime));
     printf(“%d bytes transferred in %.2f seconds\n”, bytesrecv, dtime);
     if (dtime == 0.0) {
          printf(“too fast for meaningful timing\n”);
     } else {
          rate = (double)bytesrecv / dtime;
          printf(“Transfer rate = %.2f bytes/sec (%.2f Kbytes/sec)\n”,
                 rate, rate / (double)1024);
     }
     close(se1);
}  /* end of main */
```

Using the routed Daemon

`routed` is a network routing daemon available for OS-9/Internet. It is called when you boot up the system. `routed` dynamically routes data and automatically modifies the routing tables based on broadcasts from other routed programs on gateway machines. Generally, you should run the routed program rather than modifying the ipconfig table entries.

On a system that is not acting as a gateway, routed updates the IP routing tables to allow access to other networks based on the gateway broadcasts. On a system that is acting as a gateway, routed broadcasts its routing tables on all connected networks. Thus, the network gateway information is determined dynamically from the network, rather than fixed in the static ipconfig table.

Using routed

The examples in this section, ethsendto and ethrecvf, test the ethernet raw socket. For simplification, flow control in the test program is replaced by tsleep(2) in ethsendto.c.

To use routed, follow these steps:

1. Edit the networks file in the ETC directory. Add all networks with which this node will communicate.

2. Use idbgen to create a data module containing the files hosts, networks, protocols, and services. These files are located in the ETC directory.

3. In the start.isp script file, replace the line ispstart with the following:

```
routed&
```

At this point, dynamic routing is in effect.

Using routed on Gateways

If your system is a gateway, you also need to edit socket.a in the SOCKDESC directory and change the device_names field to include all descriptors the system uses. For example:

```
device_names dc.b “/lo0 /le0 /eni0”
```

Enclose any device that sockman opens or initializes in quotes. You *must* include the device /lo0 as the first device.

This example shows a gateway configuration with two networks, /le0 and /eni0. The descriptors have the network internet numbers.

After making this change in the socket.a file, remake the socket descriptor and restart ISP. The system is now a gateway.

Implementation Notes for SysMbuf

ISP Version 1.3 (and greater) includes a substantially improved mbuf allocator. The previous version of this code used an event and allocated memory from the general system memory (via F$SRqMem). This resulted in many system calls per mbuf allocation (or return) and had the effect of loading the system and potentially causing IRQ response delays during periods of heavy network use.

When the routine is installed, all of the memory needed for mbufs is allocated at once. This has a number of useful effects.

- Because all the memory is allocated at once, all the mbufs fall into a particular range of addresses. Debugging memory allocation and wild pointer problems may be easier if all the mbufs are in a known place.
- The mbuf allocator uses the F$SRqCMem call so that systems that are using colored memory set up tables can be configured to allocate mbuf memory from a designated area of memory, instead of anywhere the system chooses.

The allocator uses a fixed-block allocation scheme. As a result of the allocation method, you can use a bitmap to track free space. Hand-coded assembler routines are used for the bit allocation, deallocation, and search functions. For best performance on the processors with bitfield instructions, a separate binary uses those instructions. You should use SysMbuf_010 for processors less than 68020, and SysMbuf_020 for processors 68020 or greater.

Because the mbuf code is called and shared by multiple IRQ service routines, by default, it masks IRQs to level 7 to protect allocate and deallocate. With the fast algorithm SysMbuf uses this is usually not a problem. You can patch the module to limit the raising of the mask to the level of the highest IRQ service routine that uses mbufs.

Use this feature with extreme care. If not done properly, this destroys the integrity of the mbuf free space resulting in a non-functioning system. Normally, you should not use this feature unless the operation of the network interferes with a higher-level IRQ routine.

The default minimum allocation blocksize is 64 bytes. This has been demonstrated to be nearly the optimal size for these routines. A smaller blocksize uses more bitmap memory and requires more iterations through the code, but wastes less memory for small allocations. A larger blocksize uses less bitmap memory and requires less iterations through the search code, but wastes more memory for small allocations. ISP itself almost never requests blocks less than 64 bytes. This allocations size allows up to 2048 bytes to be bitmapped in one 32-bit search/load/store. Ethernet mbuf requests are never larger than 1536.

The following locations in the module are of interest:

```
Addr   0 1  2 3  4 5  6 7  8 9  A B  C D  E F 0 2 4 6 8 A C E
----  ---- ---- ---- ---- ---- ---- ---- ---- ----------------
0000  4afc 0001 0000 08de 0000 0000 0000 08d2 J|.....^.......R
0010  0555 0c01 a000 000a 0000 0000 0000 0000 .U.. ...........
0020  0000 0000 0000 0000 0000 0000 0000 1c50 ...............P
0030  0000 0050 0000 0000 0000 0826 0006 8020 ...P.......&...
0040  0007 8008 0000 0000 0000 0040 0002 0000 ...H.......@....
0050  48e7 0040 43fa 0012 4e40 0032 2c4b 6100 Hg.@Cz..N@.2,Ka.

Offset   Length Meaning
------   ------ -------
003c     long   Processor identifier: 68010 or 68020 (not changable)
0040     short  Max. IRQ mask level:  7 (patchable)
0042     short  Reserved
0044     long   Colored memory typecode: 0 (patchable)
0048     long   Min. allocation blocksize:    64 (patchable)
004c     long   Memory to use for Mbufs:     128kb (patchable)
```

Glossary

This section contains definitions for some of the terms used in this manual.

address class	A classification of network Internet Protocol addresses. The three main classes are A, B, and C. The classes are separated according to the number of hosts attached to each network. Each Internet Protocol address consists of a network portion (netid) and a host portion (hostid). An Internet Protocol address identifies a network to which a host is attached and specifies the specific host attached to the network.
ARP	Address Resolution Protocol. TCP/IP uses ARP to dynamically bind a high level IP address to a low-level physical hardware address.
ARPANET	An early network established for networking research by the Advanced Research Projects Agency (ARPA). ARPA is now known as DARPA. ARPANET consists of individual packet switch nodes interconnected by leased lines.
BBRAM	Battery-backed RAM.
broadcasting	A system that delivers a copy of a given packet to all attached hosts. You may implement broadcasting with either hardware or software. See also packet.
checksum	A small integer value computed from a sequence of octets by treating them as integers and computing the sum. A checksum is used to detect errors that result when the sequence of octets is transmitted from one machine to another. Typically, protocol software computes a checksum and appends it to a packet when transmitting. When received, the protocol software verifies the contents of the packet by recomputing the checksum and comparing the value sent. Many TCP/IP protocols use a 16-bit checksum computed with one’s complement arithmetic with all integer fields in the packet stored in network byte order.

client	Any process that wishes to use a service provided by a network server.
connected socket	A socket which has been bound to a permanent destination. See also socket.
connectionless	A form of interconnection that allows communication to take place without first establishing a connection.
daemon	A system process that runs in the background. It has no attached terminal and never exits.
datagram socket	A communications channel that is not necessarily sequenced, reliable, or unduplicated. Datagram sockets allow data packets, called messages, to flow bi-directionally. Because datagram sockets do not need to be connected to a peer, messages are sent to a destination address. See also socket.
DMA	Direct Memory Access.
domain	In the Internet, a part of a naming hierarchy. Syntactically, an Internet domain name consists of a sequence of names (labels) separated by periods (dots). For example, mcrw.ulm.ia.us.
dotted decimal notation	The syntactic representation for a 32-bit integer that consists of four 8-bit numbers written in base 10 with periods (dots) separating them. Used to represent IP addresses in the Internet as in: 192.10.54.3.
encapsulation	The practice of adding additional header information for each layer of the protocol through which a packet passes.
file transfer protocol (ftp)	A high-level protocol used to transfer files between systems. You can use ftp to log into a remote system, known as a server, with a proper login user name and password (if required). You can then look at directories on the server and transfer files to and from a server. See also protocol.

fragments	The small pieces into which a datagram is divided in order for a datagram to pass through a network that has a small maximum transfer unit (MTU). See also maximum transfer unit.
fragmentation	The process of breaking an IP datagram into smaller pieces to fit the requirements of a given physical network. See maximum transfer unit.
gateway	A computer dedicated to connecting two or more networks. A gateway routes message packets from one system to another. Gateways route packets based on the destination network, not on destination host.
globbing	Refers to the expansion of wildcards for remote file names.
host	Any computer system that connects to a network.
inetdb	OS-9/Internet uses the data module inetdb to contain the Internet data files. These files are normally kept in the UNIX /etc directory. The data in these files are kept in a data module rather than in the files to allow a totally ROM-based Internet system. The idbgen utility creates inetdb from the four data files. Examples of the four files are provided in the ETC directory on the distribution disk.
Internet	The connection of two or more networks that allow computers on one network to communicate with computers on another network. The internet is sometimes referred to as the internetwork.
Internet address	A 32-bit address assigned to hosts using TCP/IP. See dotted decimal notation.
Internet Protocol (IP)	The Internet datagram delivery protocol that is central to the Internet protocol family. Programs may use IP through higher-level protocols such as the User Datagram Protocol and the Transmission Control Protocol or may interface directly using a raw socket. Protocol options defined in the IP specification may be set in outgoing datagrams. See also User Datagram Protocol (UDP) and Transmission Control Protocol (TCP).

IP datagram	The basic unit of information passed across an internet using Transmission Control Protocol/Internet Protocol (TCP/IP). An IP datagram is divided into a header area and a data area. The datagram header contains the source and destination Internet Protocol address and a type field identifying the datagram's contents. See also Transmission Control Protocol (TCP).
ISDN	Integrated Services Digital Network. An emerging technology which combines voice and digital network services in a single medium making it possible to offer customers digital data services as well as voice connections through a single wire.
ISO	International Organization for Standardization. Best known for the seven-layer OSI Reference Model.
maximum transmission unit (MTU)	The largest amount of data that can be transmitted across a given physical network.
mbuf	A common data structure used by all parts of the ISP system for data storage. The mbuf data structure provides an efficient way to store variable-length data blocks.
multicast	A special form of broadcast where copies of the packet are delivered to only a subset of all possible destinations. See broadcast.
network	A computer network is the hardware and software used to allow computers to communicate with each other. For this manual, the term network refers only to packet-switched networks.
non-blocking	A socket that is non-blocking will not block when data is unavailable for reading.
packet	A block of data for data transmission. Each packet carries identification that allows computers on the network to know whether it is destined for them or how to pass it on to its correct destination.

peer	Either of a pair of communicating sockets.
port	The abstraction used by Internet transport protocols to distinguish among multiple, simultaneous connections to a single destination host.
protocol	A formal description of message formats, the handling of error conditions, and rules that govern how to locate, request, accept, and terminate a service between two or more machines. Protocols can describe both low-level and high-level details. A communication protocol allows you to specific or understand data communication without depending on detailed knowledge of a particular vendor’s network hardware. See also Internet Protocol, Transmission Control Protocol, and User Datagram Protocol.
RAM	Random Access Memory.
raw sockets	Provide access to the low-level protocol that supports sockets. This access to the underlying protocols makes raw sockets useful for development and testing. See also socket.
remote host	A computer system to which you are logged on over the network system.
server	A process that provides a specific service.
socket	An abstraction that allows application programs to access communication protocols by serving as an endpoint of a communication path within/between operating systems.
stream socket	Stream sockets imply a data stream is passed on the socket. When used, a stream socket is connected with another stream socket to form a two-way pipe across the network. See also socket.
telnet	Allows you to log on to a remote host. Once logged into a remote host, your terminal appears to be connected to that machine, and any keys you enter are automatically passed to the remote host. See also remote host.

Transmission Control Protocol (TCP)	A standard transport level protocol that is layered on top of the Internet Protocol (IP). TCP allows a process on one machine to send a stream of data to a process on another machine. TCP provides reliable, flow controlled, orderly, two-way transmission of data between connected processes. You can also shut down one direction of flow across a TCP connection, leaving a one-way (simplex) connection. See also protocol and Internet Protocol.
unconnected socket	A socket which has not been bound to a permanent destination. See also socket.
User Datagram Protocol (UDP)	A simple, unreliable datagram protocol that allows an application program on one machine to send a datagram to an application on another machine using the Internet Protocol (IP) to deliver datagrams. UDP uses a port number and an IP address to identify the endpoint of communication. See also datagram and protocol.

Notes

Index

ALLEN-BRADLEY
A ROCKWELL INTERNATIONAL COMPANY

As a subsidiary of Rockwell International, one of the world's largest technology companies — Allen-Bradley meets today's challenges of industrial automation with over 85 years of practical plant-floor experience. More than 13,000 employees throughout the world design, manufacture and apply a wide range of control and automation products and supporting services to help our customers continuously improve quality, productivity and time to market. These products and services not only control individual machines but integrate the manufacturing process, while providing access to vital plant floor data that can be used to support decision-making throughout the enterprise.

With offices in major cities worldwide

WORLD HEADQUARTERS **Allen-Bradley** **1201 South Second Street** **Milwaukee, WI 53204 USA** **Tel: (414) 382-2000** **Telex: 43 11 016** **FAX: (414) 382-4444**	**EUROPE/MIDDLE EAST/AFRICA HEADQUARTERS** Allen-Bradley Europa B.V. Amsterdamseweg 15 1422 AC Uithoorn The Netherlands Tel: (31) 2975/60611 Telex: (844) 18042 FAX: (31) 2975/60222	**ASIA/PACIFIC HEADQUARTERS** Allen-Bradley (Hong Kong) Limited Room 1006, Block B, Sea View Estate 28 Watson Road Hong Kong Tel: (852) 887-4788 Telex: (780) 64347 FAX: (852) 510-9436	**CANADA HEADQUARTERS** Allen-Bradley Canada Limited 135 Dundas Street Cambridge, Ontario N1R 5X1 Canada Tel: (519) 623-1810 FAX: (519) 623-8930	**LATIN AMERICA HEADQUARTERS** Allen-Bradley 1201 South Second Street Milwaukee, WI 53204 USA Tel: (414) 382-2000 Telex: 43 11 016 FAX: (414) 382-2400

Publication 1771-6.4.11 September 1992

PN 955113-15
Printed in USA