Using LAN Communications Pak

Version 3.3
Copyright and Publication Information

Copyright ©2000 Microware Systems Corporation. All Rights Reserved. Reproduction of this document, in part or whole, by any means, electrical, mechanical, magnetic, optical, chemical, manual, or otherwise is prohibited, without written permission from Microware Systems Corporation.

This manual reflects version 3.3 of LAN Communications Pak.

Revision: H
Publication date: July 2000

Disclaimer

The information contained herein is believed to be accurate as of the date of publication. However, Microware will not be liable for any damages including indirect or consequential, from use of the OS-9 operating system, Microware-provided software, or reliance on the accuracy of this documentation. The information contained herein is subject to change without notice.

Reproduction Notice

The software described in this document is intended to be used on a single computer system. Microware expressly prohibits any reproduction of the software on tape, disk, or any other medium except for backup purposes. Distribution of this software, in part or whole, to any other party or on any other system may constitute copyright infringements and misappropriation of trade secrets and confidential processes which are the property of Microware and/or other parties. Unauthorized distribution of software may cause damages far in excess of the value of the copies involved.

For additional copies of this software/documentation, or if you have questions concerning the above notice, please contact your OS-9 supplier.

Trademarks

OS-9, OS-9000, DAVID, and MAUI are registered trademarks of Microware Systems Corporation. SoftStax, FasTrak, UpLink, and Hawk are trademarks of Microware Systems Corporation. All other product names referenced herein are either trademarks or registered trademarks of their respective owners.

Address

Microware Systems Corporation
1500 N.W. 118th Street
Des Moines, Iowa 50325
515-223-8000
# Table of Contents

## Chapter 1: Networking Basics

<table>
<thead>
<tr>
<th>Page</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Basic Networking Terminology</td>
</tr>
<tr>
<td>10</td>
<td>Datagrams</td>
</tr>
<tr>
<td>13</td>
<td>Client and Server</td>
</tr>
<tr>
<td>14</td>
<td>Available Network Protocols</td>
</tr>
<tr>
<td>14</td>
<td>Internet Protocol (IP)</td>
</tr>
<tr>
<td>15</td>
<td>Transmission Control Protocol (TCP)</td>
</tr>
<tr>
<td>16</td>
<td>User Datagram Protocol (UDP)</td>
</tr>
<tr>
<td>17</td>
<td>Network Addressing</td>
</tr>
<tr>
<td>17</td>
<td>Network Classes</td>
</tr>
<tr>
<td>20</td>
<td>Subnet Masks</td>
</tr>
</tbody>
</table>

## Chapter 2: LAN Communications Pak Overview

<table>
<thead>
<tr>
<th>Page</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>Introduction</td>
</tr>
<tr>
<td>22</td>
<td>LAN Communications Pak Requirements</td>
</tr>
<tr>
<td>23</td>
<td>LAN Communications Pak Components</td>
</tr>
<tr>
<td>24</td>
<td>Applications</td>
</tr>
<tr>
<td>24</td>
<td>Application Programming Interfaces (APIs)</td>
</tr>
<tr>
<td>24</td>
<td>File Manager</td>
</tr>
<tr>
<td>25</td>
<td>Protocol Drivers</td>
</tr>
<tr>
<td>25</td>
<td>Device Drivers</td>
</tr>
<tr>
<td>26</td>
<td>Data Module</td>
</tr>
<tr>
<td>27</td>
<td>Software Description</td>
</tr>
<tr>
<td>34</td>
<td>LAN Communications Pak Architecture</td>
</tr>
</tbody>
</table>
Chapter 3: Serial Line Internet Protocol (SLIP) Driver

38 SPFLIP Introduction
38 Installation
41 spslip Transmission Process
41 Initialization
41 Sending Data
42 Reading Data
42 Header Compression
43 spslip Device Descriptor

Chapter 4: Point to Point Protocol (PPP)

46 Introduction to Point-to-Point Protocol
46 PPP Drivers and Descriptors
47 Protocol Drivers
49 Installation
51 PPP Transmission Process
54 PPP Device Descriptors
54 Override Default Settings
55 PPP makefile Descriptors
55 Rebuilding the Descriptor
56 Example: Changing the Baud Rate
57 Utilities
57 PPP Daemon Utility
62 PPP Modem Dialer Utility (chat)
70 Setting Up the Client Machine

Chapter 5: Protocol Drivers

76 SPF IP (spip) Protocol Driver
76 Data Reception and Transmission Characteristics
77 Default Descriptor Values for spip
78 Application Return Codes from API Calls
78 Considerations for Other Drivers
Chapter 6: BOOTP Server

107  Bootstrap Protocol
109  Server Utilities
112  bootptab Configuration File Setup
Using LAN Communications Pak
Chapter 1: Networking Basics

The Microware Local Area Network Communications Pak (LAN Communications Pak) enables your OS-9 system to communicate with other computer systems connected to a TCP/IP network. This enables you to send data, receive data, and log on to other computer systems.

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

This chapter covers the following topics:

- Basic Networking Terminology
- Available Network Protocols
- Network Addressing

Note
If you are already familiar with networking, you may want to proceed to the next chapters.
Basic Networking Terminology

A computer *network* is the hardware and software enabling computers to communicate with each other. Each computer system connected to the network is a *host*. There can be different types of host computers, and they can be (and usually are) located at different sites. For example, you may have your OS-9 system connected to a network consisting of other OS-9 systems, as well as UNIX and DOS systems.

An *internet* is the connection of two or more networks, using the Internet Protocol, enabling computers on one network to communicate with computers on another network. An internet is sometimes referred to as an *internetwork*.

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

*Figure 1-1 Gateway Connected Networks* illustrates two networks connected by a gateway.

**Figure 1-1 Gateway Connected Networks**

![Gateway Connected Networks Diagram]

**Datagrams**

When information (data) is passed from one host to another, either on the same network or across gateways, the data are carried in packets. Packets are the actual physical data transferred across the network...
layer. A *datagram* is a specific type of packet and is the basic unit of information passed on a network. In internetworking, this unit is called an Internet Protocol 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 content.

**Figure 1-2 Basic Datagram**

<table>
<thead>
<tr>
<th>Header Area</th>
<th>Data Area</th>
</tr>
</thead>
</table>

**Fragmentation**

The datagram size depends on the network’s *Maximum Transfer Unit* (MTU). Because each network may have a different MTU, hosts and routers divide large datagrams into smaller *fragments* when the datagram needs to pass through a network having a smaller 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 where 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*.

Hosts may also fragment large datagrams into multiple packets according to the size of the local MTU.

Fragments are reassembled at the final destination to produce a complete copy of the original datagram before it can be processed by the upper protocol layers.
Encapsulation

The addition of information to the datagram is called *encapsulation*. Datagrams are encapsulated with information as they pass through layers of the network. *Figure 1-3 Encapsulated Datagram Example* illustrates this concept.

*Figure 1-3 Encapsulated Datagram Example*

![Diagram showing encapsulation of data with TCP, IP, and Ethernet headers]

- **TCP Header**: Protocol = TCP
- **Initial Packet with 16-Bit TCP Source Port Number**
- **16-Bit TCP Destination Port Number**
- **IP Header**: Protocol = IP
- **Source**: IP 32-Bit Source Address
- **Destination**: IP 32-Bit Destination Address
- **Ethernet Frame**: Ethernet 48-Bit Source Address
- **Destination**: Ethernet 48-Bit Destination Address
- **CRC Trailer**: Ethernet 32-Bit CRC Trailer
Client and Server

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 requesting to use a service provided by a server.
Available Network 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 you need to be familiar with when using networks are:

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

Internet Protocol (IP)

Internet Protocol (IP) is the datagram delivery protocol. IP is a lower-level protocol located above the network interface drivers and below the higher-level protocols such as the UDP and the TCP. IP provides packet delivery service for higher level protocols such as TCP and UDP.

Due to the IP layer’s location, datagrams flow through the IP layer in two directions:

- Network up to user processes
- User processes down to the network

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.

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. If the computed checksum does not match the checksum in the header, the packet is discarded.
The IP layer also provides an addressing scheme. Every computer on an Internet receives one (or more) 32-bit address. This allows IP datagrams to be carried over any medium.

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

**Transmission Control Protocol (TCP)**

The Transmission Control Protocol (TCP) is layered on top of the IP layer. It is a standard transport level protocol allowing 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 the underlying datagram service is unreliable. Therefore, it performs a checksum of all data to help implement reliability. TCP uses IP host level addressing and adds a per-host port address. 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 as shown in Figure 1-4 IP Datagrams.

**Figure 1-4 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 allowing an application on one machine to send a datagram to an application on another machine using IP to deliver the datagrams. The important difference between UDP and IP datagrams is that UDP includes a protocol port number. This enables 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. Datagrams that arrive after the capacity of the queue has been reached are silently discarded.
Network Addressing

Regardless of which protocol you use to send messages across a network, each host is assigned one or more unique 32-bit internet addresses, or *IP addresses*.

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

Usually, all four parts of an address are specified. In this form, the most significant byte is written first and the least significant byte is written last.

Sometimes a written address may omit one or more of the four bytes. When this happens, the address is expanded to a normal four-part address by replacing the missing bytes with zeros. The following list demonstrates this process.

<table>
<thead>
<tr>
<th>Short Address</th>
<th>Expanded Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>127</td>
<td>0.0.0.127</td>
</tr>
<tr>
<td>127.1</td>
<td>127.0.0.1</td>
</tr>
<tr>
<td>127.1.2</td>
<td>127.1.0.2</td>
</tr>
</tbody>
</table>

Network Classes

The 32-bit address space is divided into five groups, or network classes. The first three classes consist of the unicast addresses assigned to hosts, the fourth class contains multicast addresses, and the fifth class is reserved for future use. In the first three classes, each 32-bit address is divided into two parts—the network and host portion. These identify the network the host is on and which host it is within that network.
Networking Basics

**Class A**

All addresses with a 0 as the first bit in their binary representation are considered Class A addresses.

The first byte represents the netid portion of the address, and the remaining 3 bytes represent the hostid. Class A addresses range from 0.0.0.0 to 127.255.255.255.

**Class B**

Addresses that start with a binary 10 are in Class B.

These addresses contain 2 bytes for each of the netid and hostid portions. Class B addresses range from 128.0.0.0 to 191.255.255.255.

**Class C**

Class C addresses are distinguished by 110 as the first three bits in their binary representation.

The first three bytes of the address form the netid, and the remaining byte is used for the hostid. Class C addresses range from 192.0.0.0 to 223.255.255.255.
Class D

Class D addresses are multicast addresses with the first 4 bits set to 1110.

```
0  1  8  16  24  31
   1 1 1 0
```

Class D addresses range from 224.0.0.0 to 239.255.255.255.

Class E

Class E addresses are reserved for future use and have 11110 as the first five bits in their binary representation.

```
0  1  8  16  24  31
   1 1 1 1 0
```

Class E addresses range from 240.0.0.0 to 247.255.255.255
Subnet Masks

The A, B, and C class distinctions are now mostly historical. Instead of using the first few bits of the address to determine the boundary between the netid and the hostid, a subnet mask is used. Wherever the bits in the subnet mask are set to 1, the corresponding bits in the address are part of the netid. Wherever the bits are zero, the corresponding address bits are part of the hostid. For example:

Address: 172.16.193.27  
Subnet mask: 255.255.255.0

netid: 172.16.193.0 (or just 172.16.193)  
hostid: 0.0.0.27 (or just 27)

While it is common to use byte boundaries for subnet masks, it is not required. Another example is:

Address: 172.16.193.27  
Subnet mask: 255.255.192.0 (18 bits)

netid: 172.16.192.0  
hostid: 0.0.1.27
Chapter 2: LAN Communications Pak
Overview

This chapter is an overview of the Local Area Network (LAN) Communications Pak. It includes:

• A list of example programs and utilities
• A list of software libraries provided
• A list of supported drivers and descriptors
• A diagram of the architecture and organization of the modules
Introduction

The LAN Communications Pak provides a small footprint Internet package that enables small embedded devices to communicate in a network environment. The User Datagram Protocol (UDP), Transmission Control Protocol (TCP), Internet Protocol (IP), as well as a raw socket interface are supported in this package.

Note

For information on Microware Systems communications software packages that carry UDP/TCP/IP protocols over other networks (such as ATM), contact your Microware Systems sales representative.

LAN Communications Pak Requirements

The LAN Communications Pak uses SoftStax, the Stacked Protocol File Manager (SPF), for its I/O system. SoftStax is required and provides a complete SPF environment for creating applications and drivers.
LAN Communications Pak Overview

LAN Communications Pak Components

LAN Communications Pak provides local area connectivity support for SoftStax™, the Microware integrated communications and control environment for OS-9. LAN Communications Pak adds device-level ethernet and serial interface capability using PPP and CSLIP.

Figure 2-1 shows the LAN Communications Pak components. Each software subsystem is defined in the following sections.

**Figure 2-1  LAN Communications Pak Components**

<table>
<thead>
<tr>
<th>LAN Communications Pak Components</th>
<th>SoftStax Components</th>
<th>Other Microware or Partner Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applications</td>
<td>Telnet/FTP Client/Server</td>
<td>DHCP Client</td>
</tr>
<tr>
<td>APIs</td>
<td>Socket Library</td>
<td>Netdb Library</td>
</tr>
<tr>
<td>Operating System</td>
<td>OS-9</td>
<td></td>
</tr>
<tr>
<td>File Manager</td>
<td>Stacked Protocol File Manager (SPF)</td>
<td></td>
</tr>
<tr>
<td>Protocol Drivers</td>
<td>TCP</td>
<td>UDP</td>
</tr>
<tr>
<td></td>
<td>IP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ATM</td>
<td>Ethernet</td>
</tr>
<tr>
<td>Device Drivers</td>
<td>ATM Devices</td>
<td>Ethernet Devices</td>
</tr>
<tr>
<td>Hardware</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Note: The diagram shows the components and their relationships in a visual format.*
Applications

Telnet
Telnet provides the user interface for communication between systems connected to the Internet and enables log-on to remote systems.

File Transfer Protocol (FTP)
FTP transfers files to and from remote systems.

Dynamic Host Configuration Protocol (DHCP)
DHCP enables a host to retrieve the network configuration from a server.

Application Programming Interfaces (APIs)
LAN Communications Pak supports the standard Berkeley Socket Library (socket.l) and network/host library netdb.l. Socket applications access the netdb.l library or netdb trap handler for network/host functions and for local or DNS client resolution. netdb locates configuration information in the inetdb data module or through a DNS client lookup.

File Manager
LAN Communications Pak plugs into the SoftStax™ environment underneath the Stacked Protocol File Manager (SPF). The tight network/OS integration of SPF enables the speed and efficiency crucial for maximizing throughput while minimizing footprint and CPU use over local and wide area networks.
LAN Communications Pak Overview

Protocol Drivers

LAN Communications Pak provides drop-in protocol drivers that can be stacked and unstacked as required by applications to communicate over LANs. These include the following:

- Transmission Control Protocol (TCP)—TCP provides reliable data transfer service over IP.
- User Datagram Protocol (UDP)—UDP provides datagram services over IP.
- Internet Protocol (IP)—IP provides Internet packet forwarding.
- Ethernet Protocol—spenet driver provides Ethernet connectivity, ARP request and reply layer for Ethernet hardware.
- Point-to-Point Protocol (PPP)—PPP supports IP over serial links.
- Serial Line Internet Protocol (SLIP)—SLIP supports IP over serial links.

For IP-based communications over Wide Area Networks (WANs), LAN Communications Pak can be combined with many wide area connectivity communications packages available for SoftStax™.

Device Drivers

LAN Communications Pak supports the following device drivers. This is not an exhaustive list. Not all of these drivers are available on all platforms, and some platforms have additional driver support not listed here. See the processor specific documentation for more information.

- AM7990
- NS83902
- NS83690/NS83790 (SMC Elite/SMC Ultra)
- DEC21040/21140/21143
- QUICC within Communications Processor Module (CPM).
- I82596
2 LAN Communications Pak Overview

- E509 (3COM EtherLink III, PCMCIA EtherLink III, and EtherLink XL)
- SMSC LAN91C94/91C96

## Data Module

LAN Communications Pak stores Internet configuration information in resident data modules with a prefix of `inetdb`. `inetdb` is a database containing Internet configurations for the local machine as well as the hosts, networks, protocols, and services that are available.

`inetdb` functions on a standard file system or can be configured to work in small embedded environments that require Internet connectivity. `inetdb` contains information on machine names, IP addresses and interfaces, and network protocol names and their identification values.

### For More Information

See Appendix A: Configuring LAN Communications Pak for information about the files that compose the `inetdb` data modules.
### Software Description

<table>
<thead>
<tr>
<th>Examples/Utilities</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>arp</td>
<td>Print and update the ARP table.</td>
</tr>
<tr>
<td>beam, target</td>
<td>Example UDP/IP socket program. Source and objects are provided.</td>
</tr>
<tr>
<td>bootpd</td>
<td>BOOTP server daemon.</td>
</tr>
<tr>
<td>bootpdc</td>
<td>BOOTP connection handler (bootpdc is forked by bootpd).</td>
</tr>
<tr>
<td>bootptest</td>
<td>Test the BOOTP server connection.</td>
</tr>
<tr>
<td>chat</td>
<td>Modem dialer utility used with PPP.</td>
</tr>
<tr>
<td>dhcp</td>
<td>Dynamic Host Configuration Protocol (DHCP) client for setting host networking parameter.</td>
</tr>
<tr>
<td>ftp</td>
<td>File Transfer Protocol (FTP) that handles sending and receiving files.</td>
</tr>
<tr>
<td>ftpd</td>
<td>FTP server daemon.</td>
</tr>
<tr>
<td>ftpdc</td>
<td>FTP server connection handler. (ftpdc is forked by ftpd or inetd.)</td>
</tr>
<tr>
<td>hostname</td>
<td>Prints or sets the string returned by the socket library.</td>
</tr>
<tr>
<td>Examples/Utilities</td>
<td>Purpose</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------</td>
</tr>
<tr>
<td>idbdump</td>
<td>Dumps the contents of the inetdb data module.</td>
</tr>
<tr>
<td>idbgen</td>
<td>inetdb data module generator for host, networks, protocols, services, DNS resolving gethostname() function, interfaces, and routes.</td>
</tr>
<tr>
<td>ifconfig</td>
<td>Interface configuration utility.</td>
</tr>
<tr>
<td>inetd</td>
<td>Internet Services Master Daemon; inetd can be configured to fork a particular program to handle data for a particular protocol/port number combination. For example, inetd can replace the ftptd and telnetd server daemons.</td>
</tr>
<tr>
<td>ipstart</td>
<td>Initializes the IP stack.</td>
</tr>
<tr>
<td>ndbmod</td>
<td>Adds, removes, or modifies information stored in the inetdb data module.</td>
</tr>
<tr>
<td>netstat</td>
<td>Reports network information and statistics.</td>
</tr>
<tr>
<td>ping</td>
<td>Sends ICMP ECHO_REQUEST packets to host.</td>
</tr>
<tr>
<td>pppauth</td>
<td>Utility for configuring PPP authentication.</td>
</tr>
<tr>
<td>pppd</td>
<td>Utility to initiate a PPP connection.</td>
</tr>
<tr>
<td>route</td>
<td>Add or delete entries from the routing table.</td>
</tr>
<tr>
<td>routed</td>
<td>Dynamic routing daemon.</td>
</tr>
</tbody>
</table>
Table 2-1  LAN Communications Pak Examples and Utilities (continued)

<table>
<thead>
<tr>
<th>Examples/Utilities</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>tcprecv, tcpsend</td>
<td>Example TCP/IP socket program. Source and objects are provided.</td>
</tr>
<tr>
<td>telnet</td>
<td>Telnet User Interface; telnet provides the ability to log on to remote systems.</td>
</tr>
<tr>
<td>telnetd</td>
<td>Telnet Server Daemon.</td>
</tr>
<tr>
<td>telnetdc</td>
<td>Telnet Server Connection Handler. (telnetdc forked by telnetd or inetd.)</td>
</tr>
<tr>
<td>tftpdc</td>
<td>TFTP Server Connection Handler. (tftpdc forked by tftp)</td>
</tr>
</tbody>
</table>

For More Information

Chapter 7: Utilities contains more detailed information about utilities and example applications.
Table 2-2  LAN Communications Pak Libraries

<table>
<thead>
<tr>
<th>Libraries</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>socket.l</td>
<td>Berkeley socket library.</td>
</tr>
<tr>
<td>netdb.l</td>
<td>Library for local or remote host name resolution and network/host functions. Uses the netdb trap handler.</td>
</tr>
<tr>
<td>netdb_dns.l</td>
<td>DNS client host name resolution and network functions. Does not use the netdb trap handler. The code is inlined in the application.</td>
</tr>
<tr>
<td>netdb_local.l</td>
<td>Local name resolution and network functions. Does not use the netdb trap handler. The code is inlined in the application.</td>
</tr>
<tr>
<td>ndblib.l</td>
<td>Library for inetdb data module manipulation.</td>
</tr>
</tbody>
</table>

For More Information
Refer to the LAN Communications Pak Programming Reference for detailed information concerning libraries.
### Table 2-3 LAN Communications Pak Descriptors and Drivers

<table>
<thead>
<tr>
<th>Drivers &amp; Descriptors</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>netdb_local</td>
<td>Local host name resolution module. No Domain Name Service (DNS) support.</td>
</tr>
<tr>
<td>netdb_dns</td>
<td>Local and remote host name resolution module. Client DNS (Domain Name Service) support.</td>
</tr>
<tr>
<td>spenet, enet</td>
<td>Driver and descriptor for Ethernet layer. Source files are provided for the descriptor.</td>
</tr>
<tr>
<td>spip, ip0</td>
<td>Driver and descriptor for IP protocol. Source files are provided for the descriptor.</td>
</tr>
<tr>
<td>spipcp, ipcp0</td>
<td>Drivers and descriptors for the Point-to-Point Protocol (PPP). Source files are provided for the descriptors.</td>
</tr>
<tr>
<td>splcp, lcp0</td>
<td>Drivers and descriptors for the Point-to-Point Protocol (PPP). Source files are provided for the descriptors.</td>
</tr>
<tr>
<td>sphdlc, hdlc0</td>
<td>Driver and descriptor for the Serial Line Internet Protocol device driver (SLIP). The driver and descriptor provide a point-to-point serial interface between serial connections for transmitting TCP/IP packets. Source files are provided for the descriptor.</td>
</tr>
<tr>
<td>spudp, udp0</td>
<td>Driver and descriptor for the UDP protocol. Source files are provided for the descriptor.</td>
</tr>
<tr>
<td>sptcp, tcp0</td>
<td>Driver and descriptor for the TCP protocol. Source files are provided for the descriptor.</td>
</tr>
</tbody>
</table>
Table 2-3  LAN Communications Pak Descriptors and Drivers (continued)

<table>
<thead>
<tr>
<th>Drivers &amp; Descriptors</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>spraw, raw0</td>
<td>Driver and descriptor for raw IP support. Source files provided for the descriptor.</td>
</tr>
<tr>
<td>sproute, route0</td>
<td>Driver and descriptor for IP routing domain support. Source files provided for the descriptor.</td>
</tr>
</tbody>
</table>

Note
The supported Ethernet drivers are listed below. Depending on the package you ordered, you may not receive all of them, or may receive additional drivers not listed here.

- sp162, sp167, sp172, sp177, spie0  Driver and descriptor for the I82596 ethernet driver. Source and object files are provided for the driver and descriptor for the MVME162/167/177 processors.
- sp403evb, spne0, sp360, spqe0  Driver and descriptor for the 403GAEVB—chip NS83902.
- sp147, sple0  Driver and descriptor for the AM7990 LANCE Ethernet driver. Source and object files are provided for the driver and descriptor for the MVME147.
### Table 2-3  LAN Communications Pak Descriptors and Drivers (continued)

<table>
<thead>
<tr>
<th>Drivers &amp; Descriptors</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>sp1603, spde0</td>
<td>Driver and descriptor for the DEC21040 Ethernet driver. Source and object files are provided for the driver and descriptor for the MVME1603.</td>
</tr>
<tr>
<td>sp821, spqe0</td>
<td>Driver and descriptor for the QUICC Ethernet driver. Source and object files are provided for the 821ADS.</td>
</tr>
<tr>
<td>sp8390, sp83C790, spns0, spwd0</td>
<td>Driver and descriptor for the SMC Elite and SMC Ultra.</td>
</tr>
<tr>
<td>spe509, spe30, spe31</td>
<td>Driver and descriptor for the 3Com EtherLink III, PCMCIA EtherLink III, and EtherLink XL (10 Mbs).</td>
</tr>
</tbody>
</table>
LAN Communications Pak Architecture

The following figure shows the architecture and organization of the modules in the LAN Communications Pak. The example applications provided use the socket libraries (socket.l and netdb.l for network/host functions) to make standard BSD socket calls.
The `idbgen` or `ndbmod` utilities create the `inetdb` data module containing host/network and other configuration information.
The `ndbmod` utility allows dynamic `inetdb` generation for entries in the `inetdb` data module on the resident system.

For More Information

See Appendix A: Configuring LAN Communications Pak for more information about files that compose the `inetdb` data module.
Chapter 3: Serial Line Internet Protocol (SLIP) Driver

The SPF Serial Line Internet Protocol device driver (\texttt{spslip}) provides a point-to-point interface between serial connections for transferring IP packets.
**SPSLIP Introduction**

`spslip` is based on the RFC 1055 specification for SLIP. In addition, it supports multicasting and the Van Jacobson CSLIP protocol enhancements.

`spslip` is typically used as an Internet interface to SCF devices (generally an RS-232 serial port) to perform telnet and FTP sessions and other communication functions.

The following table lists the driver and descriptor provided for SLIP:

<table>
<thead>
<tr>
<th>Table 3-1 <code>spslip</code> Driver and Descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Driver</td>
</tr>
<tr>
<td>-----------------</td>
</tr>
<tr>
<td><code>spslip</code></td>
</tr>
</tbody>
</table>

**Installation**

To install the `spslip` device driver on your system, perform the following steps:

**Step 1.** Setup the `inetdb` and `inetdb2` data files.

**Note**
You may also use the `ifconfig` utility to add the slip interface after starting IP.
Serial Line Internet Protocol (SLIP) Driver

For More Information
Refer to Chapter 7: Utilities and Appendix A: Configuring LAN Communications Pak for more information about setting up the inetdb files.

Step 2. Modify the spf_desc.h file to set baud rate, stop bits, and parity. This file is located in:

MWOS/<OS>/<PROCESSOR>/PORTS/PROTOCOLS/SPF/SLIP/DEFS

Step 3. Remake the descriptor sps10.

MWOS/<OS>/<PROCESSOR>/PORTS/PROTOCOLS/SPF/SLIP
os9make

After running the make, the descriptor binary can be found in the directory:

MWOS/<OS>/<PROCESSOR>/PORTS/PROTOCOLS/CMDS/BOOTOBJS/SPF

The driver spslip can be found in the directory:

MWOS/<OS>/<PROCESSOR>/CMDS/BOOTOBJS/SPF

Step 4. Load the modules spslip, sps10, inetdb, inetdb2, and sysmbuf either manually or with loadspf script. loadspf is located in MWOS/SRC/SYS.

Step 5. On disk-based systems, initialize the protocol stack by hand or with the startspf script (see startspf example), startspf is located in MWOS/SRC/SYS. Otherwise, enter the following commands:

• mbindinstall
• ipstart

Step 6. Verify that everything is working correctly by attempting to ping to the remote host.
Note

If you have more than one SLIP connection, make sure each connection has a different port address (PORTADDR in the driver descriptor configuration section).

- Compression is not negotiated and is ON by default.
- The default serial device used is /t1.
- Baud rate is 19200 by default.
- Baud rates, parity, compression, and MTU must match on both ends.
**spslip Transmission Process**

The `spslip` driver processes the data packets it sends and receives in four ways, including:

- **Initialization**
- **Sending Data**
- **Reading Data**
- **Header Compression**

### Initialization

During initialization, the driver acquires the input and output device names to be opened from the device descriptor. After opening the input and output devices, it sets the options for the input and output paths such as baud rate, echo, and X-ON/X-OFF. `spslip` then creates two processes called `spslip`. These processes control the data flow from the input and output paths.

To see the two processes, enter a `procs -e` command after starting up the system. If both processes are not running for every SLIP port in the system, the SLIP driver did not initialize successfully.

### Sending Data

When data is sent over the serial line, `spslip` checks to see if the amount of data being sent is less than the protocol's Maximum Transmission Unit (MTU). The default MTU is 1006, as defined in RFC 1055. To change the MTU, update the `spf_desc.h` file and remake `spsl0`.

The sending output process actually performs the SLIP data packaging. Basically, SLIP defines two special characters:

- **END** (octal 300).
- **ESC** (octal 333).
The output process starts sending the data over the serial line. When a data byte is detected to be the END character, the process replaces it with the ESC and octal 334 for transmission. If the data byte is the same as the ESC character, ESC and octal 335 are sent instead. When the last character of the packet has been sent, an END is sent.

**Reading Data**

When data is received, `spslip` calls an entry point to push data up to SPF to perform the read. The input process reverses the procedures of the far end SLIP transmitter. When the END character is detected, the input process places the packet in the receive queue.

**Header Compression**

The macro `COMPRESS_FLAG` turns header compression on and off. To turn compression on (the default), in `spf_desc.h`, use:

```
#define COMPRESS_FLAG(1)
```

To turn compression off, in `spf_desc.h`, use:

```
#define COMPRESS_FLAG(0).
```
The `spslip` device descriptor is created by updating the `spf_desc.h` file in the directory:

```
MWOS/<OS>/<PROCESSOR>/PORTS/PROTOCOLS/SPF/SLIP/DEFS
```

The following is an example of configurable sections of `spf_desc.h`:

```c
**Device Descriptor for SPF Slip driver
/****************************************************************************
/** Device Descriptor for the SPF slip driver (spslip) ****
/** ***********************************************************/
/** This section contains the configurable parameters ****
/****************************************************************************
#define MAXSLIPMTU SLIPMTU
#define PORTADDR 0
/
** Name of serial device */
#define I_DEV_NAME "/t1"
#define O_DEV_NAME "/t1"
/
** serial data format */
#define SL_RCV_BUF_SIZ 4096 /* input raw receive buffer */
#define SL_FAR_BITS 0x00 /* parity/stopbits */
#define SL_BAUD_RATE 0x10 /* baud rate */
/
** These values come from the ISP slip driver and are used to
** set the buffer sizes used by spslip */
#define OUTBUFSIZE SLIPMTU * 2 + 36
#define INBUFSIZE SLIPMTU + 32
/
** thread priority */
#define SL_IN_PRIOR 128
#define SL_OUT_PRIOR 128
/*****************************/
```
The previous example makes the `spslip` driver's descriptor `/sps10` which uses `/t1` as its port.

### Table 3-2 Baud Rates

<table>
<thead>
<tr>
<th>Baud Rate</th>
<th>OS-9 for 68K Value</th>
<th>OS9 Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>9600</td>
<td>0x0e</td>
<td>0x0f</td>
</tr>
<tr>
<td>19200 (default)</td>
<td>0x0f</td>
<td>0x10</td>
</tr>
</tbody>
</table>

**For More Information**

See the *OS9 for 68K Technical I/O Manual* for 68K processors or the *OS9 Device Descriptor and Configuration Module Reference* for all other processors.
Chapter 4: Point to Point Protocol (PPP)

The Point-to-Point Protocol (PPP) device driver provides a point-to-point interface between serial connections for transferring TCP/IP packets.
Introduction to Point-to-Point Protocol

The PPP device driver provides a point-to-point serial interface (as described in RFC-1661 and RFC-1662) between serial connections for transferring TCP/IP packets. The PPP device driver provides PPP functionality in the Stacked Protocol File (SPF) manager environment.

The following PPP topics are discussed:

- Installation
- Transmission process
- Device descriptors
- Utilities

PPP is typically used as an Internet interface to SCF devices—generally on an RS-232 serial port to perform telnet sessions, ftp sessions, and debugging capabilities.

PPP Drivers and Descriptors

The following is a list of drivers and descriptors provided for PPP:

Table 4-1  PPP Drivers and Descriptors

<table>
<thead>
<tr>
<th>Drivers</th>
<th>Descriptors</th>
</tr>
</thead>
<tbody>
<tr>
<td>sphdlc</td>
<td>hdlc0</td>
</tr>
<tr>
<td>spipcp</td>
<td>ipcp0</td>
</tr>
<tr>
<td>splcp</td>
<td>lcp0</td>
</tr>
</tbody>
</table>
Protocol Drivers

The SPF PPP protocol driver is implemented as a stack of three protocol drivers as follows:

- Internet Protocol Control Protocol (IPCP) driver
- Link Control Protocol (LCP) driver
- Link level High Level Data Link Control (HDLC) framer
Figure 4-1 shows how the drivers communicate with each other and the serial device.

Figure 4-1  PPP Data Flow
Utility Programs
Three utility program examples are provided to use the PPP driver:

Table 4-2 Utility Programs

<table>
<thead>
<tr>
<th>Program</th>
<th>Name</th>
<th>Source Code Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>PPP Daemon Utility</td>
<td>pppd</td>
<td>MWOS/SRC/SPF/PPP/UTILS/PPPD</td>
</tr>
<tr>
<td>Modem dialer utility</td>
<td>chat</td>
<td>MWOS/SRC/SPF/PPP/UTILS/CHAT</td>
</tr>
<tr>
<td>Authentication setup utility</td>
<td>pppauth</td>
<td>MWOS/SRC/SPF/PPP/UTILS/PPP_AUTH</td>
</tr>
</tbody>
</table>

Installation
To install PPP on your system, perform the following steps:

Step 1. Set up inetdb and inetdb2.

Note
You may also use the ifconfig utility to add the PPP interface after starting IP
For More Information
Refer to Chapter 7: Utilities and Appendix A: Configuring LAN Communications Pak for more information about the inetdb and inetdb2 data modules.

Step 2. Modify the spf_desc.h file for the HDLC descriptor if necessary.
Descriptor information such as baud rate can be set up in this file.
spf_desc.h is located in the following directory:
MWOS/<OS>/<PROCESSOR>/PORTS/PROTOCOLS/SPF/PPP/HDLC/DEFS

Step 3. Modify the spf_desc.h file for the IPCP descriptor if necessary.
spf_desc.h is located in the following directory:
MWOS/<OS>/<PROCESSOR>/PORTS/PROTOCOLS/SPF/PPP/IPCP/DEFS

Step 4. Modify the spf_desc.h file for the LCP descriptor if necessary.
spf_desc.h is located in the following directory:
MWOS/<OS>/<PROCESSOR>/PORTS/PROTOCOLS/SPF/PPP/LCP/DEFS

Step 5. Remake the appropriate descriptors hdlc0, lcp0, ipc0.
For each, run os9make
The directories in which the makefiles can be found are:
MWOS/<OS>/<PROCESSOR>/PORTS/PROTOCOLS/SPF/PPP/HDLC
MWOS/<OS>/<PROCESSOR>/PORTS/PROTOCOLS/SPF/PPP/IPCP
MWOS/<OS>/<PROCESSOR>/PORTS/PROTOCOLS/SPF/PPP/LCP

Step 6. Load the system modules, inetdb and inetdb2 and start SPF. (For an example, review the startspf script.) Load the PPP system modules either manually or with the loadspf file. loadspf is located in MWOS/SRC/SYS.

Step 7. Start the system. If disk-based, use startspf in MWOS/SRC/SYS. Otherwise, enter the following commands:
• mbinstall
• ipstart
• pppd /hdlc0/lcp0/ipcp0&

### PPP Transmission Process

#### Initialization

During initialization, the following events occur:

1. The driver acquires the input and output device names to be opened from the device descriptor.

2. After opening the input and output devices, the driver sets the options for the input and output paths such as the baud rate.

3. `sphdlc` creates the processes `hdlc_rx` and `hdlc_tx`, which control input and output data flow, respectively.

4. To view additional processes, enter a `procs -e` command after initializing the PPP drivers to view additional processes. Refer to Figure 4-2 Data Flow through HDLC Framer.
Figure 4-2 Data Flow through HDLC Framer

Service Processing Function (SPF)→Higher Layer Drivers

memory buffer (mbuf) is used to send packets; invalid frames and non-HDLC data are discarded.

Event→dr_downdata

extracts data→TX Mbuf Queue

Serial Device

hdlc_tx→(formats PPP packets)

(transmits the data)

Signal→hdlc_rx

(checks for proper HDLC framing; strips the frame)
Sending Data

During the sending data process, the following events occur:
1. IP sends data to the IPCP driver, which checks to verify if the packet is IP, and compresses the TCP header if necessary.
2. The IPCP driver adds a protocol field to the front of the packet and passes it to the LCP driver.
3. The LCP driver passes the unprocessed packet to the HDLC driver.
4. The HDLC driver adds an HDLC frame to the packet and passes it to the transit queue. It also sends an event to the hdlc_tx.
5. The hdlc_tx (transmit thread) process extracts data from the circular transmit queue, formats it into PPP packets, and transmits the data through the serial device.

Reading Data

During the reading data process, the following events occur:
1. The hdlc_rx (receive thread) process reads data from the serial device and checks for proper HDLC framing.
2. Valid frames have the HDLC frame removed and the remaining packet is sent up to the SPF file manager in the form of an mbuf. Invalid frames and non-HDLC data are discarded.
3. The LCP driver examines the packet to check for an LCP message, and handles it accordingly.
4. Other packets are sent to higher level drivers according to the protocol field in the packet.
   Packets that cannot be delivered for any reason are discarded.
PPP Device Descriptors

The PPP device descriptors are modified by updating the `spf_desc.h` file in the directories:

- `MWOS/<OS>/<PROCESSOR>/PORTS/PROTOCOLS/SPF/PPP/HDLC/DEFS`
- `MWOS/<OS>/<PROCESSOR>/PORTS/PROTOCOLS/SPF/PPP/LCP/DEFS`
- `MWOS/<OS>/<PROCESSOR>/PORTS/PROTOCOLS/SPF/PPP/IPCP/DEFS`

The PPP device descriptors `hdlc0`, `lcp0`, and `ipcp0` contain the device settings for making a PPP connection. Default settings for each PPP device descriptor can be found in:

- `MWOS/SRC/DPIO/SPF/DRVR/SP_PPP/HDLC/defs.h`
- `MWOS/SRC/DPIO/SPF/DRVR/SP_PPP/LCP/defs.h`
- `MWOS/SRC/DPIO/SPF/DRVR/SP_PPP/IPCP/defs.h`

Note

Selectable values for device descriptor options are defined in `MWOS/SRC/DEFS/SPF/ppp.h`.

Override Default Settings

Do not modify `defs.h` to change settings. To override the default settings, add a new option definition to `spf_desc.h` in the following directories:

- `MWOS/<OS>/<PROCESSOR>/PORTS/PROTOCOLS/SPF/PPP/HDLC/DEFS`
- `MWOS/<OS>/<PROCESSOR>/PORTS/PROTOCOLS/SPF/PPP/LCP/DEFS`
- `MWOS/<OS>/<PROCESSOR>/PORTS/PROTOCOLS/SPF/PPP/IPCP/DEFS`
PPP makefile Descriptors

The makefile for each PPP descriptor can be found in the directories:

MWOS/<OS>/<PROCESSOR>/PORTS/PROTOCOLS/SPF/PPP/HDLC
MWOS/<OS>/<PROCESSOR>/PORTS/PROTOCOLS/SPF/PPP/LCP
MWOS/<OS>/<PROCESSOR>/PORTS/PROTOCOLS/SPF/PPP/IPCP

Rebuilding the Descriptor

Run the following command in each makefile directory to rebuild the descriptor:

os9make

The rebuilt descriptor modules can be found in the directory:

MWOS/<OS>/<PROCESSOR>/PORTS/PROTOCOLS/SPF/CMDS/BOOTOBS/SPF
Example: Changing the Baud Rate

To change the baud rate to 19200 for a generic PowerPC processor, complete the following steps:

**Step 1.** Baud rate is set in the HDLC descriptor (see MWOS/SRC/DPIO/SPF/DRVR/SP_PPP/HDLC/defs.h). Therefore, modify spf_desc.h in MWOS/OS9000/PPC/PORTS/PROTOCOLS/SPF/PPP/HDLC/DEFS.

**Step 2.** Add the line `#define BAUD_RATE BAUDRATE_19200` to the user modifiable section. This overrides the default baud rate set in defs.h.

**Step 3.** Save the file.

**Step 4.** Change directory to MWOS/OS9000/PPC/PORTS/PROTOCOLS/SPF/PPP/HDLC.

**Step 5.** Run os9make

**Step 6.** The rebuilt hdlc0 descriptor resides in MWOS/OS9000/PPC/PROTOCOLS/CMDS/BOOTOBS/SPF.
Utilities

The following sections detail the `pppd`, `chat`, and `pppauth` utilities provided with PPP.

**PPP Daemon Utility**

The PPP daemon utility (`pppd`) opens a connection to the PPP stack, then goes to sleep indefinitely. This `pppd` utility is capable of changing some of the device settings in the driver stack.

The source code for `pppd` is located in:

`MWOS/SRC/SPF/PPP/UTILS/PPPD`

**PPP Daemon Command Line Arguments**

The command line for the daemon program is:

```
pppd [<options>] <descriptor stack> [<options>]
```

The descriptor stack must be specified, but all other parameters are optional. The options for this program are:

- `-?` Print help text for the program.
- `-d` Read commands from data module.
- `-h` Show daemon script commands.
- `-v` Show progress information.
- `-z` Read script commands from `stdin`.
- `-z=<name>` Read script commands from file or data module.

Multiple options may be grouped together with a single hyphen. For example, you could use the command:

```
pppd -vz=script.pppd /hdlc0/lcp0/ipcp0&
```
pppd Script Commands

The following commands may be used in a pppd script. The commands may be used in any order. Each command must be on a separate line. Comments may be included in the file using a pound (#) sign and can be placed on a separate line or at the end of a command line. The commands are not case sensitive, but device names are used exactly as entered in the script.

Table 4-3  PPP Script Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>set baud[rate] &lt;rate&gt;</td>
<td>Set baud rate. &lt;rate&gt; must be one of the following values: 50, 75, 110, 134.5, 150, 300, 600, 1200, 1800, 2000, 2400, 3600, 4800, 7200, 9600, 19200, 31250, 38400, MIDI (MIDI not available for OS9 for 68K systems)</td>
</tr>
<tr>
<td>set flow &lt;RTS</td>
<td>XON&gt;</td>
</tr>
<tr>
<td>set ipcp accept_local</td>
<td>During IPCP negotiation, allow the peer to set the local address.</td>
</tr>
<tr>
<td>set ipcp accept_remote</td>
<td>During IPCP negotiation, allow the peer to set the remote address.</td>
</tr>
<tr>
<td>set ipcp cslot &lt;value&gt;</td>
<td>Set flag for compressing slot identification in TCP header compression.</td>
</tr>
<tr>
<td>set ipcp mslot &lt;value&gt;</td>
<td>Set maximum slot identification for TCP header compression.</td>
</tr>
</tbody>
</table>
Table 4-3  PPP Script Commands (continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>set ipcp proto[col]</td>
<td>Set the IP compensation protocol value = &lt;proto_num&gt;.</td>
</tr>
<tr>
<td>set ipcp defaultroute</td>
<td>Set the system IP default route to the address of the system at the other end of the link.</td>
</tr>
<tr>
<td>set mode &lt;option,option...&gt;</td>
<td>Set mode flags. Choose options from the following: nowait, passive, updata, modem, loopback, norxcomp, notxcomp, nopap, nochap, nopfc, noacfc. Options are described in Table 4-4 Mode Settings for HDLC, LCP, and IPCP Drivers.</td>
</tr>
<tr>
<td>set parity &lt;mode&gt;</td>
<td>Set parity mode. Choose &lt;mode&gt; from the following: None, Odd, Even, Mark, Space.</td>
</tr>
<tr>
<td>set rx accm &lt;value&gt;</td>
<td>Set RX-&gt;receive Async Control Character Map.</td>
</tr>
<tr>
<td>set rx acfc &lt;value&gt;</td>
<td>Set RX-&gt;receive Address/Control Field Compression flag.</td>
</tr>
<tr>
<td>set rx buffer &lt;value&gt;</td>
<td>Set RX-&gt;receive buffer size.</td>
</tr>
<tr>
<td>set rx device &lt;name&gt;</td>
<td>Set receive port device.</td>
</tr>
<tr>
<td>set rx mru &lt;value&gt;</td>
<td>Set RX-&gt;receive Max Receive Unit.</td>
</tr>
<tr>
<td>set rx pfc &lt;value&gt;</td>
<td>Set RX-&gt;receive Protocol Field Compression flag.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>set scr &lt;value&gt;</td>
<td>Set maximum number of Configure Request messages to be sent.</td>
</tr>
<tr>
<td>set stop[bits] &lt;bits&gt;</td>
<td>Set number of stop bits. Choose &lt;bits&gt; from the following: 1, 1.5, 2.</td>
</tr>
<tr>
<td>set str &lt;value&gt;</td>
<td>Set maximum number of Terminate Request messages to be sent.</td>
</tr>
<tr>
<td>set timeout &lt;value&gt;</td>
<td>Set number of seconds for timeout value.</td>
</tr>
<tr>
<td>set tx accm &lt;value&gt;</td>
<td>Set TX-&gt;transmit Async Control Character Map.</td>
</tr>
<tr>
<td>set tx acfc &lt;value&gt;</td>
<td>Set TX-&gt;transmit Address/Control Field Compression flag.</td>
</tr>
<tr>
<td>set tx block[size] &lt;value&gt;</td>
<td>Set maximum block size for transmit.</td>
</tr>
<tr>
<td>set tx device &lt;name&gt;</td>
<td>Set transmit port device.</td>
</tr>
<tr>
<td>set tx mru &lt;value&gt;</td>
<td>Set TX-&gt;transmit Max Receive Unit.</td>
</tr>
<tr>
<td>set tx pfc &lt;value&gt;</td>
<td>Set TX-&gt;transmit Protocol Field Compression flag.</td>
</tr>
<tr>
<td>set word[size] &lt;size&gt;</td>
<td>Set word size. Select &lt;size&gt; from the following: 5, 6, 7, 8.</td>
</tr>
</tbody>
</table>
**Mode Settings**

The mode setting controls the behavior of the HDLC, LCP, and IPCP drivers. It is a bitmask, and can have multiple values. The different values and their meanings are described in Table 4-4 Mode Settings for HDLC, LCP, and IPCP Drivers (symbols in parentheses indicate descriptor setting defined in **ppp.h**):

**Table 4-4 Mode Settings for HDLC, LCP, and IPCP Drivers**

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wait</td>
<td>(WAIT_FOR_MODEM) HDLC will delay coming up until the chat script executes port_ready on and finishes.</td>
</tr>
<tr>
<td>passive</td>
<td>(PASSIVE_OPEN) LCP and IPCP will not initiate the connection, but wait for a configuration request from the peer.</td>
</tr>
<tr>
<td>nowait</td>
<td>(NO_WAIT_ON_OPEN) LCP and IPCP will not wait for a lower layer to enable the I/O path.</td>
</tr>
<tr>
<td>nopap</td>
<td>(NO_PAP) LCP will not use PAP authentication.</td>
</tr>
<tr>
<td>nochap</td>
<td>(NO_CHAP) LCP will not use CHAP authentication.</td>
</tr>
<tr>
<td>nopfc</td>
<td>(NO_PFC) LCP will not use protocol field compression.</td>
</tr>
<tr>
<td>noacfc</td>
<td>(NO_ACFC) LCP will not use address/control field compression.</td>
</tr>
<tr>
<td>norxcomp</td>
<td>(NO_RX_COMPRESS) IPCP will reject compression configuration requests from the peer.</td>
</tr>
<tr>
<td>notxcomp</td>
<td>(NO_TX_COMPRESS) IPCP will not request compression during link negotiation.</td>
</tr>
</tbody>
</table>
PPP Modem Dialer Utility (chat)

The PPP modem dialer utility `chat` opens a connection to the PPP stack, places it into a special mode, then performs reads and writes. The data transmitted informs the device to perform the modem commands contained in the `chat` script specified on the command line.

The source code for `chat` is located in:

```
MWOS/SRC/SPF/PPP/UTILS/CHAT
```

The `chat` utility maintains a linked list of strings which, if encountered in a modem response, cause the program to initiate an abort sequence. By default, this list is empty. You may add strings to the list by placing the appropriate command in the `chat` script.

The `chat` utility also maintains a linked list of modem commands to be sent during the abort sequence described above. By default, this list is empty. You may add commands to the list by placing the appropriate command in the `chat` script.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>update</code></td>
<td><code>(XPARENT_UPDATA_OK) Allows a driver to be stacked below HDLC, and causes it to send data up unaltered.</code></td>
</tr>
<tr>
<td><code>loopback</code></td>
<td><code>(LOOPBACK_MODE) Notifies HDLC to loopback characters. Only useful for testing.</code></td>
</tr>
</tbody>
</table>

Table 4-4 Mode Settings for HDLC, LCP, and IPCP Drivers (continued)
PPP Modem Dialer Command Line Arguments

The command line for the modem dialer program is:

```
chat [<options>] <HDLC descriptor name> [<options>]
```

The HDLC descriptor name must be specified, but all other parameters are optional. The HDLC descriptor name must be the same as the one used when starting the daemon `pppd`. The options for this program are:

- `?-` Print help text for the program.
- `-c` Display control characters when displaying data.
- `-d` Read commands from data module.
- `-e` Echo characters received from modem.
- `-h` Show `chat` script commands.
- `-l=<name>` Specify logfile name.
- `-v` Show progress information.
- `-z` Read commands from `stdin`.
- `-z=<name>` Read commands from file or data module.

Multiple options may be grouped together with a single hyphen. For example, you could use the command:

```
chat -cevz=script.chat /hdlc0
```

Chat Script Commands

The following commands can be used in a `chat` script. They can be in any order.

- Each command must be on a separate line.
- Comments can be included in the file using a pound (#) sign.
- Comments can be placed on a separate line or at the end of a command line.
• The commands are not case sensitive, but modem data is used exactly as it appears in the script.
• Strings sent or received should be enclosed in single quotes.

**Table 4-5 Chat Script Commands**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>abort &lt;string&gt;</strong></td>
<td>Initiate abort sequence if the indicated string is received from the modem.</td>
</tr>
<tr>
<td><strong>exp_data &lt;hex data&gt;</strong></td>
<td>Similar to expect command, but compares incoming bytes as unsigned with the specified hexadecimal bytes.</td>
</tr>
<tr>
<td><strong>expect &lt;string&gt;</strong></td>
<td>Wait until the indicated string is received from the modem before proceeding with chat script.</td>
</tr>
<tr>
<td><strong>if_abort &lt;string&gt;</strong></td>
<td>Add the indicated string to the list of modem commands to be sent during the abort sequence. If abort strings are sent out in the order they were placed into the list.</td>
</tr>
<tr>
<td>**port_ready &lt;ON</td>
<td>OFF&gt;**</td>
</tr>
<tr>
<td><strong>retry &lt;count&gt;</strong></td>
<td>Set number of times to retry when an expect or send command times out. This may be specified as often as needed, resulting in different retry counts for various sections of the chat script.</td>
</tr>
</tbody>
</table>
**Table 4-5  Chat Script Commands (continued)**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>query &lt;prompt string&gt;</td>
<td>Prints the prompt on the terminal, then waits for user input. Characters typed are not echoed to the terminal. The string entered is then sent to the modem. This is useful for obtaining a password interactively.</td>
</tr>
<tr>
<td>quiet &lt;ON</td>
<td>OFF&gt;</td>
</tr>
<tr>
<td>send &lt;string&gt;</td>
<td>Send the indicated string to the modem.</td>
</tr>
<tr>
<td>show_data &lt;ON</td>
<td>OFF&gt;</td>
</tr>
<tr>
<td>timeout &lt;seconds&gt;</td>
<td>Set the value of the timeout timer. This can be specified as often as needed, resulting in different timeout periods for various sections of the chat script.</td>
</tr>
<tr>
<td>wait &lt;seconds&gt;</td>
<td>Pause for specified interval before proceeding with chat script.</td>
</tr>
</tbody>
</table>
chat script Example

```plaintext
# ppp.chat - chat script to log into server "ppp_server"
abort "login incorrect" # abort if username/password incorrect
abort "BUSY"           # abort if busy signal from modem
if_abort "+++ATH"      # this will hang up phone if we abort
if_abort "ATZ"         # this will reset modem if we abort
timeout 360            # three minute time limit otherwise we
                        # may hang if no answer from server!
send "ATDT9,555-1212"  # 9 to get outside line, comma for pause,
                        # then telephone number for server.
expect "login:"        # wait for login prompt.
send "ppp_client"      # our login name as a ppp client
expect "ssword"        # wait for password prompt
quiet ON               # Don’t print our password on the screen
send "ppp_password"    # our password – we could also use the
                        # "query" command here to get it from user
quiet OFF              # OK to print stuff again
expect "ast login:"    # other servers may send something else
port_ready ON          # if we get this far, our connection is
                        # ready!
# end of chat script
```

Troubleshooting Modem Settings for PPP

If your board uses a serial device that does not support hardware flow control (RTS/CTS), it may be necessary to turn off hardware flow control on your modem. One symptom is data not returning to your target board after a modem-to-modem connection is made.
Point to Point Protocol (PPP)

**pppauth**

Configure PPP Authentication

**Syntax**

```
pppauth <option>
```

**Options**

- **-a**
  Add mode: Add specified entry.
  `-c` or `-p` MUST be specified, along with `<ISP name>`, `<Auth ID>` and `<Secret>`.

- **-c**
  CHAP specifier: Operate on CHAP entry(ies).

- **-d**
  Delete mode: Delete specified entry.
  `-c` or `-p` MUST be specified, along with `<ISP name>`.

- **-f <num entries>**
  Free entries so `<num entries>` are available.

- **-h <ISP name>**
  Set current ISP name.

- **-i <Auth ID>**
  Used in Modify mode to specify new Auth ID.

- **-l**
  List mode: List specified entries. `-c` or `-p` may be specified.

- **-m**
  Modify mode: Modify specified entry. `-c` or `-p` must be specified, along with `<ISP name>` and parameters to change.

- **-n**
  New mode: Copy existing entry with new type.
  `-c` or `-p[ap]` MUST be specified, along with `-t[type]`.

- **-p**
  PAP specifier: Operate on PAP entry(ies).

- **-s <Secret>**
  Used in Modify mode to specify new Secret.

- **-t [CHAP | PAP]**
  Type specifier: Change type to CHAP or PAP.
  CHAP or PAP may be abbreviated C or P.

- **-v**
  Verbose mode. Show progress info.
**Description**

The `pppauth` utility creates a data module used by `splecp` during the link authentication process.

Prior to creating the authentication module, you must:

- obtain information about the peer or Internet Service Provider (ISP) on the other end of the link
- know the authentication method being used (for example CHAP)
- know the shared secret (or password)
- know your authentication ID (userid)

Choose a name for this connection, or use the name provided by the ISP. This name (ISP name) is used with `pppauth` to store the authentication method, authentication ID, and the secret. The `-a` option is used to add a new entry to the data module. It must be accompanied by `-c` or `-p` to indicate CHAP or PAP authentication. You can add multiple ISP entries.

Prior to making a PPP connection, use the `-h` option to select the ISP name to which you want to connect.

**Example pppauth Setting**

The ISP "cserve" uses CHAP authentication, and provides you with the userid "acct1304", and a password of "b5kosh". The ISP "webnet" uses PAP authentication, and provides you with the userid "webhead", and a password of "doc-oc". Use the following commands to add this information:

```
pppauth -a -c 'cserve' 'acct1304' 'b5kosh'
pppauth -a -p 'webnet' 'webhead' 'doc-oc'
```

<table>
<thead>
<tr>
<th>Step 1.</th>
<th>Prior to connecting to cserve, set it as the current ISP with:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>pppauth -h 'cserve'</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step 2.</th>
<th>Examine the current settings by using the <code>-l</code> option:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Point to Point Protocol (PPP)

```
pppauth -l
Current ISP name is 'cserve'.
Type ISP Name Authentication ID / Secret
CHAP 'cserve' 'acct1304' 'b5kosh'
PAP 'webnet' 'webhead' 'doc-oc'
```

Step 3. Once the authentication module (ppp_auth) has been created, it can be saved using the save command and loaded as part of the network startup procedure.
Setting Up the Client Machine

After the device descriptors are built, load the drivers and descriptors onto the client (target) machine, if different from the machine they were built on. Using ftp or some other file transfer mechanism, make sure the following files are loaded onto the machine to be used for the PPP client:

1. **Load** `pppd`, `pppauth`, and `chat` **from the directory:**
   
   MWOS/<OS>/<PROCESSOR>/CMDS

2. **Load** `sphdlc`, `splcp`, and `spipcp` **from the directory:**
   
   MWOS/<OS>/<PROCESSOR>/CMDS/BOOTOBS/SPF
   
   `sphdlc  splcp  spipcp`

3. **Load** `hdlc0`, `lcp0`, and `ipcp0` **from the directory:**
   
   MWOS/<OS>/<PROCESSOR>/PORTS/PROTOCOLS/CMDS/BOOTOBS/SPF

4. **Setup the** `inetdb` **and** `inetdb2` **files, making sure to include a PPP interface.**

**For More Information**

Refer to Chapter 7: Utilities for more information about the `inetdb` file.

5. **If necessary, add default route using the** `route` **command if the descriptors have not already been configured to do so.**
For More Information
Refer to Chapter 7: Utilities for more information about the route utility.

Prepare Chat Script
On the client (target) machine, prepare a chat script.

For More Information
See the PPP Modem Dialer Utility (chat) section discussed previously in this chapter.

Setup Authentication
If you are using authentication, create the ppp_auth module using the pppauth utility, or load the module previously created and saved.

For More Information
See the pppauth section above.

Start PPP Daemon Process

Step 1. On the client (target) machine, run the following command in the background:

```
pppd -v /hdlc0/lcp0/ipcp0 &
```
Point to Point Protocol (PPP)

Step 2. The daemon program prints status information.
For example:
Device/stack /hdlc0/lcp0/ipcp0 open, path = n

pppd also can read information from a file:
pppd -v=setup.pppd /hdlc0/lcp0/ipcp0 &

Run Chat Script

Step 1. To dial the modem and connect to the server, run the following
command on the client (target) machine:
chat -evz=ppp.chat /hdlc0

Step 2. You should see something similar to the following:

(status information)
Reading from file ppp.chat
send( ATDT9,5551212<CR> )
Expect( login: )
<NUL>ATDT9,5551212<CR>
<CR><LF>
CONNECT LAPM COMPRESSED<CR><LF>
<CR>
<CR><LF>
login:
Got expected string
send( ppp_login<CR> )
Expect( ssword: )
  ppp_login<CR><LF>
Password:
Got expected string
send( ********** )
Expect( ast login: )
  <CR><LF>
Last login:
Got expected string
Handling if_abort strings
Clearing abort strings
Port is ready.
Point to Point Protocol (PPP)

(chat done. Status code = 0

At this point, you should have a Point-to-Point connection with the server.

Step 3. If pppd was invoked with the -v option, you should see “pppd(n): I/O on device /ipcp0 is enabled.” This indicates the PPP link is active.
4 Point to Point Protocol (PPP)
Chapter 5: Protocol Drivers

This chapter provides information about the IP, RAW, ROUTE, TCP, UDP, and ethernet protocol drivers.
SPF IP (spip) Protocol Driver

The SPF IP (spip) protocol driver is an IP protocol implementation used in embedded systems requiring internet routing, gateway, fragmentation, and reassembly capabilities. The spip driver also contains support for ICMP control messages such as redirects, port and destination unreachable, time exceeded, and source quenches. It also responds to the ICMP_ECHO messages used by ping.

The following table lists the driver and descriptor provided for IP:

<table>
<thead>
<tr>
<th>Driver</th>
<th>Descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>spip</td>
<td>ip0</td>
</tr>
</tbody>
</table>

Data Reception and Transmission Characteristics

The spip driver receives incoming IP packets from the driver below it, and maps the protocol field in the IP header to the appropriate protocol driver above it. For transmission, spip sends the packet to the appropriate interface below it based on the routing tables it maintains. If non-IP packets, or IP packets without a corresponding protocol driver above spip are received, they are discarded.

The spip driver can send and receive packets from multiple SPF drivers below it. These drivers must support certain IP specific setstats that are described later in this chapter.

Upon reception, the packet is passed to the appropriate driver above spip based on the protocol field in the IP header. Protocol drivers above spip are tightly coupled to the internals of spip. Therefore, generic SPF drivers are not supported directly above spip. Additional protocol support can be done through the raw socket interface provided by spraw. If IP fragments are received they are reassembled back into the original packet before being delivered to higher layer protocols.
For transmission, spip passes the packet to the appropriate driver below it based on the destination IP address and the routing tables it maintains. If the packet is too large for the selected interface it is fragmented.

**Default Descriptor Values for spip**

ip0 is the only IP descriptor provided with the LAN Communications Pack. There should only be one ip0 descriptor for the machine. The following discussion explains how to configure this descriptor and change it by editing the spf_desc.h file in the SPIP directory:

MWOS/SRC/DPIO/SPF/DRVR/SPIP/DEFS/spf_desc.h

**How to Configure and Change the ip0 Descriptor**

Step 1. Edit/update spf_desc.h in the /DEFS directory.

Step 2. Change to the root /SPIP directory and run os9make.

This process creates an updated descriptor in the following directory:

MWOS/<OS>/<PROCESSOR>/CMDS/BOOTOBJS/SPF.

**For More Information**

Refer to the *Using SPF* manual for more information about the contents and usage of the spf_desc.h file.
**Other Default Settings**

You can configure the following variables found in the `spip` descriptor. All others should not be changed.

- **GATEWAY**—Defaults to 0, indicating the host should not forward packets. In this mode incoming IP packets with destination addresses that are not broadcasts and do not match any local interface addresses are discarded. If set to 1, `spip` attempts to forward the packet based on the current routing table. If no route to the destination exists, the packet is dropped.

- **DEFTTL**—This value is put into the TTL field of the IP header on all packets sourced from this host. It defaults to 64 and must be between 1 and 255.

- **MAXSOCKBUF**—This variable controls the maximum size of the send and receive buffers for a socket. The default for each protocol is set in the protocol's descriptor. The `SO_SNDBUF` and `SO_RCVBUF` socket options may be used to change the default to any value between 1 and `MAXSOCKBUF`.

- **IPOFFSET**—If interfaces are added dynamically to the system with large hardware header requirements, previously opened paths may not have enough header space allocated. This value indicates how much extra header space `spip` should request to avoid an unnecessary data copy in this case. The default is 16 bytes.

**Application Return Codes from API Calls**

The return codes for socket API functions are described along with the functions in the *LAN Programming Reference Manual*.

**Considerations for Other Drivers**

There are extra operations that `spip` performs at certain times with protocol drivers above and below it.
**Drivers above SPIP**

Only those protocol drivers (above `spip`) shipped with the LAN Communications Pak are supported. However, you can implement new protocols using the raw socket interface provided by `spraw`.

**Drivers below SPIP**

Drivers below `spip` are notified of their current IP addresses through the `SPF_SS_SETADDR` and `SPF_SS_DELADDR` setstats. `spip` can associate multiple addresses with the same interface.

When applications join and leave multicast groups, the appropriate interface is notified with an `IP_SS_IOCTL` setstat. This setstat contains a pointer to an `ifreq` structure which contains the address of the multicast group being joined or left.

When drivers below `spip` are opened (during `ipstart` or a `SPF_SS_ATTIF` setstat) an `SPF_GS_SYMBOLS` getstat is sent down the newly opened stack looking for a `bsd_if_data` symbol. If the driver does not implement this getstat, `netstat` will not print interface statistics for this interface.

When `spip` is transmitting data, it passes some additional information within the mbuf. If the packet should be sent as a link layer broadcast, the `M_BCAST` flag will be set in the mbuf's `m_flags` field. If the packet is not a broadcast, the IP address of the host to deliver it to is contained in the four bytes immediately preceding the mbuf data. This is a different address than the destination address in the IP header when the next hop is an intermediate gateway.

When `spip` receives data from drivers below through its `dr_updata` entry point the data must be 4-byte aligned. This ensures that the source and destination IP addresses may be accessed as 4-byte integers.

If a driver below `spip` supports multiple interfaces (such as `spenet`), it must set `lu_pathdesc` in the logical unit statics to the path descriptor associated with the interface that received the packet before passing it up to `spip`. Drivers that do not support multiple interfaces do not have to set `lu_pathdesc`. However, these will suffer a slight performance loss.
Note
For best performance, mbufs passed up to spip should have at least 12 unused bytes between the mbuf header and the start of data.

Getstats and Setstats above SPIP

This section provides details about getstats and setstats sent by applications and protocol drivers above spip.

**SPF_SS_ATTIF**

This setstat is used to dynamically add an interface to spip. It causes SPF to open a path to the specified protocol stack but it does not become usable until an address is added via IP_SS_IOCTL.

Example usage:

```c
#include <netdb.h>
#include <SPF/spf.h>
#include <net/if.h>

int s;
error_code error;
struct n_ifnet ifp;
struct spf_ss_pb pb;

s = socket(AF_INET, SOCK_RAW, 0); /* May also use SOCK_DGRAM or */
/* SOCK_STREAM */
strcpy(ifp.if_name, "enet0");
strcpy(ifp.if_stack_name, "/spde0/enet");
ifp.if_flags = IFF_BROADCAST; /* Initial interface status flags */
ifp.if_data.ifi_mtu = 0; /* Use the stacks TXSIZE for an MTU */
ifp.if_data.ifi_metric = 0; /* Not currently used */
pb.code = SPF_SS_ATTIF;
pb.param = &ifp;
pb.size = sizeof(struct n_ifnet);
pb.updir = SPB_GOINGDWN;
error = _os_setstat(s, SS_SPF, &pb);`
**IP_SS_IOCTL**

This setstat implements the UNIX style interface I/O controls such as add and delete addresses, get and set netmasks, broadcast addresses, destination addresses, flags, and retrieve the interface table.

This setstat can be used to add an IP address to an interface. Multiple addresses are supported by calling the **IP_SS_IOCTL** command **SIOCAIFADDR** more than once on the same interface. If the interface does not support more than one address or limits the number of addresses, the driver should return an error on the setstat. Example:

```c
#include <SPF/spf.h>
#include <SPF/spinet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <net/if.h>
#include <netinet/in.h>
#include <netinet/in_var.h>

int s;
error_code error;
struct in_aliasreq ifr;
struct bsd_ioctl ioctl_arg;
struct spf_ss_pb pb;

s = socket(AF_INET, SOCK_RAW, 0); /* May also use SOCK_DGRAM or */
/* SOCK_STREAM */
memset(&ifr, 0, sizeof(struct in_aliasreq));
strcpy(ifr.ifra_name, "enet0");
ifr.ifra_addr.sin_len = sizeof(struct sockaddr_in);
ifr.ifra_addr.sin_family = AF_INET;
ifr.ifra_addr.sin_addr = htonl(0xac1002e2);
ifr.ifra_broadaddr.sin_len = sizeof(struct sockaddr_in);
ifr.ifra_broadaddr.sin_family = AF_INET;
ifr.ifra_broadaddr.sin_addr = htonl(0xac10ffff);
ifr.ifra_mask.sin_len = sizeof(struct sockaddr_in);
ifr.ifra_mask.sin_family = AF_INET;
ifr.ifra_mask.sin_addr = htonl(0xffff0000);
ioctl_arg.cmd = SIOCAIFADDR;
ioctl_arg.arg = &ifr;
pb.code = IP_SS_IOCTL;
pb.param = &ioctl_arg;
pb.size = sizeof(struct bsd_ioctl);
pb.updir = SPB_GOINGDWN;
error = _os_setstat(s, SS_SPF, &pb);
```
Other Supported SPF_SS_IOCTL Commands

Most of the other supported SPF_SS_IOCTL commands are used similarly although the argument passed is a pointer to an ifreq structure rather than an in_aliasreq structure.

The following list consists of the commands that take the parameter described in the previous paragraph.

- **SIOCGIFCONF**  Get list of all interfaces.
- **SIOCGIFADDR**   Get address of interface.
- **SIOCGIFNETMASK** Get netmask of interface.
- **SIOCGIFDSTADDR** Get point-to-point address.
- **SIOCGIFBRDADDR** Get broadcast address of interface.
- **SIOCGIFFLAGS**  Get interface flags.
- **SIOCSIFADDR**   Set address of interface.
- **SIOCSIFNETMASK** Set netmask of interface.
- **SIOCSIFDSTADDR** Set point-to-pint address.
- **SIOCSIFBRDADDR** Set broadcast address of interface.
- **SIOCSIFFLAGS**  Get interface flags.
The following example deletes an IP address from an interface.

Example usage:

```c
#include <SPF/spf.h>
#include <SPF/spinet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <net/if.h>
#include <netinet/in.h>

int s;
error_code error;
struct ifreq ifr;
struct bsd_ioctl ioctl_arg;
struct spf_ss_pb pb;
s = socket(AF_INET, SOCK_RAW, 0); /* May also use SOCK_DGRAM or */
/* SOCK_STREAM */
memset(&ifr, 0, sizeof(struct ifreq));
ifr.ifr_name = "enet0";
ifr.ifr_addr.sa_len = sizeof(struct sockaddr_in);
ifr.ifr_addr.sa_family = AF_INET;
((struct sockaddr_in *)&ifr.ifr_addr)->sin_addr.s_addr = htonl(0xac1002e2);
ioctl_arg.cmd = SIOCDELIFADDR;
ioctl_arg.arg = &ifr;
pb.code = IP_SS_IOCTL;
pb.param = &ioctl_arg;
pb.size = sizeof(struct bsd_ioctl);
pb.updir = SPB_GOINGDWN;
error = _os_setstat(s, SS_SPF, &pb);
```
The interface table may also be retrieved using the `IP_SS_IOCTL` setstat.

**Example usage:**

```c
#include <SPF/spf.h>
#include <SPF/spinet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <net/if.h>
#include <netinet/in.h>

int s;
error_code error;
struct ifconf ifc;
struct bsd_ioctl ioctl_arg;
struct spf_ss_pb pb;
char buffer[256];
s = socket(AF_INET, SOCK_RAW, 0); /* May also use SOCK_DGRAM or */
/* SOCK_STREAM */

ifc.ifc_len = 256;
ifc.ifc_buf = buffer;
ioctl_arg.cmd = SIOCGIFCONF;
ioctl_arg.arg = &ifc;
pb.code = IP_SS_IOCTL;
pb.param = &ioctl_arg;
pb.size = sizeof(struct bsd_ioctl);
pb.updir = SPB_GOINGDWN;
error = _os_setstat(s, SS_SPF, &pb);
```
Getstats and Setstats below SPIP

This section provides details about the getstats and setstats sent by spip to drivers below it.

**SPF_SS_SETADDR**

Drivers that need to know their IP address (such as spenet for ARP processing) should implement the SPF_SS_SETADDR setstat. If the number of addresses is limited, an EOS_FULL error should be returned when the limit is reached. A driver that does not need to know the protocol addresses may return EOS_UNKSVC.

Example Usage:

```c
case SPF_SS_SETADDR: {
    struct sockaddr_in *sin = (struct sockaddr_in *)pb->param;
    error = add_ip_address(sin->sin_addr);
    return (error);
}
```

**SPF_SS_DELADDR**

If the SPF_SS_SETADDR setstat is supported to add addresses, the SPF_SS_DELADDR setstat must be supported to remove them. If an attempt is made to remove an unknown address, EADDRNOTAVAIL should be returned.

Example Usage:

```c
case SPF_SS_DELADDR: {
    struct sockaddr_in *sin = (struct sockaddr_in *)pb->param;
    error = del_ip_address(sin->sin_addr);
    return (error);
}
```
IP_SS_IOCTL

The IP_SS_IOCTL setstat is used to notify interfaces to join or leave a particular multicast group. If the interface wishes to limit the number of multicast groups joined, a EOS_FULL error should be returned when the threshold has been exceeded.

Example Usage:
```c
#include <sys/ioctl>
#include <net/if.h>
case IP_SS_IOCTL: {
    struct bsd_ioctl *arg_ptr;
    struct ifreq *ifr;

    arg_ptr = pb->param;
    ifr = arg_ptr->arg;
    switch(arg_ptr->cmd) {
        case SIOCADDMULTI:
            return (AddMulticast(ifr->ifr_addr));
        case SIOCDELMULTI:
            return (DeleteMulticast(ifr->ifr_addr));
    }
    return(EOPNOTSUPP);
}
```

SPF_GS_SYMBOLS

The SPF_GS_SYMBOLS getstat is used to retrieve the address of one or more symbols (variables) maintained by a driver. After obtaining this information, a program can examine the variable dynamically. This getstat assists porting programs that use the kvm_nlist or nlist functions on other systems, and relieves drivers of copying statistic information from driver space to user space.

When processing the SPF_GS_SYMBOLS getstat, a driver receives a pointer to an array of nlist structures. The nlist structure is defined in MWOS/SRC/DEFS/SPF/BSD/nlist.h as:
```c
struct nlist {
    char      *n_name;   /* symbol name */
    unsigned long n_value;  /* address/value of the symbol */
    unsigned char n_type;   /* type defines */
    unsigned char res[3];   /* reserved space */
};
```
You must initialize each `n_name` member to point to the name of the symbol to be retrieved, and zero the rest of the structure. On a successful return, the `n_type` member of each element found is non-zero (typically `N_ABS`). `n_value` contains the address of the symbol corresponding to the name you specified in `n_name`. The driver processes every member of the `nlist` array until it reaches an element where the `n_name` member is zero. After processing the getstat, the driver passes this getstat to the next driver in the path. This enables one system call to retrieve multiple symbols from multiple drivers.

The following code fragment shows how to use the `SPF_GS_SYMBOLS` getstat. It retrieves information for the two symbols `_ipstat` and `_rtstat` from `spip`.

```c
#include <nlist.h>
#include <spf.h>
struct nlist nl[5];     /* name list to be passed to IP */
struct nlist *nl_ptr;    /* pointer to walk through list after*/
spf_ss_pb pb;        /* SPF parameter block */
path_id path;

memset(nl, sizeof(nl), 0); /* zero list */
nl[0].n_name = "_ipstat"; /* first element to get IP statistics */
nl[1].n_name = "_rtstat"; /* second element to get routing stats */
pb.code = SPF_GS_SYMBOLS;
pb.size = sizeof(nl);    /* or # elements * sizeof(element) */
pb.param = nl;
```
pb.updir = SPB_GOINGDWN;
if (_os_open("ip0", FAM_READ, &path) == 0) {
    _os_getstat(path, SS_SPF, &pb);
    _os_close(path);
}
for (nl_ptr = nl; nl_ptr->n_name; nl_ptr++)
    if (nl_ptr->n_type) /* non-zero means symbol was found */
        printf("Address of %s is %X\n", nl_ptr->n_name,
               nl_ptr->n_value);

Note
To access the memory pointed to by n_value, a process must be
system state. User state processes must use _os_permit.

Possible return codes:

EOS_BPADDR The nlist pointed to by pb.param
failed _os_chkmem
Protocol Drivers

SPF RAW (spraw) Protocol Driver

The SPF RAW protocol driver (spraw) provides a standard raw socket interface to the IP layer.

The following table lists the driver and descriptor provided for RAW.

Table 5-2  SPRAW Driver and Descriptor

<table>
<thead>
<tr>
<th>Driver</th>
<th>Descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>spraw</td>
<td>raw0</td>
</tr>
</tbody>
</table>

Data Reception and Transmission Characteristics

The spraw driver handles input of all IP datagrams where the protocol field of the IP header is 1 (ICMP), 255 (RAW), or any other value that does not have a corresponding driver above spip. On reception of such a datagram, it is delivered to any application that has requested to receive that protocol number. If multiple processes have requested a particular protocol, the incoming mbuf is duplicated and a copy delivered to each path.

On transmission, spraw fills in the required IP header fields and passes the datagram to spip for delivery to the destination. If the IP_HDRINCL option is set for a path, the application has filled in the IP header and the datagram is passed to spip for delivery.
Default Descriptor Values for spraw

raw0 is the only spraw descriptor provided with the LAN Communications Pak. There should only be one raw0 descriptor for the machine. The following discussion explains how this descriptor is configured, and how you can change this descriptor by editing the spf_desc.h file in the SPRAW directory:

MWOS/SRC/DPIO/SPF/DRVR/SPRAW/DEFS

How to Configure and Change the raw0 Descriptor

Step 1. Edit/update spf_desc.h in the /DEFS directory.
Step 2. Change to the root /SPRAW directory and run os9make.
This process creates an updated descriptor in the following directory:
MWOS/<OS>/<PROCESSOR>/CMDS/BOOTOBJ/SPF.

For More Information
Refer to the Using SPF manual for more information about the contents and usage of the spf_desc.h file.

ITEM Addressing
The spraw driver does not support ITEM addressing.
Other Default Settings

The following variables are configurable in the raw0 descriptor. All others should not be changed.

- **READSZ**—If more than READSZ number of incoming data bytes are queued in the SPF read queue, an SPF_SS_FLOWON setstat is sent to spraw. This causes incoming data to be buffered in the receive socket buffer. When an application reads data and the queue length falls below READSZ, an SPF_SS_FLOWOFF setstat is sent, causing all buffered data in the socket buffer to be added to the SPF read queue. As a result, the maximum mbuf usage for a single path is \(2 \times TCPWINDOW + READSZ\). The default value for this variable is 4096 bytes.

- **RECVBUFFER**—The maximum amount of incoming data spraw will buffer after receiving an SPF_SS_FLOWON setstat before it starts silently dropping packets. The default value is 8K and may be changed on a per path basis using the SO_RCVBUF socket option.

- **SENDBUFFER**—Since spraw does not queue outgoing data, this variable only limits the maximum size of a single datagram that can be sent. The default value is 8K and may be changed on a per path basis using the SO_SNDBUF socket option.

Application Return Codes from API Calls

The return codes for socket API functions are described along with the functions in the *LAN Communications Pak Programming Reference*.

Consideration for Other Drivers

The spraw driver depends on functions located in spip and will only work on top of spip.
The sproute driver provides a BSD 4.4 style routing domain. This domain allows a process to send and receive routing messages with spip using the normal sockets API. A routing domain socket can be created by issuing the socket system call and specifying a family of AF_ROUTE and a socket type of SOCK_RAW.

The following table lists the driver and descriptor provided for sproute.

<table>
<thead>
<tr>
<th>Driver</th>
<th>Descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>sproute</td>
<td>route0</td>
</tr>
</tbody>
</table>

Data Reception and Transmission Characteristics

The routing domain enables an application to send a datagram containing an rt_msghdr structure to add, delete, or change routes within the system routing table.

An application reading from a routing domain socket receives datagrams containing rt_msghdr structures, indicating changes in the system routing table. It receives datagrams containing if_msghdr structures when interfaces go up and down, as well as ifa_msghdr structures when addresses are added to and deleted from the system.

Routing sockets do not require a connect or a bind. After creation they may be immediately written to and read from using the normal socket API calls.
Default Descriptor Values for sproute

route0 is the only routing domain descriptor provided with the LAN Communications Pak. There should only be one route0 descriptor for the machine. The following discussion explains how this descriptor is configured, and how you can change this descriptor by editing the spf_desc.h file in the SPROUTE directory:

MWOS/SRC/DPIO/SPF/DRVR/SPROUTE/DEFS

How to Configure and Change the route0 Descriptor

Step 1. Edit/update spf_desc.h in the /DEFS directory.
Step 2. Change to the root /SPROUTE directory and run os9make.
This process creates an updated descriptor in the following directory:
MWOS/<OS>/<PROCESSOR>/CMDS/BOOTOBJS/SPF.

For More Information
Refer to the Using SPF manual for more information about the contents and usage of the spf_desc.h file.

ITEM Addressing
The sproute driver does not support ITEM addressing.
Other Default Settings

- **READSZ**—The threshold of queued data that triggers SPF to initiate flow control. The default value is 4096 bytes. This limit is normally not reached since `sproute` is used for message passing and not bulk data transfer.

Application Return Codes from API Calls

The return codes for socket API functions are described along with the functions in the *LAN Communications Pak Programming Reference*.

Consideration for Other Drivers

The `sproute` driver depends on functions located in `spip` and thus only works on top of `spip`. 
SPF TCP (sptcp) Protocol Driver

The SPF TCP (sptcp) protocol driver provides reliable data transfer service over IP.

The following table lists the driver and descriptor provided for TCP:

Table 5-4  TCP Driver and Descriptor

<table>
<thead>
<tr>
<th>Driver</th>
<th>Descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>sptcp</td>
<td>tcp0</td>
</tr>
</tbody>
</table>

Data Reception and Transmission Characteristics

The sptcp driver receives incoming TCP packets from the spip driver and maps the TCP port and IP destination address to a particular path with matching socket address. If no matching path is found, a TCP reset is returned to the sender.

Upon transmission, sptcp repackages the data to the correct size for the transmitting interface, fills in the necessary header information, and passes the packet to spip for delivery to the destination.

Default Descriptor Values for sptcp

tcp0 is the TCP descriptor provided with the LAN Communications Pak. There should only be one descriptor for the machine. The following discussion explains how this descriptor is configured and how you can change the descriptor by editing the spf_desc.h file located in:

MWOS/SRC/DPIO/SPF/DRVR/SPTCP/DEFS
How to Configure and Change the tcp0 Descriptor

Step 1. Edit/update spf_desc.h in the /DEFS directory.

Step 2. Change to the root /SPTCP directory and run os9make.

This process creates an updated descriptor in the following directory:
MWOS/<OS>/<PROCESSOR>/CMDS/BOOTOBJ/SPF.

For More Information
Refer to the Using SPF manual for more information about the contents and usage of the spf_desc.h file.

ITEM Addressing
The sptcp driver does not support ITEM addressing.

Other Default Settings
The following variables are configurable in the tcp0 descriptor. All others should not be changed.

- **READSZ**—If more than READSZ number of incoming data bytes are queued in the SPF read queue, an SPF_SSFLOWON setstat is sent to sptcp. This causes incoming data to be buffered in the receive socket buffer. When an application reads data and the queue length falls below READSZ, an SPF_SSFLOWOFF setstat is sent, causing all buffered data in the socket buffer to be added to the SPF read queue. As a result the maximum mbuf usage for a single path is 2 * TCPWINDOW + READSZ. The default value for this variable is 4096 bytes.
Protocol Drivers

- **TCPWINDOW**—The maximum amount of incoming data sptcp will store in the socket receive buffer after receiving an SPF_SS_FLOWON setstat. The TCP window size advertised in ACK’s to the other end of the connection is equal to the space available in the receive buffer. The default value is 16K and can be changed on a per path basis using the SO_RCVBUF socket option.

- **SENDBUFFER**—The maximum transmit data sptcp buffers before blocking the writing process, or in the case of non-blocking I/O return an EWOULDBLOCK error. The default value is 20K and may be changed on a per path basis using the SO_SNDBUF socket option. The maximum amount of mbuf usage is SENDBUFFER + size of application write or 3 times the size of application write, whichever is larger. For example, if an application is passing 32K bytes of data to the write(), _os_write(), or send() system calls, it is possible to use 96K bytes of mbuf space.

- **TXSIZE**—This variable controls the maximum size of an mbuf SPF will request when packaging user data for transmission. The default value is 4380 bytes.

- **DFLT_MSS**—The maximum segment size for a connection. This is set to the MTU of the interface selected for transmission based on the routing table at the time the connection is established. DFLT_MSS is no longer used in the calculation, so changing it has no effect.

### Application Return Codes from API Calls

The return codes for socket API functions are described, along with the functions in the *LAN Communications Pak Programming Reference*.

### Considerations for Other Drivers

The sptcp driver depends on functions located in spip and only works on top of spip.
SPF UDP (spudp) Protocol Driver

The SPF UDP protocol driver (spudp) provides datagram service over IP.

The following table lists the driver and descriptor provided for UDP:

Table 5-5  UDP Driver and Descriptor

<table>
<thead>
<tr>
<th>Driver</th>
<th>Descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>spudp</td>
<td>udp0</td>
</tr>
</tbody>
</table>

Data Reception And Transmission Characteristics

The spudp driver receives incoming UDP packets from the spip driver below and maps the UDP port and IP address to a particular path with matching socket address. The spudp driver creates an address mbuf with each incoming packet and chains the data to it using the m_pnext field of the mbuf header. If multiple paths match, as can happen with wildcard addresses or multicasts, the incoming mbuf is duplicated and a copy sent to each path. If no match is found, an ICMP port unreachable error is returned.

On transmission, spudp fills in the necessary header information and passes the datagram to spip for delivery to the destination.

The amount of memory used by a single UDP connection is limited to avoid using all the available mbuf pool. When an application’s read queue grows beyond the READSZ specified in the udp0 descriptor, an SPF_SS_FLOWON setstat is generated. When spudp receives this, it buffers up to the number of bytes specified in the RECVBUFFER descriptor variable. If more data is received, the packets are silently dropped. On transmit, spudp does not buffer the data and the mbuf pool may be exhausted if the bottom layer hardware drivers queue an unlimited number of packets.
Default Descriptor Values for spudp

udp0 is the only UDP descriptor provided with the LAN Communications Pak. There should only be one udp0 descriptor for the machine. The following discussion explains how this descriptor is configured, and how you can change this descriptor by editing the spf_desc.h file in the SPUDP directory:

MWOS/SRC/DPIO/SPF/DRVR/SPUDP/DEFS

How to Configure and Change the udp0 Descriptor

Step 1. Edit/update spf_desc.h in the /DEFS directory.
Step 2. Change to the root /SPUDP directory and run os9make.
This process creates an updated descriptor in the following directory:
MWOS/<OS>/<PROCESSOR>/CMDS/BOOTOBS/SPF.

For More Information
Refer to the Using SPF manual for more information about the contents and usage of the spf_desc.h file.

ITEM Addressing
The spudp driver does not support ITEM addressing.
Other Default Settings

The following variables can be configured in the `udp0` descriptor. All others should not be changed.

- **CHECKSUM**—By default this is set to 1, causing a UDP checksum to be calculated for all transmitted datagrams. A value of 0 will disable checksums. This has no effect on received mbufs because their checksum, if present, is always checked.

- **READSZ**—If more than `READSZ` number of incoming data bytes are queued in the SPF read queue, an `SPF_SS_FLOWON` setstat is sent to `spudp`. This causes incoming data to be buffered in the receive socket buffer. When an application reads data and the queue length falls below `READSZ`, an `SPF_SS_FLOWOFF` setstat is sent, causing all buffered data in the socket buffer to be added to the SPF read queue. As a result, the maximum mbuf usage for a single path is `2 * RECVBUFFER + READSZ`. The default value for this variable is 4096 bytes.

- **RECVBUFFER**—The maximum amount of incoming data `spudp` buffers after receiving an `SPF_SS_FLOWON` setstat before it starts silently dropping packets. The default value is 32K and may be changed on a per path basis using the `SO_RCVBUF` socket option.

- **SENDBUFFER**—Since `spudp` does not queue outgoing data, this variable only limits the size of the largest UDP datagram that can be sent and defaults to 9216 bytes. This value can be changed on a per path basis using the `SO_SNDBUF` socket option.

Application Return Codes from API Calls

The return codes for socket API functions are described, along with the functions in the *LAN Communications Pak Programming Reference*.

Considerations for Other Drivers

The `spudp` driver depends on functions located in `spip` and, therefore, only works on top of `spip`. 

The `spenet` protocol driver sits between the hardware Ethernet drivers and `spip`. Its main function is to map between Ethernet addresses and IP addresses using the Address Resolution Protocol (ARP). See RFC-826 for additional details. The `spenet` driver also adds and removes Ethernet headers for outgoing and incoming packets.

`Spenet` maintains an ARP table for mapping between Ethernet and IP addresses. When `spenet` receives a unicast packet from `spip`, and the destination address has no entry in the ARP table, `spenet` broadcasts an ARP request to discover the Ethernet address. The results are then added to the ARP table, and are removed after 20 minutes. The `arp` utility can be used to view or modify the ARP table.

### Table 5-6 SPF Ethernet Driver and Descriptor

<table>
<thead>
<tr>
<th>Driver</th>
<th>Descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>spenet</td>
<td>enet</td>
</tr>
</tbody>
</table>

### Data Reception and Transmission Characteristics

The `spenet` driver receives incoming Ethernet packets from one or more drivers below it and maps the destination Ethernet address to the destination IP address. Before passing the packet to `spip`, the `lu_pathdesc` component of the logical unit statics is set to point to the appropriate path descriptor to enable `spip` to determine which interface received the packet.

For transmission, `spenet` adds the appropriate Ethernet hardware destination and source addresses to the packet. If the `M_BCAST` flag is set in the `mbuf` header, the packet is assumed to be a broadcast and the all 1's hardware broadcast address is used. If the `M_MCAST` flag is set the destination address in the IP header is converted to the appropriate link layer multicast address. In all other cases the ARP cache is searched using the IP address of the next hop destination. If
no ARP entry exists, the ARP protocol is initiated. When the Ethernet header has been completed, the packet is sent to an interface driver below.

**Default Descriptor Values for `spenet`**

`enet` is the only descriptor for `spenet` provided with the LAN Communications Pak. This descriptor is generic for all Ethernet drivers and should not need to be updated.

This descriptor is configured to change fields by editing the `spf_desc.h` file in the `SPENET` directory:

```
MWOS/SRC/DPIO/SPF/DRVR/SPENET/DEFS
```

**How to Configure and Change the `enet` Descriptor**

1. **Step 1.** Edit/update `spf_desc.h` in the `/DEFS` directory.
2. **Step 2.** Change to the root `/SPENET` directory and run `os9make`.

This process creates an updated descriptor in the following directory:

```
MWOS/<OS>/<PROCESSOR>/CMDS/BOOTOBJ/SPF.
```

**For More Information**

Refer to the *Using SPF* manual for more information about the contents and usage of the `spf_desc.h` file.

**ITEM Addressing**

The `spenet` driver does not support ITEM addressing.
Other Default Settings

The following variables can be configured in the enet descriptor. All others should not be changed.

MAXADDR_PER_IFACE—indicates the maximum number of protocol addresses that can be associated per hardware interface. The LAN Communications Pak supports more than one protocol address (IP) per hardware interface. The default value is 4.

TIMER_INT—spenet runs a cyclic timer that is used to remove old arp entries. This value defines the timer interval in seconds. The default value is 60.

KILL_C—If a completed (received arp reply) entry is not used in this many timer intervals it is deleted. The default value is 20.

KILL_I—If an entry remains incomplete (no arp reply received) for this many timer intervals it is deleted. The default value is 3.

Considerations for Other Drivers

Drivers Below spenet

When spip receives data from drivers below through its dr_updata entry point the data must be 4-byte aligned. This requirement ensures that the source and destination IP addresses may be accessed as 4-byte integers. When receiving data, the four bytes immediately preceding the mbuf data must contain a pointer to the device entry of the driver sending data up the stack.

Getstats for SPENET

The structures used for getstats and setstats for spenet are found in MWOS/SRC/DEFS/SPF/BSD/net/if_arp.h and MWOS/SRC/DEFS/SPF/BSD/netinet/if_ether.h. The spenet driver provides the following getstats to programmers.
For More Information
See the SPF Programming Reference Manual for additional details on \_os\_getstat and the spf\_ss\_pb structure.

**SPF\_GS\_ARPENT**
This setstat retrieves a particular entry from the ARP table. The `param` member of the `spf\_ss\_pb` structure must point to a user allocated `arptab` structure. The `at_iaddr` member of the `arptab` structure must be set to the IP address (in network order) of the entry to retrieve. On success, 0 is returned. EOS\_PNNF is returned if the entry cannot be found.

**SPF\_GS\_ARPTBL**
This setstat retrieves the entire ARP table. The `param` member of the `spf\_ss\_pb` structure points to a user allocated array of `arptab` structures. The size member must be set to the size of this array in bytes. On success, 0 is returned, indicating `spenet` copied as much of the table as possible to the users array, and set the size member of the `spf\_ss\_pb` to the actual size of the ARP table (in bytes).

It is recommended that you retrieve the ARP table using two getstats. The first getstat sets the size to zero. On return, size will indicate the size of the current ARP table. Dynamically allocate this much space (plus some additional space in case the table grows), and issue another getstat.

**ENET\_GS\_STATS**
This setstat retrieves the statistics maintained by `spenet`. The `param` member of the `spf\_ss\_pb` structure points to a user allocated `enet\_stat\_pb` structure. On success, 0 is returned.
Setstats for SPENET

The spenet driver provides the following setstats to programmers.

For More Information
See the SPF Programming Reference for additional details on _os_setstat and the spf_ss_pb structure.

To alter the ARP table, a process must have super user access.

SPF_SS_ADDARP
This setstat adds an entry to the ARP table. The param member of the spf_ss_pb structure points to a user allocated arprec structure. You must initialize the arp_pa (in network order), arp_ha, and arp_flags members of this structure. See the file if_arp.h for settings of arp_flags. The arp_pa member should be treated as a sockaddr_in structure, setting the sin_family to AF_INET. On success, 0 is returned.

SPF_SS_DELARP
This setstat removes an entry from the ARP table. The param member of the spf_ss_pb structure points to a user allocated arprec structure. You must set the arp_pa member to the IP address (in network order) of the entry to be deleted. This member should also be treated as a sockaddr_in structure, setting the sin_family to AF_INET. On success, 0 is returned. EOS_PNNF indicates the address was not found in the ARP table.
For More Information
See the `arp` command in Chapter 7: Utilities for additional details.
Chapter 6: BOOTP Server

The Bootstrap Protocol (BOOTP) enables booting from the network. BOOTP clients require a BOOTP server on the connected network to support the BOOTP protocol as specified in RFC 951 (Croft/Gilmore) and Trivial File Transfer Protocol (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).

This chapter covers the following topics:

• Overview of the BOOTP server
• BOOTP server utilities
• Setting up the bootptab configuration file

Note
The 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.
Bootstrap Protocol

Bootstrap Protocol (BOOTP) is a client-server protocol. The system being booted is the client. The process includes the following steps.

1. The client system makes requests to a server system on the network or the same VME chassis over the backplane. The server or the client may or may not be an OS-9 system. OS-9 clients request the server to identify the following:
   - Client IP address
   - Server IP address
   - Path to the bootfile
   - Size of the bootfile

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

3. The ROM boot code starts the network boot option (BOOTP) either through the menu selection or automatically without operator intervention. The client broadcasts the BOOTP request containing the client’s hardware address (for example, Ethernet address) retrieved from SRAM. A server responds with the information listed above.

Note

You can adjust the number of contact attempts the client makes to a server by editing the config.des file in the following directory:

```
mwos\OS9000\<PROCESSOR>\PORTS\<TARGET>\ROM\CNFGDATA
```

You must add the line `maxbootptry=<number>` to the “eb” section of the file. `<number>` can be from 1 to infinity. The default is 8.
4. 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 client reads the bootfile as it is transferred across the network and copies it into local RAM in the same manner as other boot device drivers.

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

**Server Utilities**

A BOOTP server includes the `bootptab` configuration file and the utility programs identified in Table 6-1 **BOOTP Server Utilities**.

**Table 6-1  **BOOTP Server Utilities**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bootpd</td>
<td>Responds to BOOTP client requests with BOOTP server responses.</td>
</tr>
<tr>
<td>bootptest</td>
<td>A simple utility to test bootpd server response.</td>
</tr>
<tr>
<td>tftpdc</td>
<td>Responds to tftp read requests and forks tftpdc to handle the transfer.</td>
</tr>
<tr>
<td>tftpdc</td>
<td>Reads a bootfile for a client using the TFTP protocol.</td>
</tr>
</tbody>
</table>
For More Information
Refer to Chapter 7: Utilities for more information about these utilities.

Utility programs are located in the /MWOS/CMDS directory. The bootptab configuration file is usually located in the TFTPBOOT directory copied from MWOS/SRC/TFTBOOT.

The procedure for starting a BOOTP server and the associated utilities:

```
tftpd <<>/nil &
bootpd /h0/TFTPBOOT/bootptab<<>/nil &
```

The boot file name is dependent on the client BOOTP system. On OS-9, the boot file is called OS9boot.<hostname>. The OS9boot.<hostname> file should have public read permissions set to allow tftpd to access it. While in the /h0/TFTPBOOT directory, use the following command to turn on the public read permissions for all OS9boot files:

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

The 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.
 */
When `bootpd` is first started, it performs the following functions:

1. Reads a configuration file to build an internal database of clients and desired boot responses for each.
2. Listens for BOOTP boot requests on UDP socket port 67 (`bootps`).
3. Checks the file time stamp on the configuration file before processing a boot request. If the file time stamp changed since the last check, the client database is rebuilt.

The configuration file has a format similar to `termcap` in which two-character, case-sensitive tag symbols represent host parameters. These parameter declarations are separated by colons (`:`). The general format for the `bootptab` file 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 Boolean form with no value (`:tg:`).

`bootpd` recognizes the following tags.

**Table 6-2 Bootp Tags**

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bf</code></td>
<td>Bootfile</td>
</tr>
<tr>
<td><code>bs</code></td>
<td>Bootfile size in 512-octet (byte) blocks</td>
</tr>
<tr>
<td><code>ha</code></td>
<td>Host hardware address</td>
</tr>
<tr>
<td><code>hd</code></td>
<td>Bootfile home directory</td>
</tr>
<tr>
<td><code>hn</code></td>
<td>Send hostname</td>
</tr>
</tbody>
</table>
There is also a generic tag, Tn, where n is an RFC-1048 vendor field tag number. This enables immediate use of future extensions to RFC-1048 without first modifying bootpd. Generic data may be represented 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).

### Hardware Type

The ht tag specifies the hardware type code as:

- Unsigned decimal, octal, or hexadecimal integer
- ethernet or ether for 10Mb Ethernet
Address

The `ha` tag takes a hardware address. The hardware address must be specified in hexadecimal. You can 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`).

Host Name, Home Directory, and Bootfile

The host name, home directory, and bootfile are ASCII strings which can optionally be 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 a reply is not sent.
- If the client specifies a relative path name, a full path name is formed by appending 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 BOOTP clients normally supply `os9boot` as the bootfile name, the relative path name case is used. OS-9 BOOTP clients normally supply `sysboot` as the bootfile name.

Clients specifying null boot files 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, in addition to actually being present, the public read access bit of the file must be 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 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.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>No</td>
<td>Null</td>
<td>Send null file name</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
<td>Relative</td>
<td>Discard request</td>
</tr>
<tr>
<td>No</td>
<td>Yes</td>
<td>Null</td>
<td>Send if absolute else discard request</td>
</tr>
<tr>
<td>No</td>
<td>Yes</td>
<td>Relative</td>
<td>Discard request</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
<td>Null</td>
<td>Send null file name</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
<td>Relative</td>
<td>Lookup with <code>.host</code></td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Null</td>
<td>Send home/boot or bootfile</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Relative</td>
<td>Lookup with <code>.host</code></td>
</tr>
</tbody>
</table>

**Bootfile Size**

The bootfile size, `bs`, may be 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 Boolean has the same effect as specifying *auto* as its value. OS-9 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 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 does 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 fits, 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.

---

**Note**

*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 the 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 new lines. 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.

**bootptab File Example**

An example `/h0/TFTPBOOT/bootptab` file follows:

```plaintext
# 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:
```
6  BOOTP Server

Using LAN Communications Pak
Chapter 7: Utilities

This chapter examines the LAN Communications Pak utilities provided with this package.
## Overview

The following utilities are provided with the LAN Communications Pak.

### Table 7-1  LAN Communications Pak Utilities

<table>
<thead>
<tr>
<th>Utility</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>arp</strong></td>
<td>Print and update the ARP table.</td>
</tr>
<tr>
<td><strong>bootptest</strong></td>
<td>Test the bootpd and tftpd daemons.</td>
</tr>
<tr>
<td><strong>dhcp</strong></td>
<td>DHCP client negotiation utility.</td>
</tr>
<tr>
<td><strong>ftp</strong></td>
<td>File Transfer Protocol. ftp transfers files to and from remote systems. There are many ftp commands for file manipulation between systems.</td>
</tr>
<tr>
<td><strong>hostname</strong></td>
<td>Prints or sets the string returned by the socket library gethostname() function.</td>
</tr>
<tr>
<td><strong>idbgen</strong></td>
<td>Internet Database Generation. idbgen builds the internet data module from the data files: host.conf, hosts, hosts.equiv, inetd.conf, networks, protocols, resolv.conf, interfaces.conf, routes.conf, services, drpw, idbgen must be run each time any of these files are updated.</td>
</tr>
<tr>
<td><strong>idbdump</strong></td>
<td>Internet Database Display. idbdump dumps the current entries in the internet data module (inetdb).</td>
</tr>
<tr>
<td><strong>ifconfig</strong></td>
<td>Displays and modifies the interface table. ifconfig allows the addition of new interfaces, modification of IP addresses and broadcast addresses, and deletion of IP addresses.</td>
</tr>
</tbody>
</table>
### Table 7-1 LAN Communications Pak Utilities (continued)

<table>
<thead>
<tr>
<th>Utility</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ipstart</strong></td>
<td>Initializes IP stack.</td>
</tr>
<tr>
<td><strong>ndbmod</strong></td>
<td>Allows you to add, remove, or modify information stored in the <em>inetdb</em> and <em>inetdb2</em> data modules.</td>
</tr>
<tr>
<td><strong>netstat</strong></td>
<td>Report network information and statistics.</td>
</tr>
<tr>
<td><strong>ping</strong></td>
<td>send ICMP ECHO_REQUEST packets to host.</td>
</tr>
<tr>
<td><strong>route</strong></td>
<td>Add or delete routes.</td>
</tr>
<tr>
<td><strong>telnet</strong></td>
<td>Telnet user interface; telnet provides the ability to log on to remote systems.</td>
</tr>
</tbody>
</table>

---

**Note**

All LAN Communications Pak utilities and servers use the *netdb* shared module and the *inetdb* and *inetdb2* data modules for name resolution.

---

**Note**

Windows 95/NT versions of the *idbgen*, *idbdump*, *rpcdbgen*, and *rpcdump* utilities are provided in addition to the OS-9 versions.
Daemon server programs and connection handlers are identified in the following table.

### Table 7-2 Daemon Server Programs and Connection Handlers

<table>
<thead>
<tr>
<th>Daemon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bootpd</td>
<td>Bootp Server Daemon.</td>
</tr>
<tr>
<td>ftpd</td>
<td>FTP Server Daemon.</td>
</tr>
<tr>
<td>ftpdc</td>
<td>FTP Server Connection Handler (forked by ftpd or inetd).</td>
</tr>
<tr>
<td>inetd</td>
<td>Internet Services Master Daemon. <em>inetd</em> can be configured to fork a particular program to handle data on a particular protocol/port number combination. <em>inetd</em> can replace the <em>ftpd</em> and <em>telnetd</em> server daemons. <em>telnetdc</em> and <em>ftpdc</em> must still be available.</td>
</tr>
<tr>
<td>routed</td>
<td>Dynamic Routing Daemon.</td>
</tr>
<tr>
<td>telnetd</td>
<td>Telnet Server Daemon.</td>
</tr>
<tr>
<td>telnetdc</td>
<td>Telnet Server Connection Handler (forked by telnetd or inetd).</td>
</tr>
<tr>
<td>tftpd</td>
<td>TFTP Server Daemon.</td>
</tr>
<tr>
<td>tftpdc</td>
<td>TFTP Server Connection Handler (forked by tftpd).</td>
</tr>
</tbody>
</table>
Utilities

This section includes utility definitions in alphabetical order according to the following alpha sort rules.

1. Special characters (not letters, numbers, or underscores) are listed first.
2. Utilities are listed in alphabetic order next without regard for numbers and underscores.
3. If two utility names are identical using these rules, then they are alphabetized according to the following order:
   1. Symbols
   2. Underscores
   3. Alphabetic characters
   4. Numbers

Sections

Each utility defined includes a minimum of the following sections:

- Syntax
- Package
- Options
- Description

In addition, definitions may contain sections for the following:

- Commands
- Examples
- See Also
Utilities

Syntax

Each utility description includes a syntactical description of the command line. These symbolic descriptions use the following notations:

\[
\begin{align*}
[ . ] &= \text{Enclosed items are optional} \\
\{ \ } &= \text{Enclosed items may be used 0, 1, or multiple times} \\
< > &= \text{Enclosed item is a description of the parameter to use. For example:}
\end{align*}
\]

- \texttt{<path> = A legal pathlist}
- \texttt{<devname> = A legal device name}
- \texttt{<modname> = A legal memory module name}
- \texttt{<procID> = A process number}
- \texttt{<opts> = One or more options specified in the command description}
- \texttt{<arglist> = A list of parameters}
- \texttt{<text> = A character string ended by end-of-line}
- \texttt{<num> = A decimal number, unless otherwise specified}
- \texttt{<file> = An existing file}
- \texttt{<string> = An alphanumeric string of ASCII characters}

An example of a syntax line follows.

Syntax

\texttt{bootpd [\langle opts\rangle] \{\langle configfile\rangle\}}

Package

\textit{Package} identifies the OEM licensing configuration in which the utility is shipped. Package types are embedded, disk-based, and extended.

An example of a package identification line follows.
Package

- Embedded
- Disk-Based
- Extended

Embedded

The embedded package provides the standard OS-9 Kernel and the unified I/O subsystem manager for ROM-based, embedded applications. The embedded package type is a subset of the disk-based package type.

Disk-Based

The disk-based package adds support for random accessible, block-oriented devices such as floppy disks, SCSI hard disks, RAM disks, optical/WORM drives, etc. The disk-based package type referred to in this manual also supports the tape option package type. The tape option (SBF) supports sequentially accessible, block-oriented devices such as tape drives. The embedded and disk-based package types are subsets of the extended package type.

Extended

The extended package adds industry standard TCP/IP networking, LAN Communications Pak.

Options

The options section lists and defines command line options for the utility.

Description

The description section defines the processing of the utility.
Utilities

Commands

Where applicable, commands for each utility are listed and defined.

Examples

The examples section provides sample uses for the utility.

See Also

This section lists utilities that are related to the current one. You may want to refer to these utilities for additional information.
Utilities

**arp**

**Ethernet/IP Address Resolution Display and Control**

**Syntax**

```
arp [opts]
```

**Package**

- Embedded
- Disk-Based
- Extended

**Options**

- `<hostname>`: Display ARP entry for `<hostname>`.
- `-a`: Display all of the ARP table entries.
- `-d <hostname>`: Delete an entry for the host called `hostname`. This option can only be used by the super-user.
- `-n`: This option can be used when specifying a single hostname, or with the `-a` option. It indicates that IP addresses should not be resolved to hostnames. A "?" will be printed instead of the hostname.
- `-s <hostname> <eth_addr> [temp][pub]`: Create an ARP entry for the host called `<hostname>` with the Ethernet address `<ether_addr>`. The Ethernet address is given as six hex bytes separated by colons. The entry is permanent unless the word `temp` is given in the command. If the word `pub` is given, the entry will be published. For instance, this system responds to ARP requests for `hostname` even though the hostname is not its own. This option can only be used by the super-user.
Description

The `arp` program displays and modifies the Internet-to-Ethernet address translation tables used by the address resolution protocol (ARP). This table is maintained by the `spenet` driver. The age field indicates the number of minutes the entry has been in the table. Non-permanent entries are removed after 20 minutes.

With no flags, the program displays the current ARP entry for `<hostname>`. The host may be specified by name or by number, using Internet dot notation.

Note

To use `arp` you must have at least edition 38 of the `spenet` driver. To check the `spenet` edition on your system, type `ident -m spenet` in your OS-9 command line.

Examples

Publish a temporary arp entry for a machine called odin.

```
arp -s odin 04:00:00:12:34:56 pub temp
```

This entry will expire after 20 minutes. To make it permanent, leave the `temp` qualifier off.

See Also

`ifconfig`
`netstat`
**bootpd**

**BOOTP Request Server Daemon**

**Syntax**

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

**Package**

- Embedded
- Disk-Based
- Extended

**Options**

- `-?` Displays the syntax, options, and command description of `bootpd`.
- `-d` Log debug information to `<stderr>`.
- `-t <num>` Exit after `<num>` minutes of no activity.

**Description**

`bootpd` is the server daemon handling 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 enables changes to `<configfile>` without restarting `bootpd`. By default, `<configfile>` is `/h0/TFTPBOOT/bootptab`.

`bootpd` is normally run in a LAN Communications Pak startup file as follows:

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

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

- `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.

Note

- Super user account is required to run bootpd.
- End the command line with an ampersand (&) to place bootpd in the background (example, bootpd<>>>/nil&).
- bootpd is used in conjunction with tftpd.

For More Information

Refer to Chapter 6: BOOTP Server for how to set up the BOOTP Server.
**bootptest**  
Test Utility for BOOTP Server Response

**Syntax**

```
bootptest -h=<hostname> -e=<etheraddr> -n=<filename> [<opts>]
```

**Package**

- Embedded
- Disk-Based
- Extended

**Options**

- `-h=<hostname>`  
  Target server IP address (name or dotted decimal).

- `-e=<etheradr>`  
  Ethernet address in colon notation.

- `-n=<filename>`  
  Bootfile name for `bootp` server.

- `-f=<filename>`  
  Copy bootfile into `<filename>`.

**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 required and must appear on the command line. `-h` accepts a name which is converted to an IP address using `gethostbyname()`.

If a host name is unavailable, the IP address can be given in dotted decimal notation.

To broadcast, specify 0 or 255 as the host portion of the IP network address. This solicits a response from any BOOTP server on the named network.
The BOOTP client test utility may use all ones (255.255.255.255) for the server IP address when it boots if 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 indicate where the broadcast came. Otherwise, bootptest can use the IP address of the system.

The bootptab configuration file on the bootpd server must specify an entry for the system on which bootptest is running. bootptest cannot perform 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.

Example: The following is an example of bootptest:

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

**dhcpp**

DHCP Client Negotiation Utility

**Syntax**

```
dhcp [<eth_dev>] [<opts>]
```

**Package**

- Embedded
- Disk-Based
- Extended

**Options**

- **-v**  
  Verbose mode. Prints additional information about what the program is doing.

- **eth_dev**  
  Is the name of Ethernet device in inetdb module.

- **-broadcast <address>**  
  Sets the expected DHCP broadcast address.

- **-nofork**  
  Does not fork child DHCP process.

- **-timeout <seconds>**  
  Number of seconds to wait for DHCP reply.

- **-port <port>**  
  UDP port of DHCP server.

**Description**

dhcp is the DHCP client negotiation utility.

The **eth_dev** parameter is the name of the Ethernet interface used with ndbmod or idbgen. This name should not be confused with the ethernet device descriptor or driver. If no device is specified on the command line, and only one interface was added with ndbmod or idbgen, dhcp uses it.

The **broadcast** flag specifies that the DHCP server does not correctly broadcast IP packets to the address 255.255.255.255, but to some other broadcast address (usually `<network>.255`).

The **timeout** flag enables the user to specify how long dhcp waits before retrying a failed request. The default is 10 seconds.
The `-nofork` flag stops the `dhcp` client creating a child process and exiting. This is not normally used.

If the lease time on the IP address supplied by the DHCP server expires, DHCP removes the address, and reverts back to requesting an IP address.

If the DHCP Server supplies DNS information, `dhcp` attempts to add it to an `inetdb` module. Space for this is provided by creating a new `inetdb3` with the `ndbmod` command.

### Examples

Sample output from `dhcp` using the verbose mode.

```
# dhcp -v
DHCP: Ethernet device name 'enet0'
DHCP: Ethernet SFF descriptor name '/spe30'
DHCP: Eth Address is 00:60:97:8C:28:7B
DHCP: Adding IP address: 0.0.0.0
DHCP: Adding broadcast address: 255.255.255.255
DHCP: Adding subnet mask: 0.0.0.0
Sending DHCPDISCOVER to 255.255.255.255
Sending DHCPREQUEST to 255.255.255.255
DHCP: Adding IP address: 192.168.3.200
DHCP: Adding broadcast address: 192.168.3.255
DHCP: Adding subnet mask: 255.255.255.0
DHCP: Received a lease for 3600 seconds
DHCP: adding default route to 192.168.3.225
DHCP: Offered domain name: microware.com
DHCP: Offered DNS Server 172.16.1.32
DHCP: Adding DNS Server 172.16.1.32
```

### See Also

- `ifconfig`
- `ndbmod`
Utilities

**ftp**

File Transfer Manipulation/
Remote Internet Site Communication

**Syntax**

```bash
ftp [opts] [host][opts]
```

**Package**

- Embedded
- Disk-Based
- Extended

**Options**

Options may be specified at the command line or to the command interpreter.

- `-?` Displays the description, options, and command syntax for `ftp`.
- `-d` Turns on debugging mode.
- `-g` Does not expand wildcard file name expansion (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. It then prompts for a password, and optionally, an account with which to login.
- `-r` Turn on overwrite mode to allow copying over existing same-name files with files received when performing `get` operations.
- `-s` Does not set the file size on received data. By default, `ftp` attempts to pre-extend the file when retrieving 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
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. Then, a current status report of the ftp modes displays and ftp waits for instructions. When waiting for commands, ftp displays the prompt ftp. For example:

```
ftp
Not connected.
Mode: stream Type: asciiForm: non-printStructure: file
Verbose: on Bell: offPrompting: onGlobbing: on
Hash mark printing: off Use of PORT commands: off
Overwrite: off Directory Recursion: off
```

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.
Note

ftp does not process remote file names. They are passed just as they are typed, except for the mdelete, mdir, mget, mput, 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 or dir to the server.

File Transfer Protocols

The FTP specifies many parameters which may affect a file transfer. LAN Communications Pak currently supports the following type, mode, and structure parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Can be:</td>
</tr>
<tr>
<td></td>
<td>• ascii specifies network ASCII (used where end of line conversion is required)</td>
</tr>
<tr>
<td></td>
<td>• image or binary specify image</td>
</tr>
<tr>
<td>mode</td>
<td>Must be stream.</td>
</tr>
<tr>
<td>structure</td>
<td>Must be file.</td>
</tr>
</tbody>
</table>

Commands

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

$ [<command>]

Runs as a shell command on the local machine. If <command> is unspecified, it starts an interactive shell.

append [<local>] [<remote>]

Appends <local> to a file on the remote machine. If <remote> is unspecified, the <local> name is used in naming the remote
If neither `<local>` or `<remote>` is specified, `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 [<directory>]**
Changes the current directory on the remote system (same as `chd`). If `<directory>` is not specified, `cd` prompts for one.

**cdup**
Changes remote working directory to parent directory.

**chd [<directory>]**
Changes the current directory on the remote system (same as `cd`). If `<directory>` is not specified, `chd` prompts for one.

**close**
Terminates the remote session and returns to the FTP command interpreter.

**connect [<host> [<port>]]**
Connects to remote FTP. Same as open.

**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 accepted as a parameter, but is not supported.
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 supported.

get [remote] [local] Copies the file remote to the local system. If local is unspecified, 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. If remote is not specified, get prompts for it and local.

glob Toggles globbing (wildcard expansion) of remote file names for mdelete, mget, and mput. If globbing is turned off, file names are taken literally. Globbing is on by default.

hash Toggles printing # (hash marks) for each buffer transferred. Hash is off by default.

help [command] Prints a help message about a command if specified (same as ?). If command is unspecified, help displays a list of available commands.

lcd [directory] Changes the current local directory (same as lchd). If directory is unspecified, ftp changes to local home directory.

lchd [directory] Changes the current local directory (same as lcd). If directory is unspecified, ftp changes to local home directory.
ls [<remote> [<local>]]
Displays an abbreviated directory from the remote system. If <remote> is unspecified, 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. Specify <remote> using file name wildcards (example, ch*.doc).

mdir [<remote> <local>]
Redirects the display of the contents of remote directories to a local file. Specify <remote> using file name wildcards (example, chap*.doc). If <local> and <remote> are unspecified, ftp prompts for the appropriate information.

mget [<remote>]
Copies the remote files to the current local directory. <remote> may be specified using file name wildcards (example, chap*.doc).

mkdir [<remote>]
Makes a directory on the remote system (same as mkdir). If <remote> is unspecified, ftp prompts for the directory name.

mkdir [<remote>]
Makes a directory on the remote system (same as mkdir). If <remote> is unspecified, ftp prompts for the directory name.

mls [<remote> [<local>]]
Redirects the display of remote file/directory listings to the file <local>. <remote> may be specified using file name wildcards (example, chap*.doc). If <remote> and <local> are unspecified, ftp prompts for the appropriate information.

mode [<mode>]
Sets the transfer mode to <mode>. stream is the only accepted <mode>. 
Utilities

mput [<local>] Expands wildcards in the list of <local> and copies each file in the resulting list to the current remote directory. If <local> is not specified, mput prompts for files.

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 <host> is unspecified, ftp prompts for the host name.

overwrite Toggles whether local files are overwritten (if the user has write permissions) on subsequent get/mget commands. Overwrite is off by default.

pd Prints the name of the current remote directory (same as pwd).

prompt Toggles interactive prompting for commands involving multiple file transfers. Interactive prompting occurs during multiple file transfers to enable selective retrieval or storage of files. By default, prompting is turned ON. If prompting is turned off, mget or mput transfers all files, and mdelete deletes all files.

put [<local> [<remote>]] Copies a local file to the remote machine. If <local> is unspecified 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 unspecified, 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 parameters are
unspecified on the command line, ftp prompts
for them.

recurse
Toggles whether directory recursion takes place
during any subsequent mget/mput commands,
(example, transfer all files in subdirectories,
making any needed directories along the way).
Recursion is off by default.

recv [remote] [local]
Copies a remote file to the local system. If
local is unspecified, 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. If
remote is unspecified, recv prompts for
both remote and local.

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 or new are
unspecified 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.
Utilities

Utilities

rmdir [<remote>]

Deletes a directory on the remote machine. If <remote> is unspecified on the command line, ftp prompts for the directory name.

send [<local> [<remote>]]

Copies <local> to the remote machine. If <local> is unspecified, 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 that ignore PORT commands, but incorrectly indicate they were accepted.

Note

This command does not have any effect. It is accepted as a legal command, but not implemented. Sendport 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>. The only valid <struct> is file.

type [<type>]

Sets the representation type to <type>. The valid values for <type> are:
1. ascii for network ASCII
2. binary or image for image
   If <type> not specified, type displays the current setting.

user [<user> [<password>] [<account>]]
Identifies you to the remote FTP server. If <user> is unspecified, ftp prompts for it. If the password and/or the account field is unspecified 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.

? Displays a list of commands available. Reference the help command for more information.
Utilities

**ftpd**

**Incoming FTP Server Daemon**

**Syntax**

```bash
ftpd [<opts>]
```

**Package**

- Embedded
- Disk-Based
- Extended

**Options**

- `?-` Displays the description, options, and command syntax for `ftpd`.
- `-d` Prints debugging information to standard error.
- `-l` Prints user login information to standard error.
- `-u` Allow unauthenticated access.
- `-f "<cmd> <opts>"` File listing command (default: `dir -ua`)
- `-e "<cmd> <opts>"` Extended file listing (default: `dir -ea`)

**Description**

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

To save logging output for later use, redirect the standard error path and standard output path to an appropriate file on the command line:

```bash
ftpd -d </nil >>>-/h0/SYS/ftpd.debug&
```

or

```bash
ftpd -l </nil >>>-/h0/SYS/ftpd.login&
```

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

```bash
ftpd >>>>/nil&
```
Anonymous FTP access is allowed if the password file contains an entry for username ftp with a password of anonymous. An incoming user can specify either ftp or anonymous as a user name along with any password to successfully login. Anonymous ftp users can access all files on the system using the group/user ID specified for the ftp user in the password file.

If the -u option is specified, no authentication is performed and any username and password combination is accepted. This option is useful to download modules to an embedded target because it does not require either the login utility or a password file.

The -e and -f options can be used to override the default commands ftpd runs when responding to directory listing requests. This can be useful for clients that are unable to parse OS-9’s dir command output. For example, a port of the Unix ls command can be run instead of dir to allow web browsers to display directory listings.

```
ftpd -e "ls -l" <>>/nil&
```

The FTPDIRCMD and FTPEDIRCMD environment variables may also be used. The previous ftpd command line is equivalent to the following.

```
setenv FTPEDIRCMD "ls -l"
ftpd <>>/nil&
```

**Note**

- Super user account is required to run ftpd.
- End the command line with an ampersand (&) to place ftpd in the background (example, ftpd<>>/nil&).
- inetd can be used in place of ftpd.
ftpdc **FTP Server Connection Handler**

**Syntax**

```
ftpdc [<opts>]
```

**Package**

- □ Embedded
- □ Disk-Based
- □ Extended

**Options**

- `?-` Displays the description, options, and command syntax for `ftpdc`.
- `-d` Prints debugging information to standard error.
- `-l` Prints user login information to standard error.
- `-u` Allow unauthenticated access.
- `-f "<cmd> <opts>"` File listing command (default: `dir -ua`)
- `-e "<cmd> <opts>"` Extended file listing (default: `dir -ea`)

**Description**

`ftpdc` is the incoming communications handler for `ftp`. `ftpd` and `inetd` can fork `ftpdc`.

**Note**

Do not run this from the command line. Only `ftpd` and `inetd` can invoke this utility.
hostname

Display or Set Internet Name of Host

Syntax

hostname [<name>]

Package

☐ Embedded ☐ Disk-Based ☑ Extended

Options

-? Displays the description, options, and command syntax for hostname.

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 inetdb data module. Use hostname to override the default. The length of the string is limited to the length of the current string in the inetdb data module.
idbdump

Display Internet Database Entries

Syntax

idbdump [<opts>] [<inetdb_file>] [<opts>]

Package

- Embedded
- Disk-Based
- Extended

Options

- ? Displays the description, options, and command syntax for idbdump.
- d[=]<file> Only display <file> entries. Allowable files are host.conf, hosts, hosts.equiv, inetd.conf, networks, protocols, resolv.conf, services, interfaces.conf, hostname, routes.conf, or rpc.
- m Use the inetdb data module in memory (OS-9 resident systems only).

Description

idbdump displays a formatted listing of the entries in the internet database. If options are not specified, idbdump displays all entries in the database. Specific options display the appropriate type of database entries.

By default, idbdump looks for the internet database as the file inetdb in the current data directory. This default can be overridden by a command line path list to the file, or by the –m option.

This command is also available under the Windows 95/NT operating systems.

See Also

idbgen

Using LAN Communications Pak
**idbgen**

Generate Network Database Module(s)

**Syntax**

```
idbgen [opts] [filename] [opts]
```

**Package**

- Embedded
- Disk-Based
- Extended

**Options**

- `-?` Displays the description, options, and command syntax for idbgen.
- `-c` Produces an inetdb compatible with ISP.
- `-d=<path>` Specifies the directory to find the network database files (default is current directory). This option can be repeated to search multiple directories.
- `-r=<num>` Sets the module revision to `<num>`.
- `-to[=]<target>` Specifies the target operating system. (The default is OS9000)

**Table 7-3 Target Operating System <target>**

<table>
<thead>
<tr>
<th>&lt;target&gt;</th>
<th>Operating System</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSK</td>
<td>OS-9 for Motorola 68000 family processors</td>
</tr>
<tr>
<td>OS9000  or OS9K (default)</td>
<td>OS-9</td>
</tr>
</tbody>
</table>

- `-tp=<target>` Specifies the target processor and options. (default is PPC on cross-hosted machines.)
WARNING
When using `-to=<target>` or `-tp=<target>`, `idbgen` does not recognize the target options in lowercase. For example, `osk`, is not recognized; `OSK` is.

<table>
<thead>
<tr>
<th><code>&lt;target&gt;</code></th>
<th>Processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>68k or 68000</td>
<td>Motorola 68000/68010/68070</td>
</tr>
<tr>
<td>CPU32</td>
<td>Motorola 68300 family</td>
</tr>
<tr>
<td>020 or 68020</td>
<td>Motorola 68020/68030/68040</td>
</tr>
<tr>
<td>040 or 68040</td>
<td>Motorola 68040</td>
</tr>
<tr>
<td>386 or 80386</td>
<td>Intel 80386/80486/Pentium</td>
</tr>
<tr>
<td>PPC (default)</td>
<td>Generic PowerPC</td>
</tr>
<tr>
<td>403</td>
<td>PPC 403</td>
</tr>
<tr>
<td>601</td>
<td>MPC 601</td>
</tr>
<tr>
<td>603</td>
<td>MPC 603</td>
</tr>
<tr>
<td>ARM</td>
<td>Generic ARM processor</td>
</tr>
<tr>
<td>ARMV3</td>
<td>ARM V3 processor</td>
</tr>
<tr>
<td>ARMV4</td>
<td>ARM V4 processor</td>
</tr>
</tbody>
</table>
-x
Places the modules in the execution directory (only for OS-9 for 68K/OS-9 resident systems).

-z[=]<file>
Read additional command line arguments from <file>. (The default is standard input.)

Description

`idbgen` generates the OS-9 data modules `inetdb` and `inetdb2` from the network database files: `host.conf`, `hosts`, `hosts.equiv`, `inetd.conf`, `networks`, `protocols`, `resolv.conf`, `interfaces.conf`, `routes.conf`, `services` and `rpc`. Any time a change is made to any of these files, `idbgen` must be used to generate new data modules.

By default, the output internet database modules are named `inetdb` and `inetdb2` and are placed in the current directory. An optional command line file name can override this default by specifying the pathlist to the output files.

By default, `idbgen` looks for the network database files in the current directory. However, the `-d` option can be used to specify the directories containing the files.

For More Information

Refer to Appendix A: Configuring LAN Communications Pak for definitions and descriptions of the configuration files.

This command is also available under the Windows 95/NT operating systems.

See Also

`iddump`
**ifconfig** Configure Network Interface

**Syntax**

```
ifconfig [<interface> [<address> [<dest_address>]] [options]]
```

**Package**

- Embedded
- Disk-Based
- Extended

**Options**

- `<IP Address> <dest_addr|broadcast> [alias]`
  - Change IP address and broadcast address [add an alias].
- `netmask <subnet mask>`
  - Change subnet mask.
- `broadcast <addr>`
  - Change broadcast address.
- `up`
  - Mark interface as up.
- `down`
  - Mark interface as down.
- `delete`
  - Delete IP address or alias.
- `binding <device>`
  - Bind device to interface name.
- `iff_nopointopoint`
  - Override driver `iff_pointopoint` flag.
- `iff_pointopoint`
  - Indicate the interface being added is a point-to-point link. (deprecated)
- `iff_nobroadcast`
  - Override driver `iff_broadcast` flag.
- `iff_broadcast`
  - Indicate the interface being added is capable of sending broadcasts. (deprecated)
- `iff_nomulticast`
  - Override driver `iff_multicast` flag.
Description

`ifconfig` assigns an address to a network interface and/or configures network interface settings. It can also dynamically add an interface to the system.

**address**

The address is either a host name present in the host name data base, or an Internet address expressed in the standard Internet dot notation.

**alias**

Establish an additional network address for this interface. Useful when changing network numbers and accepting packets addressed to the old interface.

**binding <device>**

Dynamically add a new interface to the system. The device parameter specifies the device/protocol descriptors(s) which will be opened and associated with the specified interface name (for example, `/spe30/enet`).

**broadcast <addr>**

Specify the address to use to represent broadcasts to the network. The default broadcast address has a host part of all 1s.

**delete**

Remove the network address specified. Used when an alias is incorrectly specified, or no longer needed.

**dest_address**

Specify the address of the peer on the other end of a point-to-point link.

**down**

Mark an interface down. When an interface is marked down, the system does not attempt to transmit messages through that interface. Incoming packets on the interface are discarded. This action does not automatically disable routes using the interface.
### Utilities

**iff_nobroadcast**

Drivers that are capable of sending broadcasts will automatically set the `iff_broadcast` flag. If you do not want to send broadcasts from a particular ethernet card this option could be used to prevent the flag from being set. This option requires the `binding` option.

**iff_broadcast**

Indicates the interface can send broadcasts. Used in conjunction with the `binding` option when creating a new interface. The broadcast option automatically implies this setting. This option is provided for compatibility, the preferred method is for the driver to set this flag.

**iff_nopointopoint**

Overrides the `iff_pointopoint` flag that is set automatically by the slip and ppp drivers. This option requires the `binding` option.

**iff_pointopoint**

Indicates that the interface is a point-to-point link. Used in conjunction with the `binding` option when creating a new interface. This option is provided for compatibility, the preferred method is for the driver to set this flag.

**iff_nomulticast**

Overrides the `iff_multicast` flag set by multicast capable interfaces. This option is useful if you do not wish to send or receive any multicast packets and are on a network with a high volume of multicast traffic.

**interface**

The interface parameter is a string of the form `nameX` where `X` is the integer unit number (for example, `enet0`). The name is used internally by IP and is not the name of a file or module.
netmask <mask>

Specify how much of the address to reserve for subdividing networks into sub-networks. The mask includes the network part of the local address and the subnet part, which is taken from the host field of the address. The mask can be specified as a single hexadecimal number with a leading 0x, with a dot-notation Internet address, or with a pseudo-network name listed in the inetdb network table "networks". The mask contains 1s for the bit positions in the 32-bit address which are to be used for the network and subnet parts, and 0s for the host part. The mask contains at least the standard network portion, and the subnet field is contiguous with the network portion.

up

Mark an interface up. Used to enable an interface after an ifconfig down. It happens automatically when setting the first address on an interface.

ifconfig displays the current configuration for a network interface when no options are supplied. Only the super-user can modify the configuration of a network interface.

Examples

Create a new interface enet0:

ifconfig enet0 172.16.1.1 binding /spe30/enet

Since no netmask or broadcast address was specified, the appropriate class A, B, or C addresses will be used. In this case, the netmask 255.255.0.0 and a broadcast address of 172.16.255.255 will be used. To override the default netmask:

ifconfig enet0 172.16.1.1 netmask 255.255.255.0 \ binding /spe30/enet

This will also change the broadcast address to 172.16.1.255.
Create a new interface, but disable its multicast capability.

```bash
ifconfig enet0 odin iff_nomulticast binding /spe30/enet
```

This example also uses a machine name instead of an IP address. For this to work, the name must either be resolvable in your local `inetdb` or by a DNS name server reachable by an interface other than the one being added.

Change the address of an existing interface:

```bash
ifconfig enet0 loki
```

If the hardware driver supports it, you can add an alias to an already existing interface:

```bash
ifconfig enet0 thor alias
```

Remove the alias:

```bash
ifconfig enet0 thor delete
```

Create the point-to-point interface `ppp0` without any address information:

```bash
ifconfig ppp0 binding /ipcp0
```

Set the local and destination address of a point-to-point link:

```bash
ifconfig slip0 192.168.8.175 192.168.8.174
```

Show the address information for interface `enet0`:

```bash
ifconfig enet0
```

The output is similar to the following:

```
enet0: flags=8003<UP,BROADCAST,MULTICAST> mtu 1500
      inet 172.16.1.1 netmask 0xffff0000 broadcast 172.16.255.255
```

or for a point-to-point interface:

```
ppp0: flags=8011<UP,POINTOPOINT,MULTICAST> mtu 1500
      inet 192.168.8.175 --> 192.168.8.174 netmask 0xffffff00
```

**See also**

`netstat`
### inetd

#### Internet Master Daemon

**Syntax**

```
inetd [<opts>]
```

**Package**

- Embedded
- Disk-Based
- Extended

**Options**

```
-?

 Displays the description, options, and command syntax for inetd.

-i[...]

 Internal inetd options.
```

**Description**

*inetd* is a master internet daemon process, that can be configured to listen for incoming TCP or UDP connections on up to 25 separate ports. When *inetd* detects an incoming connection, it forks a child process (for example *telnetdc* or *ftpdc*) to handle the connection.

The *inetd.conf* file specifies the configuration for the *inetd* process. Each non-comment or non-white space line in this file specifies a service which *inetd* provides. Each line has the following syntax:

```
<ServiceName> <SocketType> <Protocol> <Flags> <User>
<ServerPathname> [<Args>]
```

- **ServiceName**
  Specifies the internet service to be provided. This service name must also be specified in the `<services>` section of the *inetdb* data module. The port number for the specified service is referenced through the `<services>` section of *inetdb*.

- **SocketType**
  Specifies whether the socket type is a stream (for TCP services) or *dgram* (for UDP services).
| **Protocol** | Specifies the protocol type to use for this service. This protocol name must also be specified in the `<protocols>` section of the `inetdb` data module. |
| **Flags** | Specifies special actions `inetd` should take when processing incoming connections for this service. `Wait` is the only valid `Flags` action. |
| **User** | Specifies the group/user under which the forked child process should run—either in the format `<group>..<user>` (such as “12.136”), the keyword `root` (resolving to user “0.0”), or the keyword `nobody` (resolving to user “1.0”). |
| **ServerPathname** | Specifies the child process to fork when an incoming connection for the specified service is detected. This may be just the program name if it is in memory or in the current path, or it may be a full path name to the program file. Eight special services, `echo` (dgram & stream), `discard` (dgram & stream), `daytime` (dgram & stream), and `chargen` (dgram & stream) are handled internally by `inetd` by specifying the server path name “internal”. |
| **Args** | Specifies additional arguments to use when forking the child process. |

When `inetd` forks a child process to handle an incoming service request, the child is forked with three paths: `stdin` and `stdout` are the connected socket paths and `stderr` is the `stderr` path with which `inetd` was forked. If the service was on a `udp/dgram` socket, a `connect()` is performed before forking the child. If the service was on a `tcp/stream` socket, an `accept()` is performed before forking the child.

---

**Note**

Super user account is required to run `inetd`.
ipstart

Syntax

ipstart

Package

☐ Embedded  ☑ Disk-Based  ☑ Extended

Options

None.

Description

ipstart initializes the IP stack at startup. Run this utility in the foreground.
ndbmod

**Dynamically Update the Internet Data Module**

**Utilities**

**Syntax**

ndbmod <[opts]>

**Package**

- Embedded
- Disk-Based
- Extended

**Options**

- ? Displays the description, options, and command syntax for ndbmod.

hostname <hostname> Set hostname of the system.

interface <options> Add or delete an interface.

Options are defined as follows:

```plaintext
add <intname> [address <addr> [netmask <addr>]] binding <device> [mtu <mtu>] [metric <metric>] [up|down] [iff_broadcast][iff_pointopoint] [iff_nomulticast] [iff_nobroadcast] [iff_nopointopoint]
del <intname>
```

The [iff_broadcast] and [iff_broadcast] flags are for compatibility with older version of the LAN Communications Pak. The preferred method is to let the driver set these flags and to use the [iff_noXXX] flags to override the driver defaults.

resolve <arglist> Add DNS search and resolver information.

Options are defined as follows:

```plaintext
<domainname [server <addr>] [search <srch-1> <srch-2> ...]
```
host <options>  Add or delete a host entry.
Options are defined as follows:

   add <addr> <name> [alias-1 <alias-2> ...]
   del <name|alias>

route <options>  Add static route.
Options are defined as follows:

   add [host|net] <dst-route> <gateway> [netmask]
   add [default]<gateway>[netmask]

create <modname> <num-files> <size-1> <size-2>...
    <size-n>
Create a dynamic Internet data module having <num-files> files.

   <size-1> indicates bytes to reserve for file 1
   <size-2> indicates bytes to reserve for file 2
All file sizes must be specified, with 0 indicating no space is reserved for that file.

   <modname> is one of 'inetdb2', 'inetdb3', or 'inetdb4'

File Numbers

1  /etc/hosts (aprox. 25 bytes per host)
2  /etc/hosts.equiv (not used)
3  /etc/networks (aprox. 40 bytes per network)
4  /etc/protocols (aprox. 25 bytes per protocol)
5  /etc/services (aprox. 25 bytes per service)
Utilities

6 /etc/inetd.conf (aprox. 50 bytes per entry)
7 /etc/resolv.conf (aprox. 100 bytes)
8 host config (not used)
9 host interfaces (aprox 200 bytes per interface)
10 hostname (>= length of hostname + 1, recommended 65)
11 static routes (aprox. 64 bytes per entry)
12-32 reserved

Description

The ndbmod utility allows the user to add, remove, or modify information stored in the inetdb data module such as host names, IP addresses, and DNS Server fields dynamically. It also enables the creation of expansion inetdb modules in the event that the inetdb data module is full or in ROM.

These expansion modules can also hold additional host specific information such as the machines hostname, and interface settings such as IP address and network masks.

Note

ndbmod must be run before the IP stack is initialized.
Examples

Example 1:
To create an alternate inetdb module called inetdb2 with 100 bytes for new host information, 0 for hosts.equiv, 30 for new network entries:
ndbmod create inetdb2 100 0 30 30 0 50 200 100 0 150 64

Example 2:
To set the hostname for a system name alpha:
ndbmod hostname alpha

Example 3:
To add a host entry for system beta with alias of beta1 and beta2:
ndbmod host add 192.1.1.3 beta beta1 beta2

Example 4:
To remove the host entry for beta:
ndbmod host del beta

Example 5:
To add a SLIP interface to an inetdb2 data module:
ndbmod interface add slip0 address 192.1.1.1 \destaddr 192.1.1.2 binding /sps10
Example 6:
To delete a SLIP interface from the inetdb2 data module:
ndbmod interface del slip0

Example 8:
To add an ethernet interface and disable the multicast capabilities:
ndbmod interface add enet0 address 172.16.1.1 \
iff_nomulticast binding /spe30/enet

Example 7:
To add a new DNS resolve entry to the inetdb2 data module:
ndbmod resolve testdns.com server 192.1.1.3 search
testdns.com misc.org

Example 8:
To add a static network route to the inetdb2 data module:
ndbmod route add network 192.2.2.0 192.1.1.2

See Also
idbgen
netstat

Report Network Information

Syntax

netstat [<opts>]

Package

- Embedded
- Disk-Based
- Extended

Options

-A
With the default display, show the address of any protocol control blocks associated with sockets; used for debugging.

-a
With the default display, show the state of all sockets; normally sockets used by server processes are not shown. With the interface display show all multicast groups each interface has joined.

-d
With either interface display (option -i or an interval), show the number of dropped packets.

-f address_family
Limit statistics or address control block reports to those of the specified address family. Only inet (for AF_INET) is currently supported.

-I interface
Show information about the specified interface; used with a wait interval.

-i
Show information for all interfaces.

-n
Show network addresses as numbers (normally netstat interprets addresses and attempts to display them symbolically). This option may be used with any of the display formats.
Utilities

-p protocol  Show statistics about protocol, which is either a well-known name for a protocol or an alias for it. A null response typically means that there are no interesting numbers to report. The program complains if the protocol is unknown or if there is no statistics routine for it.

-s  Show per-protocol statistics. If this option is repeated, counters with a value of zero are suppressed.

-r  Show the routing tables. When -s is also present, show routing statistics instead.

-w wait  Show network interface statistics at intervals of wait seconds.

Description

The netstat command symbolically displays the contents of various network-related data structures.

The default display, for active sockets, shows the local and remote addresses, send and receive queue sizes (in bytes), protocol, and the internal state of the protocol. Address formats are of the form “host.port” or “network.port” if a socket's address specifies a network but no specific host address. When known, the host and network addresses are displayed symbolically according to the data module inetdb hosts and networks sections. If a symbolic name for an address is unknown, or if the -n option is specified, the address is printed numerically. Unspecified, or “wildcard”, addresses and ports appear as “*”.

The interface display provides a table of cumulative statistics regarding packets transferred, errors, and collisions. The network addresses of the interface and the maximum transmission unit (“mtu”) are also displayed.
The routing table display indicates the available routes and their status. Each route consists of a destination host or network and a gateway to use in forwarding packets. The flags field shows a collection of information about the route stored as binary choices.

The mapping between letters and flags is shown in Table 7-5 Mapping Between Letters and Flags.

Table 7-5  Mapping Between Letters and Flags

<table>
<thead>
<tr>
<th>Letter</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RTFPROTO2</td>
<td>Protocol specific routing flag #1</td>
</tr>
<tr>
<td>2</td>
<td>RTFPROTO1</td>
<td>Protocol specific routing flag #2</td>
</tr>
<tr>
<td>B</td>
<td>RTFBLACKHOLE</td>
<td>Just discard pkts (during updates)</td>
</tr>
<tr>
<td>C</td>
<td>RTFCLONING</td>
<td>Generate new routes on use</td>
</tr>
<tr>
<td>D</td>
<td>RTFDYNAMIC</td>
<td>Created dynamically (by redirect)</td>
</tr>
<tr>
<td>G</td>
<td>RTFGATEWAY</td>
<td>Destination requires forwarding by intermediary</td>
</tr>
<tr>
<td>H</td>
<td>RTFHOST</td>
<td>Host entry (net otherwise)</td>
</tr>
<tr>
<td>L</td>
<td>RTFLLINFO</td>
<td>Valid protocol to link address translation.</td>
</tr>
<tr>
<td>M</td>
<td>RTFMODIFIED</td>
<td>Modified dynamically (by redirect)</td>
</tr>
<tr>
<td>R</td>
<td>RTFREJECT</td>
<td>Host or net unreachable</td>
</tr>
<tr>
<td>S</td>
<td>RTFSTATIC</td>
<td>Manually added</td>
</tr>
<tr>
<td>U</td>
<td>RTFUP</td>
<td>Route usable</td>
</tr>
<tr>
<td>X</td>
<td>RTFXRESOLVE</td>
<td>External daemon translates proto-to-link address</td>
</tr>
</tbody>
</table>
Routes are automatically created for each directly connected subnet. The gateway field for such entries shows the address of the outgoing interface. In addition a route to the loopback address (127.0.0.1) is also added for each interface. The `Refs` field gives the current number of active uses of the route. Connection oriented protocols normally hold on to a single route for the duration of a connection while connectionless protocols obtain a route while sending to the same destination. The `use` field provides a count of the number of packets sent using that route. The interface entry indicates the network interface used for the route.

When `netstat` is invoked with the `-w` option and a wait interval argument, it displays a running count of statistics related to network interfaces. This display consists of a column for the loopback interface and a column summarizing information for all interfaces. The loopback interface may be replaced with another interface using the `-I` option. The first line of each screen of information contains a summary since the system was last rebooted. Subsequent lines of output show values accumulated over the preceding interval.

### Examples

Each of the examples consists of the `netstat` command line followed by sample results.

**Display all open sockets**

```
netstat -a
```

Active Internet connections (including servers)

<table>
<thead>
<tr>
<th>Proto</th>
<th>Recv-Q</th>
<th>Send-Q</th>
<th>Local Address</th>
<th>Foreign Address</th>
<th>(state)</th>
</tr>
</thead>
<tbody>
<tr>
<td>tcp</td>
<td>0</td>
<td>0</td>
<td>heimdall.1024</td>
<td>thor.ftp</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>tcp</td>
<td>0</td>
<td>2</td>
<td>heimdall.telnet</td>
<td>odin.1032</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>tcp</td>
<td>0</td>
<td>0</td>
<td><em>.</em>.sunrpc</td>
<td><em>.</em></td>
<td>LISTEN</td>
</tr>
<tr>
<td>tcp</td>
<td>0</td>
<td>0</td>
<td><em>.</em>.chargen</td>
<td><em>.</em></td>
<td>LISTEN</td>
</tr>
<tr>
<td>tcp</td>
<td>0</td>
<td>0</td>
<td><em>.</em>.daytime</td>
<td><em>.</em></td>
<td>LISTEN</td>
</tr>
<tr>
<td>tcp</td>
<td>0</td>
<td>0</td>
<td><em>.</em>.discard</td>
<td><em>.</em></td>
<td>LISTEN</td>
</tr>
<tr>
<td>tcp</td>
<td>0</td>
<td>0</td>
<td><em>.</em>.echo</td>
<td><em>.</em></td>
<td>LISTEN</td>
</tr>
<tr>
<td>tcp</td>
<td>0</td>
<td>0</td>
<td><em>.</em>.telnet</td>
<td><em>.</em></td>
<td>LISTEN</td>
</tr>
<tr>
<td>tcp</td>
<td>0</td>
<td>0</td>
<td><em>.</em>.ftp</td>
<td><em>.</em></td>
<td>LISTEN</td>
</tr>
<tr>
<td>udp</td>
<td>0</td>
<td>0</td>
<td><em>.</em>.route</td>
<td><em>.</em></td>
<td>LISTEN</td>
</tr>
<tr>
<td>udp</td>
<td>0</td>
<td>0</td>
<td><em>.</em>.chargen</td>
<td><em>.</em></td>
<td>LISTEN</td>
</tr>
<tr>
<td>udp</td>
<td>0</td>
<td>0</td>
<td><em>.</em>.daytime</td>
<td><em>.</em></td>
<td>LISTEN</td>
</tr>
<tr>
<td>udp</td>
<td>0</td>
<td>0</td>
<td><em>.</em>.discard</td>
<td><em>.</em></td>
<td>LISTEN</td>
</tr>
<tr>
<td>udp</td>
<td>0</td>
<td>0</td>
<td><em>.</em>.echo</td>
<td><em>.</em></td>
<td>LISTEN</td>
</tr>
</tbody>
</table>
Display the interface information for `enet0`:

```
netstat -I enet0
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Mtu</th>
<th>Network</th>
<th>Address</th>
<th>Ipkts</th>
<th>Ierrs</th>
<th>Opkts</th>
<th>Oerrs</th>
<th>Coll</th>
</tr>
</thead>
<tbody>
<tr>
<td>enet0</td>
<td>1500</td>
<td>&lt;Link&gt;</td>
<td>08.00.3E.30.12.5B</td>
<td>8506469</td>
<td>0</td>
<td>833830</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>enet0</td>
<td>1500</td>
<td>corp-net</td>
<td>heimdall</td>
<td>8506469</td>
<td>0</td>
<td>833830</td>
<td>8</td>
<td>0</td>
</tr>
</tbody>
</table>

Show the interface information for all interfaces, including multicast groups. Display as numbers rather than names.

```
netstat -ian
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Mtu</th>
<th>Network</th>
<th>Address</th>
<th>Ipkts</th>
<th>Ierrs</th>
<th>Opkts</th>
<th>Oerrs</th>
<th>Coll</th>
</tr>
</thead>
<tbody>
<tr>
<td>lo0</td>
<td>1536</td>
<td>&lt;Link&gt;</td>
<td></td>
<td>120</td>
<td>0</td>
<td>120</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>lo0</td>
<td>1536</td>
<td>127</td>
<td>127.0.0.1</td>
<td>120</td>
<td>0</td>
<td>120</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>enet0</td>
<td>1500</td>
<td>&lt;Link&gt;</td>
<td>08.00.3E.30.12.5B</td>
<td>8506188</td>
<td>0</td>
<td>833825</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>enet0</td>
<td>1500</td>
<td>172.16</td>
<td>172.16.2.56</td>
<td>8506188</td>
<td>0</td>
<td>833825</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>enet1</td>
<td>1500</td>
<td>&lt;Link&gt;</td>
<td>00.60.97.60.FE.07</td>
<td>767197</td>
<td>0</td>
<td>847350</td>
<td>2</td>
<td>53</td>
</tr>
<tr>
<td>enet1</td>
<td>1500</td>
<td>192.168.3</td>
<td>192.168.3.225</td>
<td>767197</td>
<td>0</td>
<td>847350</td>
<td>2</td>
<td>53</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>224.0.0.9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>224.0.0.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ppp0*</td>
<td>65535</td>
<td>&lt;Link&gt;</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Display UDP protocol statistics:

```
netstat -p udp
```

```
udp:
    1675124 datagrams received
    0 with incomplete header
    0 with bad data length field
    0 with bad checksum
    20 dropped due to no socket
    1397615 broadcast/multicast datagrams dropped due to no socket
    0 dropped due to full socket buffers
    277489 delivered
    198780 datagrams output
```

Display the TCP protocol statistics, supressing all zero values:

```
netstat -p tcp -ss
```

```
tcp:
    722 packets sent
    702 data packets (41861 bytes)
    16 ack-only packets (9 delayed)
    4 control packets
    1092 packets received
    697 acks (for 41865 bytes)
    4 duplicate acks
    621 packets (822 bytes) received in-sequence
    1 completely duplicate packet (0 bytes)
    3 out-of-order packets (0 bytes)
    1 connection request
    3 connection accepts
```
Utilities

4 connections established (including accepts)
4 connections closed (including 0 drops)
698 segments updated rtt (of 699 attempts)
415 correct ACK header predictions
384 correct data packet header predictions
14 PCB cache misses

Display the current routing table:

```
netstat -r
Routing tables

Internet:  Destination      Gateway            Flags     Refs     Use  Interface
          default          inet-router        UG          1       46  enet0
          localhost        localhost          UH          0        0  lo0
          corp-net         heimdall           U           9   831463  enet0
          heimdall         localhost          UHS         3      120  lo0
          test-net         heimdall2          U           2   846425  enet1
          heimdall2        localhost          UHS          0       0  lo0
          odin             thor               UGH         0       0  enet0
```
**ping**

Send ICMP ECHO_REQUEST Packets to Host

---

**Syntax**

```
ping [opts] host
```

**Package**

- Embedded
- Disk-Based
- Extended

**Options**

- `-s <packetsize>`: Number of data bytes to send.
- `?-`: Print help.

**Description**

The `ping` command sends an ICMP echo request to a specified host and waits for a reply. With the `-s` option, you can specify the size of data to send to the host. Upon success, `ping` displays the number of bytes received from the host and transmission time.

**Note**

You can ping broadcast and multicast addresses; however, only the first machine to respond will be printed.
route

Syntax

route <opts>

Package

☐ Embedded ☐ Disk-Based ☑ Extended

Options

-n Show addresses as numbers rather than names.

add [-net|-host] <dest> <gateway> [netmask] [-hopcount <num>]

Add a network or host route to <dest> going through <gateway> to the routing table.

delete [-net|-host] <dest> [gateway]

Delete the route to <dest> from the routing table.

print

Print routing table.

Description

route updates and prints the current routing table once the IP stack has been initialized.

If the optional <netmask> is supplied when adding a route, -net is automatically assumed. If no <netmask> is supplied and the route is a network route, the standard class A, B, or C netmask for <dest> is used.

If routed is running, the -hopcount parameter causes the new route to be included in RIP responses with a metric of <num>.
Examples

Add a route to the 172.16.64.0 network using heimdall2 as a gateway. The optional `<netmask>` must be used to override the default class B subnet mask.

```
route add -net 172.16.64.0 heimdall2 255.255.255.0
```

To delete the route use either:

```
route delete -net 172.16.64.0 heimdall2 255.255.255.0
```

or, if the network address unambiguously identifies the route, you can use the shorter:

```
route delete 172.16.64.0
```

Add a network route that will be included in routing updates with a metric of 3:

```
route add -net 172.16.0.0 heimdall2 -hopcount 3
```

See Also

```
netstat
routed
```
Utilities

**routed**

Dynamic Routing Daemon

**Syntax**

```
routed [<opts>]
```

**Package**

- Embedded
- Disk-Based
- Extended

**Options**

- `?`
  - Print help.
- `-s`
  - Force `routed` to send routing updates. This is the default if more than one network interface is configured and IP forwarding has been enabled in `spip`.
- `-q`
  - Do not send any routing updates.
- `-h`
  - Do not advertise host routes if a network route to the host also exists.
- `-m`
  - Advertise a host route for the machine’s primary address. The primary address is found by resolving the machine name returned by `gethostname()`.
- `-t`
  - Increase the debugging level causing more trace information to be written to `stdout`. This option can be specified up to four times to select the highest debug level. The debug level can also be increased or decreased by sending a `SIGUSR1` or `SIGUSR2` signal to the `routed` process.
-A
Ignore authenticated RIPv2 packets if authentication is disabled. This option is required for RFC 1723 conformance although it can cause valid routes to be ignored.

-T <file>
Output trace information to <file> instead of stderr.

-P <parm>{[,<parm>)}
Equivalent to adding <parm>{[,<parm>]} to a single line in the /h0/SYS/gateways file.

-F net[/mask],metric
Only send a "fake" default route to this network. This option is used to minimize RIP traffic but can cause routing loops if the route is propagated.

Description

Note
routed requires that routing domain support (sproute and route0) be loaded in the system.

Routed is a network routing daemon used to maintain routing tables. It supports the Routing Information Protocol (RIP) versions 1 and 2, and the Internet Router Discovery Protocol as defined in RFC's 1058, 1723, and 1256.

The daemon opens a UDP socket and listens for any routing packets sent to the route port (normally 520) and updates the routing table maintained by spip. If the system is configured as a router, copies of the routing table are periodically sent to all directly connected hosts and networks.
When `routed` is started, all non-static routes are removed from the routing table. Any static routes present (such as those added by the `route` utility) are preserved and included in RIP responses if they have a valid RIP metric. Also, if the file `/h0/SYS/gateways` exists, it is read for additional configuration options.

The following options are supported with the `-P` command line option or from `/h0/SYS/gateways`.

- `if=<ifname>`
  All other options on this line apply to the interface `<ifname>`.
- `passive`
  Do not advertise this interface, and do not use this interface for RIP and router discovery.
- `no_rip`
  Disable RIP processing on the specified interface. If Router Discovery Advertisements are not enabled with the `-s` option, `routed` acts as a client router discovery daemon.
- `no_ripv1_in`
  Ignore all RIP version 1 responses.
- `no_ripv2_in`
  Ignore all RIP version 2 responses.
- `no_rdisc`
  Disables the Internet Router Discovery Protocol.
- `no_solicit`
  Disable the transmission of router discovery solicitations.
- `no_rdisc_adv`
  Disable transmission of router discovery advertisements.
- `rdisc_pref=<num>`
  Set the preference in router discovery advertisements to `<num>`.
- `rdisc_interval=<num>`
  Set the interval that router discovery advertisements are sent to `<num>` seconds and their lifetime to `3 * `<num>`.`

See Also

`route`
**telnet**

**Provide Internet Communication Interface**

**Syntax**

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

**Package**

- **Embedded**
- **Disk-Based**
- **Extended**

**Options**

- `-?` Displays the description, option, and command syntax for `telnet`.
- `-d` Turns on socket level debugging.
- `-e=<ctrl-character> or “-e=^<character>”` Set escape to user defined character.
- `-n` No escape available.
- `-o` Shows options processing.

**Description**

`telnet` communicates with another host using the TELNET protocol. Executing `telnet` without parameters defaults to 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 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 can be redefined either from the environmental variable `TELNETESCC` or the command line option `-e`. In command mode, normal terminal editing conventions are available.
Using LAN Communications Pak

Utilities

Commands

The following commands are available. Type enough of each command to uniquely identify it.

capture [<param>] Captures I/O of a telnet session to a specified file. capture supports four parameters as identified in the following table.

close Closes the current connection and returns to telnet command mode.

display Displays the current telnet operating parameters. Refer to toggle.

help [<command>] Prints help information for specified ?[<command>] command. If no command is specified, help lists them all.

Table 7-6 Capture Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;file&gt;</td>
<td>Specifies a new file in which to write the I/O of a telnet session. If &lt;file&gt; already exists, capture returns an error. When &lt;file&gt; is specified, capture creates and opens the file and turns on capture mode.</td>
</tr>
<tr>
<td>on</td>
<td>Turns on capture mode; begins to write I/O to the current specified capture file.</td>
</tr>
<tr>
<td>off</td>
<td>Turns off capture mode; stops writing I/O to the specified capture file. <strong>NOTE:</strong> This does not close the file.</td>
</tr>
<tr>
<td>close</td>
<td>Closes the capture file.</td>
</tr>
</tbody>
</table>
mode <param>  Tries to enter line-by-line or character-at-a-time mode.

Table 7-7  Mode Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>character</td>
<td>character at a time.</td>
</tr>
<tr>
<td>line</td>
<td>line-by-line.</td>
</tr>
</tbody>
</table>

open [<host>][<port>]

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 unspecified, 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 as identified in the following table.

Table 7-8  Send Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ao</td>
<td>Abort Output</td>
</tr>
<tr>
<td>ayt</td>
<td>‘Are You There’</td>
</tr>
<tr>
<td>brk</td>
<td>Break</td>
</tr>
<tr>
<td>ec</td>
<td>Erase Character</td>
</tr>
<tr>
<td>el</td>
<td>Erase Line</td>
</tr>
</tbody>
</table>
**Table 7-8 Send Parameters (continued)**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>escape</td>
<td>Current Escape Character</td>
</tr>
<tr>
<td>ga</td>
<td>‘Go Ahead’ Sequence</td>
</tr>
<tr>
<td>ip</td>
<td>Interrupt Process</td>
</tr>
<tr>
<td>nop</td>
<td>‘No Operation’</td>
</tr>
<tr>
<td>synch</td>
<td>‘Synch Operation’ Command</td>
</tr>
<tr>
<td>?</td>
<td>Display send parameters.</td>
</tr>
</tbody>
</table>

**Table 7-9 Set Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>echo &lt;local char&gt;</td>
<td>Sets character to toggle local echoing on and off.</td>
</tr>
<tr>
<td>erase &lt;local char&gt;</td>
<td>Sets the telnet erase character.</td>
</tr>
<tr>
<td>flushoutput &lt;local char&gt;</td>
<td>Abort output.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>interrupt &lt;local char&gt;</td>
<td>Sets the telnet interrupt character. This character sends an Interrupt Process.</td>
</tr>
<tr>
<td>kill &lt;local char&gt;</td>
<td>Sets the telnet kill character. This character sends an Erase Line.</td>
</tr>
<tr>
<td>quit &lt;local char&gt;</td>
<td>Sets the telnet quit character. This character sends a Break.</td>
</tr>
<tr>
<td>?</td>
<td>Display help information.</td>
</tr>
<tr>
<td>status</td>
<td>Shows the current telnet status. This includes the connected peer and the state of debugging.</td>
</tr>
<tr>
<td>toggle &lt;param&gt;</td>
<td>Toggles telnet operating parameters, listed in the following table.</td>
</tr>
</tbody>
</table>
### Table 7-10  Toggle Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>crmod</td>
<td>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 characters typed, only those received. This mode is sometimes required for some hosts asking the user to perform local echoing.</td>
</tr>
<tr>
<td>localchars</td>
<td>Toggles the effects of the set using the set command.</td>
</tr>
<tr>
<td>debug</td>
<td>Toggles debugging mode. When debug is on, it opens connections with socket level debugging on. Turning debug on does not affect existing connections.</td>
</tr>
<tr>
<td>netdata</td>
<td>Toggles the printing of hexadecimal network data in debugging mode.</td>
</tr>
<tr>
<td>options</td>
<td>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.</td>
</tr>
<tr>
<td>?</td>
<td>Display help information.</td>
</tr>
<tr>
<td>z</td>
<td>Suspends the current telnet session and forks a shell.</td>
</tr>
<tr>
<td>$</td>
<td>Suspends the current telnet session and forks a shell.</td>
</tr>
</tbody>
</table>
**telnetd**

*Incoming Telnet Server Daemon*

**Syntax**

```
telnetd [<opts>]
```

**Package**

- Embedded
- Disk-Based
- Extended

**Options**

- `-?`
  - Displays the description, options, and command syntax for `telnetd`.
- `-d`
  - Prints the debug information to standard error.
- `-f=<program>`
  - Program to fork (default = “login”).
- `-l`
  - Prints login information to standard error.
- `-t`
  - Idle connection timeout value (in minutes).
  - Default = 0 (no connection timeout).

**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.

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

```
telnetd -d </nil >>>-/h0/SYS/telnetd.debug&
```

or

```
telnetd -l </nil >>>-/h0/SYS/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&
```
**Note**

- Super user account is required to run `telnetd`.

- End the command line with an ampersand (`&`) to place `telnetd` in the background (example, `telnetd<>>>/nil&`).

**Note**

The `-f` option is useful for remote access to a diskless embedded system. The need for a RAM disk with a password file is eliminated with the following command line:

```
telnetd -f=shell <>>>/nil/b
```

(`mshell can be used in place of `shell`)

A telnet to this machine goes straight to a shell prompt WITHOUT a login prompt.
**telnetdc**  
**Telnet Server Connection Handler**

**Syntax**

telnetdc [<opts>]

**Package**

- Embedded
- Disk-Based
- Extended

**Options**

- `-?` Displays the description, options, and command syntax for `telnetdc`.
- `-d` Prints the debug information to standard error.
- `-l` Prints the login information to standard error.
- `-t` Idle connection timeout value (in minutes). Default = 0 (no connection timeout).
- `-f = <program>` Program to fork (default = ‘login’)

**Description**

telnetdc is the incoming connection handler for telnet. telnetdc can only be forked from telnetd or inetd.

**Note**

Do not run this from the command line. Only telnetd and inetd can fork this utility.

The pseudo keyboard modules pkman, pkdrv, and pk are required. OS-9 for 68K also requires pks.
**tftpd**  
**Respond to tftpd Boot Requests**

**Syntax**

```bash
tftpd [<opts>] [<dirname>] [<opts>]
```

**Package**

- Embedded
- Disk-Based
- Extended

**Options**

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

**Description**

`tftpd` is the server daemon handling the client Trivial File Transfer Protocol (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 a way to validate or restrict a transfer request since login procedures do not exist. 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`.

**For More Information**

Refer to Chapter 6: BOOTP Server for more information about the BOOTP Server.
**tftpdc**

**TFTP Server Connection Handler**

**Syntax**

`tftpdc [opts]`

**Package**

- Embedded
- Disk-Based
- Extended

**Options**

- `?`
  
  Display usage.

**Description**

`tftpdc` is the incoming communications handler for TFTP. Each time a TFTP service connection is made, `tftpd` forks this process.

`tftpdc` is intended to be run only by `tftpd` and, therefore, has no command line options.
Chapter 8: Programming

This chapter covers the following topics:

• Establishing a socket
• Reading/writing data using sockets
• Setting up non-blocking sockets
• Broadcasting
• Multicasting
• Controlling socket operation
LAN Communications Pak is based on sockets. Sockets have the following characteristics:

- Sockets are abstractions providing application programs with access to the communication protocols.
- Sockets serve as endpoints of a communication path between processes running on the same or different hosts.
- Sockets enable one system to send and receive information from other systems on the network.
- Sockets also enable programmers to use complicated protocols, such as TCP/IP, with little effort.

The following illustrates how a socket might look in a network:

**Figure 8-1  Network**

For More Information
See Appendix A: Example Programs and Test Utilities of the *LAN Communications Pak Programming Reference*. It contains example programs for various types of sockets. You can use these programs as templates for writing your own programs.
Socket Types

Each socket has a specific address family, a specific protocol, and an associated type. Before establishing a socket, familiarize yourself with the types of sockets available and decide which is best for your needs.

The following address families are supported:

Table 8-1 Address Families

<table>
<thead>
<tr>
<th>Address Family</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AF_INET</td>
<td>ARPA internet address family.</td>
</tr>
<tr>
<td>AF_ROUTE</td>
<td>BSD 4.4 style routing domain.</td>
</tr>
</tbody>
</table>

Two high-level protocols, the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP), are supported.

For More Information

These protocols are discussed in Chapter 1: Networking Basics.
The following types of sockets are supported. Both `SOCK_STREAM` and `SOCK_DGRAM` imply a specific protocol to use. When using raw sockets the protocol is indicated when opening the socket.

### Table 8-2 Socket Type/Protocol

<table>
<thead>
<tr>
<th>Socket Type</th>
<th>Description</th>
<th>Implied Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SOCK_STREAM</code></td>
<td>Stream sockets</td>
<td>TCP</td>
</tr>
<tr>
<td><code>SOCK_DGRAM</code></td>
<td>Datagram sockets</td>
<td>UDP</td>
</tr>
<tr>
<td><code>SOCK_RAW</code></td>
<td>RAW Sockets</td>
<td>none</td>
</tr>
</tbody>
</table>

**Stream Sockets**

Stream sockets are full-duplex byte streams, similar to pipes. Stream sockets must be in a connected state, and only two sockets can be connected at a time. A socket in the connected state has been bound to a permanent destination. Each socket in the connected pair is a peer of the other.

Stream sockets use the TCP protocol. TCP ensures data is not lost or duplicated. If data for which the peer protocol has buffer space cannot be successfully transmitted within a reasonable length of time, the connection is considered broken.

**Datagram Sockets**

Datagram sockets provide bi-directional flow of data packets called *messages*. A datagram socket state can be either connected or unconnected.

Datagram sockets use the UDP protocol. UDP does not guarantee sequenced, reliable, or unduplicated message delivery. However, UDP has a reduced protocol overhead and is adequate in many cases.
Raw Sockets

Raw sockets provide an unreliable datagram service. They enable an application to create its own protocol headers and are used to send and receive ICMP messages as well as implementing protocol types not supported directly by IP. The use of raw sockets is restricted to super user processes.
Establishing a Socket

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

The following describes how to establish sockets and the necessary LAN Communications Pak system calls.

Stream Sockets

Stream sockets use TCP for a transport protocol and must be connected before sending or receiving data. The sequence for connecting a socket is slightly different between the server and client sides.

Server Steps

Follow this sequence on the server side:

Step 1. Create the Socket

Sockets are created with the `socket()` function. `socket()` requires three parameters:

- Address format (`AF_INET`)
- Type (`SOCK_STREAM`)
- Protocol (0)

`socket()` returns a path number to the created socket. This path number is a handle to the socket, similar to what the `open()` or `create()` functions return.
The following syntax is used to create a stream socket:

```c
int s;
s=socket(AF_INET,SOCK_STREAM,0);
```

`s` is the returned path number of the socket.

**Step 2. Bind to the Socket**

When a socket is created with `socket()`, it has no association to local or destination addresses. This means a local port number is not assigned to the socket. Server processes operating on a well-known port must specify this port to the system by using the `bind()` LAN Communications Pak function. The `bind()` call binds the port to the address.

The structure used for the address is `sockaddr_in` which is defined in the `in.h` header file.

Normally a server process accepts connections that arrive on any of the machine's interfaces. In this case the IP address passed to `bind()` in the `sockaddr_in` structure is the wildcard address 0.0.0.0, which is represented by `INADDR_ANY`. If the server process wishes to restrict incoming connections to a single interface, the IP address associated with that interface may be used in place of `INADDR_ANY`.

The following syntax is valid for `bind()`:

```c
#define PORT 27000 /* port number to bind socket*/
struct sockaddr_in name; /* to define variable 'name' */
    /* as a sockaddr_in */
    /* structure */
memset(&name,0,sizeof(name)); /*initialize structure to zero*/
name.sin_family = AF_INET;
    /* address family is AF_INET */
name.sin_port = htons(PORT);/* assign our port number in network byte order */
name.sin_addr.s_addr = INADDR_ANY;
    /* allow any client to access this socket */
bind(s, (struct sockaddr*) &name, sizeof(name));
    /* bind the port to the socket.
    The 's' parameter is the path number for the socket and was returned by socket() */
```
Step 3. Listen for a Connecting Socket

If you have a stream socket (type **SOCK_STREAM**), the **listen()** function sets the socket in a passive state and allows servers to prepare a socket for incoming connections. It also informs the system to queue multiple client requests which arrive at a socket very close in time. This queue length must be specified in the **listen()** call. After client requests fill this queue, any further requests are rejected, and the client socket is notified with an **ECONNREFUSED** error.

**listen()** requires two parameters:

- The socket’s path number
- The client’s requested queue size

For example, the following syntax is valid for **listen()**:

```c
listen(s, 4);
```

In this example, four client requests are allowed to queue. When the queue is full, additional requests are refused.

Step 4. Accept a Connecting Socket

Once **listen()** sets up a passive socket, the **accept()** function is used to accept connections from clients. **accept()** blocks the process if pending connections are not presently queued.

If the socket is set up as non-blocking and an **accept()** call is made with no pending connections, **accept()** returns with an **EWOULDBLOCK** error.

**accept()** requires three parameters:

- The socket’s path number
- A pointer to a variable with type **sockaddr_in**
- The size of the **sockaddr_in** structure
The following syntax is valid for `accept()`:

```c
struct sockaddr_in from;
int ns, size;

size = sizeof(struct sockaddr_in);
ns = accept(s, (struct sockaddr*) &from, &size);
```

Once `accept()` returns without error, a connection has been made to a client process and the following events occur:

1. The client request is removed from the `listen` queue.
2. The client foreign address was put into the `from` variable. This allows the server to see who connected to it.
3. `accept()` returns a new socket path number (ns).

This new socket transfers data to and from the client process. It represents a connected path to the client process. The original socket path number returned by `socket()` can then be used to accept further connections or it can be closed, determined by the application.

At this point in the server process, the socket has been established and communication between the two processes can begin.
Client Steps

Follow these steps on the client side:

---

Step 1. Create the Socket

Sockets are created with the `socket()` function. `socket()` requires three parameters:

- An address format (`AF_INET`)
- A type (`SOCK_STREAM` or `SOCK_DGRAM`)
- A protocol (usually `0`)

`socket()` returns a path number to the created socket. This path number is a handle to the socket, similar to what the `open()` or `create()` functions return.

The following syntax is used to create a stream socket:

```c
int s;
s=socket(AF_INET,SOCK_STREAM,0);
```

`s` is the returned path number to the socket.

---

Step 2. Connect to a Listening Socket

To connect to a listening socket, use the `connect()` function call.

`connect()` requires three parameters:

- The path descriptor returned by `socket()`
- A pointer to a structure containing the name
- The length of the name
Using Connect

The following example demonstrates the use of connect().

```c
#define PORT 27000
/* Port number of server process */
struct sockaddr_in ls_addr;
memset(&ls_addr, 0, sizeof(ls_addr)); /* Initialize structure to zero */
ls_addr.sin_family = AF_INET;
ls_addr.sin_port = htons(PORT); /* Assign port in network byte order */
ls_addr.sin_addr.s_addr = inet_addr("thor"); /* IP address we wish to connect to */
connect (s, (struct sockaddr *)&ls_addr, sizeof(ls_addr));
```

Note
The connect() call automatically binds the socket to a local port number if bind() was not explicitly called. bind() is not usually called by clients unless the server restricts the ports from which it will accept connections.
Datagram Sockets

Datagram sockets use UDP as a transport protocol and can be connected or connectionless.

The steps for creating connected and connectionless sockets are the same for both the server and client sides:

1. Create the socket

   The following syntax is used to create a datagram socket:

   ```c
   int s;
   s=socket(AF_INET,SOCK_DGRAM,0);
   ```

2. Bind a port and/or address to the socket

   ```c
   struct sockaddr_in ls_addr;
   bind(s,(struct sockaddr*) &ls_addr, sizeof (ls_addr);
   ```

Note

As with TCP, the client does not need to bind a socket.

The following syntax is valid for bind():

```c
struct sockaddr_in ls_addr;
bind(s,(struct sockaddr*) &ls_addr, sizeof (ls_addr);
```

Connect a Socket

Either the client or server can optionally call `connect()` on a UDP socket. Unlike TCP sockets, calling `connect()` on a UDP socket does not cause any information to be sent to the peer. It simply stores the peer address in the local socket and allows the application to use `send()` and `recv()` instead of `sendto()` and `recvfrom()`.

A process may "disconnect" a UDP socket by calling `connect()` with `INADDR_ANY` as the IP address. The application can then no longer use `send()` and `receive()`, but can still use `sendto()` and `recvfrom()`.
The `connect()` function call requires three parameters:

- The path descriptor returned by `socket()`.
- A pointer to a `sockaddr_in` structure containing the peer port and IP address.
- The size of the `sockaddr_in` structure.

The following syntax is used to connect a datagram socket:

```c
struct sockaddr_in ls_addr;
connect(s, (struct sockaddr *)&ls_addr, sizeof(ls_addr));
```
# Header Files

<table>
<thead>
<tr>
<th>Header File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>netinet/in.h</code></td>
<td>Provides structures and functions for the socket family <code>AF_INET</code>.</td>
</tr>
<tr>
<td><code>netdb.h</code></td>
<td>Provides structures for hosts, networks, services, protocols, and functions calling socket address structures and socket family <code>AF_INET</code> structures.</td>
</tr>
<tr>
<td><code>sys/socket.h</code></td>
<td>Provides a definition to the socket address structure.</td>
</tr>
</tbody>
</table>

## Note

Header files are found in `MWOS/SRC/DEFS/SPF/bsd`. 
The main internet application structure is `sockaddr_in`, defined in the `in.h` header file. The structure is defined as follows:

```c
struct sockaddr_in {
    u_char         sin_len;
    u_char         sin_family;
    u_short        sin_port;
    struct in_addr sin_addr;
    char           sin_zero[8];
};
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:

```c
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_list; /* list of addresses from */
        /* name server */
    #define h_addr   h_addr_list[0]
        /* backwards compatibility: */
        /* pointer to the address of */
        /* the host */
};
```
Reading Data Using Sockets

Four functions are available for reading data:

Table 8-4 Reading Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_os_read()</td>
<td>Returns the data available on the specified socket path, up to the amount requested. For packet oriented protocols, such as UDP, only a single datagram is returned, even if more are available. These calls are only valid on connected sockets.</td>
</tr>
<tr>
<td>read()</td>
<td>Functions similar to recv() except that it also returns a sockaddr_in structure containing the address information of the sender. recvfrom() works with unconnected sockets.</td>
</tr>
<tr>
<td>recv()</td>
<td>Functions similar to recv() except that it also returns a sockaddr_in structure containing the address information of the sender. recvfrom() works with unconnected sockets.</td>
</tr>
</tbody>
</table>

For More Information

For more specific information on recv() and recvfrom(), refer to the LAN Communications Pak Programming Reference. For more specific information on _os_read() and read(), refer to the Ultra C Library Reference.
Writing Data Using Sockets

Four similar functions are available for writing data.

### Table 8-5  Writing Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>_os_write()</code></td>
<td>Writes data from a buffer to the socket path. These functions are only valid with connected sockets.</td>
</tr>
<tr>
<td>write()</td>
<td>Functions the same as <code>send()</code> except it also takes a <code>sockaddr_in</code> parameter specifying where the data is to be sent. <code>sendto()</code> works with unconnected sockets.</td>
</tr>
<tr>
<td>send()</td>
<td></td>
</tr>
<tr>
<td>sendto()</td>
<td></td>
</tr>
</tbody>
</table>

**Note**

If a socket path has been set to nonblocking and there is not enough buffer space available, the call can succeed, but only part of the data may be sent. Check the return value to see how many bytes were sent.

**For More Information**

For more specific information on `send()` and `sendto()`, refer to the *LAN Communications Pak Programming Reference*. For more specific information on `_os_write()` and `write()`, refer to the Hawk On-line Help System from the Hawk interface.
Setting up Non-Blocking Sockets

When a program tries to perform some socket functions, the program can block. The following are situations in which this would happen:

- Read from a socket that has no data (read/recv/recvfrom).
- Write to a socket that does not have enough space to satisfy size of write (write/send/sendto).
- Accept a connection with no waiting connection available (listen/accept).
- Calling connect on a TCP socket.

LAN Communications Pak enables you to create a non-blocking socket to prevent the above problems. When your program tries to access a non-blocking socket in one of the above listed blocking conditions, the socket returns the error EWOULDBLOCK. The connect call is the exception and returns EINPROGRESS.

The following function can be used to set a socket to non-blocking or blocking:

spath is the socket path
blockflag is either set to IO_SYNC for blocking, or IO_ASYNC for nonblocking.

```
#include <SPF/spf.h>

error_code setblock(path_id spath, u_int8 blockflag)
{
    struct spf_popts sopts;
    u_int32 soptsz=sizeof(sopts)

    if ((errno=_os_gs_popt(spath,&soptsz,&sopts)) !=SUCCESS) {
        return(errno);
    }
    sopts.pd_ioasync=blockflag;
    return (_os_ss_popt(spath,soptsz,&sopts));
} /*end of setblock*/
```
The common UNIX function `ioctl()` can also be used.

```c
#include <UNIX/ioctl.h>
error_code setblock(path_id spath, u_int8 blockflag)
{
    if (ioctl(spath, FIONBIO, dblockflag)<SUCCESS) {
        return (errno);
    }
    return(SUCCESS);
}
```
Broadcasting

Broadcasting support allows the network to deliver one copy of a packet to all attached hosts. Due to the extra overhead involved, multicasting is much preferred over broadcasting as a way to deliver data to multiple recipients. Also, broadcast packets are only sent to the local network and cannot be forwarded by routers.

Note
Stream sockets cannot be used for broadcasting messages. The SO_BROADCAST option must be enabled before broadcasting on a datagram socket.

Broadcasting Process

The broadcasting process sends its message to the pre-selected port number.

The broadcasting process uses the internet address for the address of the network. Usually this address is the same as the sender’s broadcast address. For example, on a class C network, a system with address 192.7.44.105 has a broadcast address of 192.7.44.255.

The broadcasting process must open the socket as a datagram socket. A bind() call is required before broadcasting.

The broadcasting process must use the sendto() function.

WARNING
Do not use send() or _os_write().
Receiving Process

The receiving processes must receive from the pre-selected port number the broadcaster is using.

The receiving socket must be opened as a datagram socket.

A `bind()` call is required before attempting to receive.

The receiving processes use the `recvfrom()` function.
Multicasting support allows the delivery of a single packet to multiple hosts. The main difference between multicasting and broadcasting is that with multicasts, only the machines interested in receiving the packets see them. Also, multicast packets can be routed across networks, while broadcasts are only seen on the local network.

Note
Currently, the LAN Communications Pak does not support routing multicast packets. It can only be used as an end node to send or receive them. Also, only datagram sockets can be used for multicasting.

Sending Multicasts

The same mechanism used to send unicast datagrams can be use to send multicasts by simply using a multicast destination IP address. However, to provide more control, 3 additional socket options may be used.

- **IP_MULTICAST_TTL**—This option limits the number of routers a multicast packet can pass through before being dropped. The default value is 1, which means multicasts will not travel beyond the locally attached network. Link local multicasts (addresses 224.0.0.0 - 224.0.0.255) are never routed, regardless of the TTL value.

- **IP_MULTICAST_LOOP**—The default is for multicasts not to be sent to the loopback interface. If you want to see the multicast packets you send, or another application on your hosts needs to see them, this option needs to be enabled.
• **IP_MULTICAST_IF**—When sending multicast packets, the output interface is selected via the normal routing procedure. This option allows an application to override this selection and pick the appropriate interface.

**Note**
The normal routing lookup is performed first and fails, the **IP_MULTICAST_IF** option is not checked. This means a route to the selected multicast address must exist in the routing table. Normally this consists of either a network route such as 224.0.0.0 or, more commonly, the presence of a default route.

---

**Receiving Multicasts**

In order for an application to receive multicast packets, it must join the appropriate multicast group. After the group has been joined, receiving multicast packets is no different than normal unicast packets. Two socket options are provided for joining and leaving multicast groups.

• **IP_ADD_MEMBERSHIP**—This option informs IP that an application wants to receive packets for the given multicast group. If this is the first application on the host to join the group on the requested interface, IP will notify the interface to begin receiving multicast packets for group. Additionally, any multicast routers present on the subnet will be notified that a host has joined the group.

• **IP_DROP_MEMBERSHIP**—This option informs IP that packets for the indicated group should no longer be sent to the application. If this is the last application receiving this group on the indicated interface, IP will notify the interface to no longer receive the group.
For More Information
See the *LAN Communications Pak Programming Reference* for more information on using `setsockopt`. 
Controlling Socket Operations

Socket level options control the socket’s operation. These options are defined in the `socket.h` header file. `setsockopt()` and `getsockopt()` are used to set and get options.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SO_DEBUG *</td>
<td>Turns on recording of debugging information. This allows debugging in the underlying protocol modules.</td>
</tr>
<tr>
<td>SO_REUSEADDR</td>
<td>Indicates the rules used in validating addresses supplied in a <code>bind()</code> call should allow reuse of local addresses.</td>
</tr>
<tr>
<td>SO_KEEPALIVE</td>
<td>Allows 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 the socket is closed.</td>
</tr>
<tr>
<td>SO_DONTROUTE</td>
<td>Indicates 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.</td>
</tr>
</tbody>
</table>
### Table 8-6  Socket Level Options (continued)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SO_LINGER</td>
<td>Controls the actions taken when unsent messages are queued on a socket and a <code>close()</code> is performed. If the socket promises reliable delivery of data and <code>SO_LINGER</code> 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 <em>linger interval</em>, is specified by <code>setsockopt()</code> when <code>SO_LINGER</code> is requested.</td>
</tr>
<tr>
<td>SO_BROADCAST</td>
<td>Enable an application to send broadcast messages. Valid for datagram sockets only.</td>
</tr>
<tr>
<td>SO_OOBINLINE</td>
<td>Place any incoming out-of-band data in the normal input queue.</td>
</tr>
<tr>
<td>SO_SNDBUF</td>
<td>Set or retrieve the size of the socket send buffer.</td>
</tr>
<tr>
<td>SO_RCVBUF</td>
<td>Set or retrieve the size of the socket receive buffer.</td>
</tr>
<tr>
<td>SO_SNDLOWAT</td>
<td>Indicates the minimum amount of space that must be available in the send buffer before data is accepted for sending. If this much space is not available the process blocks, or if the socket is non-blocking an <code>EWOULDBLOCK</code> error is returned.</td>
</tr>
<tr>
<td>SO_RCVLOWAT*</td>
<td>Indicates the minimum amount of data that must be available to be read before <code>select</code> considers a path &quot;readable&quot;.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SO_SNDTIMEO*</td>
<td>Control the maximum amount of time a process will block waiting for buffer space when sending data.</td>
</tr>
<tr>
<td>SO_RCVTIMEO*</td>
<td>Control the maximum amount of time a process will block waiting for incoming data.</td>
</tr>
<tr>
<td>SO_TYPE</td>
<td>Return the type of a socket such as SOCK_STREAM or SOCK_DGRAM. This option is only valid for getsockopt().</td>
</tr>
<tr>
<td>SO_ERROR</td>
<td>Retrieves the current socket error if one exists. This option is only valid for getsockopt().</td>
</tr>
<tr>
<td>SO_USELOOPBACK</td>
<td>This option is only valid for sockets in the routing domain (AF_ROUTE). It controls whether a process receives a copy of everything it sends.</td>
</tr>
</tbody>
</table>

* Unsupported option
Using LAN Communications Pak
Appendix A: Configuring LAN Communications Pak

This appendix explains how to configure and start the LAN Communications Pak for Internet access. It includes the following sections:

- Configuring Network Modules
- Starting the Protocol Stack
- Example Configuration
- Configuration Files
Configuring Network Modules

You can configure the host, network, DNS client, routing, and interface information by updating text files found in MWOS/SRC/ETC.

Step 1: Updating Files

You may need to update the following files for your system.

- hosts
- networks
- resolv.conf (if using DNS Client)
- routes.conf (if using static routing)
- interfaces.conf (located in MWOS/<OS>/<CPU>/PORTS/<BOARD>/SPF/ETC or MWOS/SRC/ETC)

Note
These files are described in later in this appendix.

Step 2: Creating Modules

Create the inetdb/inetdb2/rpcdb modules by running os9make on cross-development systems. This executes the idbgen and rpcdbgen utilities.

The makefile is found in MWOS/SRC/ETC. If you are working from a ports directory, the makefile is found in the MWOS/<OS>/<CPU>/PORTS/<TARGET>/SPF/ETC directory in the port.
The `inetdb/inetdb2/rpcdb` data modules will be placed in `MWOS/<OS>/<CPU>/CMDS/BOOTOBJSPF`, or the local ports `CMDS/BOOTOBJSPF`.

The `idbgen` utility is called, which reads files in the local SPF/ETC directory and/or files in `MWOS/SRC/ETC`. The `rpcdbgen` utility reads files from `MWOS/SRC/ETC`.

The `idbdump` utility can be used to print out the contents of the `inetdb/inetdb2` data modules. The `rpdump` utility can be used to print out the contents of the `rpcdb` data module.

---

**For More Information**

The `rpcdbgen` utility and `rpcdb` data module are described in the Using Network File System/Remote Procedure Call manual.

---

**Contents of inetdb**

Below is an example of the contents of an `inetdb` data module.

```
> idbdump inetdb
Dump of OS-9000/PowerPC INETDB network database module [inetdb]
    Compatibility : 1
    Version Number: 9

Host Entries:

<table>
<thead>
<tr>
<th>Host Name</th>
<th>Host Address</th>
<th>Host Aliases</th>
</tr>
</thead>
<tbody>
<tr>
<td>localhost</td>
<td>127.0.0.1</td>
<td>me</td>
</tr>
</tbody>
</table>

Hosts Equivalent Entries:

Network Entries:

<table>
<thead>
<tr>
<th>Network Name</th>
<th>Network Address</th>
<th>Network Aliases</th>
</tr>
</thead>
<tbody>
<tr>
<td>loopback</td>
<td>127</td>
<td></td>
</tr>
<tr>
<td>private-A</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>private-B</td>
<td>172.1</td>
<td></td>
</tr>
<tr>
<td>private-C</td>
<td>192.1.2</td>
<td></td>
</tr>
<tr>
<td>localnet</td>
<td>10.0.0</td>
<td></td>
</tr>
</tbody>
</table>
```
### Protocol Entries:

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Number</th>
<th>Protocol Aliases</th>
</tr>
</thead>
<tbody>
<tr>
<td>ip</td>
<td>0</td>
<td>IP</td>
</tr>
<tr>
<td>icmp</td>
<td>1</td>
<td>ICMP</td>
</tr>
<tr>
<td>igmp</td>
<td>2</td>
<td>IGMP</td>
</tr>
<tr>
<td>tcp</td>
<td>6</td>
<td>TCP</td>
</tr>
<tr>
<td>udp</td>
<td>17</td>
<td>UDP</td>
</tr>
</tbody>
</table>

### Service Entries:

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Port/Protocol</th>
<th>Service Aliases</th>
</tr>
</thead>
<tbody>
<tr>
<td>ndp</td>
<td>13312/tcp</td>
<td>ndpd</td>
</tr>
<tr>
<td>npp</td>
<td>13568/tcp</td>
<td>nppd</td>
</tr>
<tr>
<td>echo</td>
<td>7/tcp</td>
<td></td>
</tr>
<tr>
<td>echo</td>
<td>7/udp</td>
<td></td>
</tr>
<tr>
<td>discard</td>
<td>9/tcp</td>
<td></td>
</tr>
<tr>
<td>discard</td>
<td>9/udp</td>
<td></td>
</tr>
<tr>
<td>daytime</td>
<td>13/tcp</td>
<td></td>
</tr>
<tr>
<td>daytime</td>
<td>13/udp</td>
<td></td>
</tr>
<tr>
<td>chargen</td>
<td>19/tcp</td>
<td></td>
</tr>
<tr>
<td>chargen</td>
<td>19/udp</td>
<td></td>
</tr>
<tr>
<td>ftp-data</td>
<td>20/tcp</td>
<td></td>
</tr>
<tr>
<td>ftp-data</td>
<td>20/udp</td>
<td></td>
</tr>
<tr>
<td>ftp</td>
<td>21/tcp</td>
<td></td>
</tr>
<tr>
<td>ftp</td>
<td>21/udp</td>
<td></td>
</tr>
<tr>
<td>telnet</td>
<td>23/tcp</td>
<td></td>
</tr>
<tr>
<td>telnet</td>
<td>23/udp</td>
<td></td>
</tr>
<tr>
<td>nameserver</td>
<td>42/tcp</td>
<td></td>
</tr>
<tr>
<td>nameserver</td>
<td>42/udp</td>
<td></td>
</tr>
<tr>
<td>bootps</td>
<td>67/tcp</td>
<td></td>
</tr>
<tr>
<td>bootps</td>
<td>67/udp</td>
<td></td>
</tr>
<tr>
<td>bootpc</td>
<td>68/tcp</td>
<td></td>
</tr>
<tr>
<td>bootpc</td>
<td>68/udp</td>
<td></td>
</tr>
<tr>
<td>tftp</td>
<td>69/tcp</td>
<td></td>
</tr>
<tr>
<td>tftp</td>
<td>69/udp</td>
<td></td>
</tr>
<tr>
<td>touyr</td>
<td>520/udp</td>
<td></td>
</tr>
</tbody>
</table>

### InetD Configuration Entries:

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Socket Type</th>
<th>Protocol</th>
<th>Flags</th>
<th>Owner</th>
<th>Server Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftp</td>
<td>SOCK_STREAM</td>
<td>tcp</td>
<td>WAIT</td>
<td>0.0</td>
<td>ftpdc</td>
</tr>
<tr>
<td>telnet</td>
<td>SOCK_STREAM</td>
<td>tcp</td>
<td>WAIT</td>
<td>0.0</td>
<td>telnetdc</td>
</tr>
<tr>
<td>echo</td>
<td>SOCK_STREAM</td>
<td>tcp</td>
<td>WAIT</td>
<td>0.0</td>
<td>internal</td>
</tr>
<tr>
<td>echo</td>
<td>SOCK_DGRAM</td>
<td>udp</td>
<td>WAIT</td>
<td>0.0</td>
<td>internal</td>
</tr>
</tbody>
</table>
### Resolve Configuration Entries:

- **Domain Name:** alpha.com
- **Nameserver List:**
  - 1: 10.0.0.1
  - 2: 10.0.0.2
  - 3: 10.0.0.3
- **Search List:**
  - 1: alpha.com

### Host Configuration Entries:

### Interface Configuration Entries:

### Hostname Configuration Entries:

### Route Configuration Entries:

<table>
<thead>
<tr>
<th>Destination</th>
<th>Gateway</th>
<th>Netmask</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### RPC Entries:

<table>
<thead>
<tr>
<th>Program</th>
<th>Number</th>
<th>Aliases</th>
</tr>
</thead>
<tbody>
<tr>
<td>portmapper</td>
<td>100000</td>
<td>rpcinfo</td>
</tr>
<tr>
<td>rstatd</td>
<td>100001</td>
<td>rup</td>
</tr>
<tr>
<td>rusersd</td>
<td>100002</td>
<td>rusers</td>
</tr>
<tr>
<td>nfs</td>
<td>100003</td>
<td>nfsrbuf</td>
</tr>
<tr>
<td>ypserv</td>
<td>100004</td>
<td>yp</td>
</tr>
<tr>
<td>mountd</td>
<td>100005</td>
<td>mount showmount</td>
</tr>
<tr>
<td>ypbinder</td>
<td>100007</td>
<td></td>
</tr>
<tr>
<td>wallld</td>
<td>100008</td>
<td>rwall shutdown</td>
</tr>
<tr>
<td>yppasswdd</td>
<td>100009</td>
<td>yppasswd</td>
</tr>
<tr>
<td>etherstatd</td>
<td>100010</td>
<td>etherstat</td>
</tr>
<tr>
<td>rquotad</td>
<td>100011</td>
<td>rquotaprog quota rquota</td>
</tr>
<tr>
<td>sprayd</td>
<td>100012</td>
<td>spray</td>
</tr>
<tr>
<td>rje_mapper</td>
<td>100014</td>
<td></td>
</tr>
<tr>
<td>selection_svc</td>
<td>100015</td>
<td>selnsvc</td>
</tr>
<tr>
<td>database_svc</td>
<td>100016</td>
<td></td>
</tr>
<tr>
<td>rexd</td>
<td>100017</td>
<td>on</td>
</tr>
<tr>
<td>llockmgr</td>
<td>100020</td>
<td></td>
</tr>
<tr>
<td>nlockmgr</td>
<td>100021</td>
<td></td>
</tr>
<tr>
<td>statmon</td>
<td>100023</td>
<td></td>
</tr>
<tr>
<td>status</td>
<td>100024</td>
<td></td>
</tr>
<tr>
<td>bootparam</td>
<td>100026</td>
<td></td>
</tr>
<tr>
<td>ypupdated</td>
<td>100028</td>
<td>ypupdate</td>
</tr>
<tr>
<td>keyserv</td>
<td>100029</td>
<td>keyserv</td>
</tr>
<tr>
<td>dird</td>
<td>76</td>
<td>rdir</td>
</tr>
<tr>
<td>msgd</td>
<td>99</td>
<td>msg</td>
</tr>
<tr>
<td>sortd</td>
<td>22855</td>
<td>rsort</td>
</tr>
</tbody>
</table>
Contents of inetdb2

Below is an example of the inetdb2 data module.

```shell
/idmbdump inetdb2
Dump of OS-9000/PowerPC INETDB network database module [inetdb2]
    Compatibility : 1
    Version Number: 9
Host Entries:
    Host Name     Host Address     Host Aliases
    ----------------     ---------------     -----------------------------
Hosts Equivalent Entries:
Network Entries:
    Network Name     Network Address     Network Aliases
    ----------------     ---------------     -----------------------------
Protocol Entries:
    Protocol    Number     Protocol Aliases
    ----------     ------     ------------------------------------------------
Service Entries:
    Service Name     Port/Protocol     Service Aliases
    ----------------     -----------------     --------------------------------
InetD Configuration Entries:
    Service Name Socket Type  Protocol   Flags   Owner  Server Arguments
    ------------- -----------  ---------  ------  ------ ----------------
Resolve Configuration Entries:
Host Configuration Entries:
Interface Configuration Entries:
Interface: enet0
    Binding: /spe30/enet
    Flags: 0x1 <UP>
    MW_Flags: 0x8000 <NO_MULTICAST>
    MTU: 0
    Metric: 0
    Address         Netmask         Broadcast
    172.16.4.32     0.0.0.0         0.0.0.0
Interface: ppp0
    Binding: /ipcp0
    Flags: 0x1 <UP>
    MTU: 0
    Metric: 0
    Address         Netmask         Broadcast
    None
Hostname Configuration Entries:
Hostname: Beta
Route Configuration Entries:
Destination     Gateway         Netmask         Type
    ----------------     ---------------     -------------     ----
RPC Entries:
    Program     Number     Aliases
```
Step 3: Configure the Interface Descriptor

To set port-specific parameters such as baud rate or interrupt number, you must configure the interface descriptor for PPP, SLIP, or Ethernet. To do this, perform the following steps:

Step 1. Update the `spf_desc.h` descriptor file in the local ports directory. For Ethernet, this file is found in the port directory for your target under `MWOS/<OS>/<CPU>/PORTS/<TARGET>/SPF/<driver>/DEFS`.

Step 2. To make, run `os9make` in the `SPF/<driver>` directory.

The following list provides the supported Ethernet devices for LAN Communications Pak.

### Supported Ethernet Devices for 68000

- **MVME147.** Support for the AM7990 chip is available. The `spl47` driver and `spie0` descriptor can be found in:
  - `MWOS/OS9/68020/PORTS/MVME147/CMDS/BOOTOBJ/SFP`
  - The driver and descriptor can be compiled in:
  - `MWOS/OS9/68020/PORTS/MVME147/SPF/SPLANCE`

- **MVME162/167/172/177.** Support for the I82596 chip is available. The `spl62`/`spl67`/`spl72`/`spl77` driver and `spie0` descriptor can be found in:
  - `MWOS/OS9/68020/PORTS/MVME162/CMDS/BOOTOBJ/SFP`
  - `MWOS/OS9/68020/PORTS/MVME167/CMDS/BOOTOBJ/SFP`
  - `MWOS/OS9/68060/PORTS/MVME172/CMDS/BOOTOBJ/SFP`
  - `MWOS/OS9/68060/PORTS/MVME177/CMDS/BOOTOBJ/SFP`
  - The driver and descriptor can be compiled in:
  - `MWOS/OS9/<PROCESSOR>/PORTS/<TARGET>/SPF/SP82596`

- **QUADS.** Support for the QUICC chip is available. The `sp360` driver and `spqe0` descriptor can be found in:
A Configuring LAN Communications Pak

MWOS/OS9/CPU32/PORTS/QUADS/CMDS/BOOTOBJ/S

The driver and descriptor can be compiled in:
MWOS/OS9/CPU32/PORTS/QUADS/SPF/SPQUICC

Supported Ethernet Devices for PPC

• 821ADS. Support for the QUICC chip is available. The sp821 driver
  and spqe0 descriptor can be found in:
MWOS/OS9000/821/PORTS/821ADS/CMDS/BOOTOBJ/S

The driver and descriptor can be recompiled in:
MWOS/OS9000/821/PORTS/821ADS/SPF/SPQUICC

Supported Ethernet Devices for 80X86

• PCAT. Support for the 3COM Etherlink III, SMC Ultra, and SMC Elite
  is available. The drivers include spe509 (3Com), sp8390 (elite),
  sp83c790 (ultra). The descriptors include spe30 (3Com), spwd0
  (elite), ns0 (ultra).

The drivers and descriptors can be found in:
MWOS/OS9000/80386/PORTS/PCAT/CMDS/BOOTOBJ/S

The driver and descriptor can be recompiled in:
MWOS/OS9000/80386/PORTS/PCAT/SPF/SP509
MWOS/OS9000/80386/PORTS/PCAT/SPF/SP8390
MWOS/OS9000/80386/PORTS/PCAT/SPF/SP83C790
### Step 4: Load LAN Communications Pak modules

The minimum modules required to test connectivity with ping are included in the following list. Uncomment or add these files to your bootlist.

- `inetdb`  
- `inetdb2`  
- `SysMbuf`  
- `mbinstall`  
- `pkman`  
- `pks` (OS-9 68K only)  
- `pkdvr`  
- `ip0`  
- `spip`  
- `tcp0`  
- `spudp`  
- `sptcp`  
- `udp0`  
- `spf`  
- `netdb_local` or `netdb_dns`  
- `ipstart`  
- `spraw`  
- `raw0`  
- `ping`

- For SLIP, PPP, or Ethernet support, uncomment the appropriate driver(s) and descriptor(s).

- To include other utilities in your boot, such as telnet, uncomment the appropriate entry.

An example bootlist follows. Depending on the OS software version you are using, you may need a relative path of:

```
.../.../.../.../.../<CPU> or .../.../.../<CPU>.
```

*  
* SysMbuf P2 Module:  
*  
```
./.../.../.../.../<CPU>/CMDS/BOOTOBS/SPF/SysMbuf
```
*  
* SysMbuf utilities:  
*  
```
*./.../.../.../.../<CPU>/CMDS/mbinstall  
./.../.../.../.../<CPU>/CMDS/mbinstall_csl  
*./.../.../.../.../<CPU>/CMDS/mbdump
```
*  
* SPF file manager:  
*  
```
*./.../.../.../.../<CPU>/CMDS/BOOTOBS/SPF/spf
```
*  
* LAN protocol drivers and descriptors:  
*  
```
*./.../.../.../.../<CPU>/CMDS/BOOTOBS/SPF/sptcp
```
Configuring LAN Communications Pak

* Pseudo Key Board File Manager and Driver required for telnetdc)

* LAN trap handler and configuration data module:

* LAN SLIP drivers and descriptors:

* LAN PPP client drivers and descriptors:

* LAN PPP client utilities:

* Ethernet Protocol Driver (required for hardware ethernet drivers)

* Ethernet Protocol Driver
Configuring LAN Communications Pak

* LAN ethernet driver and descriptor:
  * ../../../CMDS/BOOTOBJS/SPF/<ethernet driver>
  ../../../CMDS/BOOTOBJS/SPF/<ethernet descriptor>
* LAN utilities:
  *../../../../<CPU>/CMDS/arp
  *../../../../<CPU>/CMDS/bootpd
  *../../../../<CPU>/CMDS/bootptest
  *../../../../<CPU>/CMDS/dhcp
  *../../../../<CPU>/CMDS/ftp
  *../../../../<CPU>/CMDS/ftpd
  *../../../../<CPU>/CMDS/ftpdc
  *../../../../<CPU>/CMDS/hostname
  *../../../../<CPU>/CMDS/idbdump
  *../../../../<CPU>/CMDS/idbgen
  *../../../../<CPU>/CMDS/ifconfig
  *../../../../<CPU>/CMDS/inetd
  *../../../../<CPU>/CMDS/ipstart
  *../../../../<CPU>/CMDS/ndbmod
  *../../../../<CPU>/CMDS/netstat
  *../../../../<CPU>/CMDS/ping
  *../../../../<CPU>/CMDS/route
  *../../../../<CPU>/CMDS/routed
  *../../../../<CPU>/CMDS/telnet
  *../../../../<CPU>/CMDS/telnetd
  *../../../../<CPU>/CMDS/telnetdc
  *../../../../<CPU>/CMDS/tftpd
  *../../../../<CPU>/CMDS/tftpdc

Also available is the example loadspf script in MWOS/SRC/SYS. This file provides an example of the modules to load in order to start the protocol stack.

If you are using a disk-based system, the following file can be used as an example or modified to match your system. This file can be copied from MWOS/SRC/SYS.

* loadspf for SPF LAN Communication Package Release
* *
* Load SPF System Modules
* chd CMDS/BOOTOBJS/SPF
*
load -d inetdb inetdb2 ;* Load system specific inetdb modules
load -d SysMbuf ;* System Mbuf module
 *(sets size of mbuf pool on OS-9)
load -d pkman pkdvr pk ;* Pseudo keyboard modules (required for telnetdc)
*load -d pks; ;* Pseudo keyboard additional for 68K
 *
load -d spf ;* SPF file manager
load -d spip ip0 ;* IP driver and descriptor
load -d sptcp tcp0 ;* TCP driver and descriptor
load -d spudp udp0 ;* UDP driver and descriptor
load -d spraw raw0 ;* RAW IP driver and descriptor
*load -d sproute route0 ;* Dynamic Routing driver and descriptor
 *
* Load SPF Trap library and Commands
* Load one of the following Netdb name resolution trap handlers
*load -d netdb_local ;* Load trap handler for local name resolution
*load -d netdb_dns ;* Load trap handler for DNS name resolution
 *
* Load SPF Ethernet Drivers and Descriptors
*load -d spenet enet ;* Ethernet Protocol driver and descriptor (required by hardware Ethernet drivers)
 *
* Load SPF Drivers and Descriptors ... Uncomment those needed
*<Ethernet drivers and descriptors>
 *
* Serial Drivers and Descriptors
*load -d spsli p s p s l 0 ;* Slip /t1
*load -d spipcp opc p 0 ;* PPP IPCP
*load -d sp1cp lcp0 ;* PPP LCP
*load -d spdhlc hdlc0 ;* PPP HDLC
*(chd ..; load -d chat pppd ppplog ppauth; chd BOOTOBS/SPF) ;
*PPP Utilities
*
* Chd up to CMDS directory
* chd ..../.. load -d mbindall  ;* Load mbindall memory
* handler (or can be done
* within init
load -d ipstart  ;* Load ipstart stack
* initializer
*
*load -d routed  ;* Dynamic routing daemon
*load -d telnet telnetd telnetdc ;* Telnet support
* modules
*load -d ftp ftpd ftpdc ;* FTP support modules
*load -d tftpd tftpdc bootpd ;* Bootp/TFTP support
* modules
*load -d inetd  ;* Super-Server Daemon
*load -d idbgen idbdump ndbmod ;* Development tools
*load -d route hostname ifconfig arp ;* Runtime tools
load -d netstat ping  ;* Statistics/
* verification tools
Starting the Protocol Stack

Step 1. Install the network or SPF memory handler.

After loading the modules, the first step is to install the network or SPF memory buffer handler (SysMbuf). This can be done with the `mbinstall` utility.

```
shell> mbinstall
```

Step 2. Start the TCP/IP protocol stack by executing the `ipstart` utility.

```
shell> ipstart
```

If you did not define your interfaces in `inetdb2`, you can add them now using `ifconfig`.

Running `devs` and `procs` shows the protocol drivers initialized and SPF receive thread process in the process table.

Step 3. Use the `ping` utility to verify that the network components are working correctly.

```
shell> ping localhost
```

The following appears on your screen:

PING localhost (127.0.0.1): 56 data bytes
64 bytes from 127.0.0.1: ttl=255 time=0 ms

Next test to another system.

```
shell> ping <hostname>
```

Where `<hostname>` is the name of a computer in the `inetdb` module (hosts section), or a name that can be resolved by the DNS server specified in the `resolve.conf` section of `inetdb`. `<hostname>` can also be an IP address.

Something similar to the following appears on your screen:

PING delta.microware.com (172.16.1.40): 56 data bytes
64 bytes from 172.16.1.40: ttl=255 time=10 ms
Example Configuration

If you are using a disk-based system, the following `startspf` file can be used as an example or modified to match your system. This file can be copied from `MWOS/SRC/SYS`.

```
* startspf
* Shell Script to Start SPF System
* Set default directories before starting daemon programs
  chd /h0
  chx /h0/cmds
* Load SPF modules
  SYS/loadspf
* Load and start mbuf handler  (May be done via p2 list in init module)
  Allow for error returned in case sysmbuf is already initialized.
  -nx
  mbinstall
  * Start SPF system using ipstart
  * ipstart
* Add interfaces not specified in inetdb2
  * ifconfig enet0 <my_address> binding /<dev>/enet
  * ifconfig ppp0 binding /ipcp0
* Add any static routes. Even if running routed it may be useful
  to add multicast routes.
  * route add -net 224.0.0.0 <my_address>
* Start service daemons
  * routed: Dynamic routing server
  * inetd: FTP/Telnet and other protocols server
  * telnetd: Remote terminal server
  * ftfd: Remote file-transfer server (FTP)
  * bootpd: Network boot protocol server
  * tftp: Trivial file transfer protocol server
  routed >>>/nil
  inetd >>>/nil
  *telnetd >>>/nil &
  *ftpd >>>/nil &
  *bootpd /h0/TFTPBOOT/bootptab >>>/nil
  *tftp /h0/TFTPBOOT >>>/nil &
* spfndpd: Hawk User state debugging daemon
* spfnppd: Hawk Profiling daemon
  spfndpd >>>/nil &
  *spfnppd >>>/nil &
```
Configuration Files

Files identified in Table A-1 on page 232 reside in MWOS/SRC/ETC and provide the protocol stack with pertinent information. Utilities available to create, modify, or dump the inetdb modules are: idbgen, ndbmod, and idbdump, respectively.

Table A-1 Networking Files

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hosts</td>
<td>A list of hosts known to your system. If using DNS, this file may not need to be updated. Otherwise, add an entry for each of your hosts (including the host you are using) to the file.</td>
</tr>
<tr>
<td>networks</td>
<td>A list of networks analogous to hosts.</td>
</tr>
<tr>
<td>protocols</td>
<td>A list of protocols available.</td>
</tr>
<tr>
<td>services</td>
<td>A list of services available.</td>
</tr>
<tr>
<td>inetd.conf</td>
<td>A list of server daemon routines inetd supports (for example, telnet and FTP).</td>
</tr>
<tr>
<td>resolv.conf</td>
<td>Configuration information for a Domain Name System (DNS).</td>
</tr>
<tr>
<td>interfaces.conf</td>
<td>Configuration information for hostname and network interfaces to initialize when the stack is brought up.</td>
</tr>
<tr>
<td>routes.conf</td>
<td>List of static routes to add when the stack is brought up.</td>
</tr>
<tr>
<td>rpc</td>
<td>List of RPC support services, program number, and the client program.</td>
</tr>
</tbody>
</table>
Each file contains single-line entries consisting of one or more fields and (optionally) comments. Fields are separated by any number of spaces 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. All files are described in the following sections.

**Hosts**

The `hosts` file contains information regarding the known hosts on the internet. For each host, a single-line entry must be present. Each entry contains the following:

- Internet address
- Official host name
- Aliases (optional)

Internet addresses are specified in the conventional dot notation. Host names can contain any printable character other than a field delimiter, new line, or comment character.

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

```
192.1.1.1 balin #documentation
```

**Networks**

The `networks` file contains information regarding the known networks composing the internet. A single line entry must 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 dot notation. Network names can contain any printable character other than a field delimiter, new line, or comment character.
The following example `networks` entry consists of a name, number, alias, and comment:

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

**Note**
Consult your local network administrator for conventions to determine the proper host and network information.

---

**Protocols**

The `protocols` file contains information regarding the known protocols used in the internet. A single-line entry must 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, new line, or comment character.

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

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

**Services**

The `services` file contains information regarding known services available to your system. 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 must be present in the services file for each service. Each entry contains the following information:

- Official service name
- Port number at which the service resides
- Official protocol name
- Aliases (optional)

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

The port number and protocol name are considered a single item; a slash character (/) separates them.

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

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

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

**inetd.conf Configuration File**

The inetd.conf file contains information regarding program services the inetd service daemon handles. inetd can currently take the place of ftpd and telnetd.

A single-line entry must be present in the inetd.conf file for each service available. Entries contains the following information:

- service name (program)
- socket type
- protocol
- flags
Configuring LAN Communications Pak

- user
- server path name (child process to fork)
- [additional arguments]

The following example `inetd.conf` entry consists of a service name, socket type, protocol, flags, user, and server path name.

```
ftp    stream tcp    wait    root    ftpdc
```

**resolv.conf Configuration File**

A resolver finds the IP address of a host—given the host name—by using the Domain Name System (DNS). The `resolv.conf` file contains the necessary information to use DNS. Consult your local network administrator for local conventions.

The `resolv.conf` file contains three main sections:

- local domain name
- name server list
- optional domain search list

The following example `resolv.conf` listing contains these three sections.

```bash
# NAME RESOLUTION CONFIGURATION
#
# format: <keyword> <value>
# see keyword explanations for specific formatting
#
#
# local domain name (1): domain <DomainName>
# domain test.com
#
# ordered local nameserver list (1-3): nameserver <IPAddress>
# nameserver 192.1.1.1
# nameserver 192.1.1.2
# nameserver 192.1.1.3
```
#optional domain search list
#   <Domain1> [<Domain2> ...]
#
search test.com

**interfaces.conf Configuration File**

The *interfaces.conf* file contains the hostname and interfaces to initialize when `ipstart` is run. The entry for the host name is a string and may be up to 64 characters long.

The entry for the interface list may contain the following information:

- interface name
- **address** keyword and IP address of interface
- **broadcast** or **destaddr** keyword and broadcast or destination IP address
- **binding** keyword and device list
- optional values
  - `[mtu <mtu>]` [metric <metric>] [up|down]
  - `[netmask<mask>]` [iff_bROADCAST] [iff_pointopoint]
  - `[iff_nomulticast]` [iff_nobroadcast]
  - [iff_nopointopoint]

The following *interfaces.conf* entry contains the initialization information for a SLIP device:

```
slip0 address 10.0.0.1 destaddr 10.0.0.2 binding /spsl0
```
routes.conf Configuration File

The `routes.conf` file contains a static list of default, host, and network routes to be initialized when the stack is started. The entry for the route list contains a keyword and appropriate addresses as follows:

- default keyword and IP address of gateway network IP
- host keyword, host IP address, and gateway IP address
- network keyword, network IP address, gateway IP address, and optional network mask

There may be multiple host and network routes, but only one default route.

The following `routes.conf` entry contains the static host route entry to get to IP address 192.2.2.1 by going through router 192.1.1.2:

```
host 192.2.2.1 192.1.1.2
network 10.0.0.0 192.2.3.3
network 172.16.40.0 192.2.3.3 255.255.255.0
```

rpc Configuration File

The `rpc` file contains a list of supported RPC services, associated program numbers, and the client program. Update this table to register services and program numbers with `portmap`.

The following `rpc` entry contains the service mapping for `rstatd`:

```
rstatd 100001 rup
```
<table>
<thead>
<tr>
<th>Symbols</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(SPSLIP) Driver, Serial Line Internet Protocol</td>
<td>37</td>
</tr>
<tr>
<td>Address</td>
<td>114</td>
</tr>
<tr>
<td>Address Families</td>
<td>191</td>
</tr>
<tr>
<td>Addressing, ITEM</td>
<td>90, 93, 96, 99, 102</td>
</tr>
<tr>
<td>Addressing, Network</td>
<td>17</td>
</tr>
<tr>
<td>API Calls, Application Return Codes from</td>
<td>78, 91, 94, 97, 100</td>
</tr>
<tr>
<td>Application Return Codes from API Calls</td>
<td>78, 91, 94, 97, 100</td>
</tr>
<tr>
<td>Arguments, PPP Daemon Command Line</td>
<td>57</td>
</tr>
<tr>
<td>Arguments, PPP Daemon Command Line</td>
<td>57</td>
</tr>
<tr>
<td>Arguments, PPP Daemon Command Line</td>
<td>57</td>
</tr>
<tr>
<td>Arguments, PPP Daemon Command Line</td>
<td>57</td>
</tr>
<tr>
<td>Arguments, PPP Daemon Command Line</td>
<td>57</td>
</tr>
<tr>
<td>Arguments, PPP Daemon Command Line</td>
<td>57</td>
</tr>
<tr>
<td>Authentication, Setup</td>
<td>71</td>
</tr>
<tr>
<td>Available Network Protocols</td>
<td>14</td>
</tr>
<tr>
<td>Basic Datagram</td>
<td>11</td>
</tr>
<tr>
<td>Basic Networking Terminology</td>
<td>10</td>
</tr>
<tr>
<td>Baud Rates</td>
<td>44</td>
</tr>
<tr>
<td>Berkeley Sockets, Header Files Associated with</td>
<td>202</td>
</tr>
<tr>
<td>Bootfile Size</td>
<td>115</td>
</tr>
<tr>
<td>BOOTP Request Matrix</td>
<td>115</td>
</tr>
<tr>
<td>BOOTP Server</td>
<td>107</td>
</tr>
<tr>
<td>BOOTP Server Utilities</td>
<td>109</td>
</tr>
<tr>
<td>Bootp Tags</td>
<td>112</td>
</tr>
<tr>
<td>bootpd</td>
<td>129</td>
</tr>
<tr>
<td>bootptab Configuration File Setup</td>
<td>112</td>
</tr>
<tr>
<td>bootptab File Example</td>
<td>117</td>
</tr>
<tr>
<td>bootptest</td>
<td>131</td>
</tr>
<tr>
<td>Bootstrap Protocol</td>
<td>108</td>
</tr>
<tr>
<td>Broadcasting on Internet</td>
<td>208</td>
</tr>
</tbody>
</table>
Broadcasting Process 208

Capture Parameters 179
chat Example 66
Chat Script Commands 63, 64
Chat Script Commands 64
Chat Script, Prepare 71
Chat Script, Run 72
Classes, Network 17
Client and Server 13
Client Machine, Setting Up the 70
Client Steps 198
Command Line Arguments, PPP Daemon 57
Commands 126
Commands, Chat Script 63, 64
Commands, SPF_SS_IOCTL 82
Compression, Header 42
Configuration File
  inetd.conf 235
  interfaces.conf 237
  resolv.conf 236
  routes.conf 238
  rpc 238
Connect a Socket 200
Connect, Using 199
Connection Handlers, Daemon Server Programs and 122
Contents of inetdb 219
Contents of inetdb2 222
Controlling Socket Operations 213
Conventions, File Naming 136

Daemon Server Programs and Connection Handlers 122
Daemon Utility, PPP 57
Data Flow through HDLC Framer 52
Data Reception And Transmission Characteristics 98
Data Reception and Transmission Characteristics 76, 89, 92, 95, 101
Data, Reading 42, 53
Data, Sending 41, 53
Datagram Example, Encapsulated 12
Datagram Sockets 192, 200
Datagram, Basic 11
Datagrams 10
Datagrams, IP 16
Default Descriptor Values
for spenet 102
For spip 77
for spraw 90
for sproute 93
for sptcp 95
For spudp 99
Description 125
Description, Software 27
Descriptor, IP Driver and 76
Descriptor, SPRAW Driver and 89
Descriptor, SPROUTE Driver and 92
Descriptor, spslip Device 43
Descriptor, TCP Driver and 95
Descriptor, UDP Driver and 98
Device Descriptor, spslip 43
dhcp 133
Disk-Based 125
Domain Name Server
  DNS 232
Driver and Descriptor, SPRAW 89
Driver and Descriptor, SPROUTE 92
Driver and Descriptor, TCP 95
Driver and Descriptor, UDP 98
Driver, Serial Line Internet Protocol (SPLIP) 37
Drivers above SPIP 79
Drivers below SPIP 79
Drivers, Protocol 47, 75

Embedded 125
<p>| A          | B          | C          | D          | E          | F          | G          | H          | I          | J          | K          | L          | M          | N          | O          | P          | Q          | R          | S          | T          | U          | V          | W          | X          | Y          | Z          |
|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|
| Encapsulated Datagram Example | 12         | Encapsulation | 12         | ENET_GS_STATS | 104        | Establishing a Socket | 194        | Ethernet Devices | 68000, Supported | 223        | 80X86, Supported | 224        | PPC, Supported | 224        | Extended | 125       |
|            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |
| Families, Address | 191        | file       |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |
|            |            | hosts      | 233        |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |
|            |            | networks   | 233        |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |
|            |            | protocols  | 234        |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |
|            |            | services   | 234        |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |
| File Naming Conventions | 136        |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |
| File Setup, bootptab Configuration | 112        |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |
| File Transfer Protocols | 137        |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |
| Files, Networking | 232        |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |
| Flags, Mapping Between Letters and | 168        |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |
| Fragmentation | 11         |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |
| ftp         | 135        |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |
| ftpd        | 145        |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |
| ftpdc       | 147        |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |
| Functions, Reading | 204        |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |
| Functions, Writing | 205        |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |
| Gateway Connected Networks | 10         |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |
| Getstats and Setstats Above SPIP | 80         |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |
| Getstats and Setstats Below SPIP | 85         |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |
| Getstats for SPENET | 103        |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |
| Hardware Type | 113        |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |            |</p>
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDLC Framer, Data Flow through</td>
<td>52</td>
</tr>
<tr>
<td>Header Compression</td>
<td>42</td>
</tr>
<tr>
<td>Header Files</td>
<td>202</td>
</tr>
<tr>
<td>Header Files Associated with Berkeley Sockets</td>
<td>202</td>
</tr>
<tr>
<td>Host Name, Home Directory, and Bootfile</td>
<td>114</td>
</tr>
<tr>
<td>Host Name, Sending a</td>
<td>116</td>
</tr>
<tr>
<td>Hosts</td>
<td>233</td>
</tr>
<tr>
<td>hosts file</td>
<td>233</td>
</tr>
<tr>
<td>example entry</td>
<td>233</td>
</tr>
<tr>
<td>idbdump</td>
<td>149</td>
</tr>
<tr>
<td>idbgen</td>
<td>150</td>
</tr>
<tr>
<td>ifconfig</td>
<td>153</td>
</tr>
<tr>
<td>inetd</td>
<td>158</td>
</tr>
<tr>
<td>inetd.conf</td>
<td>232, 235</td>
</tr>
<tr>
<td>inetd.conf Configuration File</td>
<td>235</td>
</tr>
<tr>
<td>inetdb, Contents of</td>
<td>219</td>
</tr>
<tr>
<td>inetdb2, Contents of</td>
<td>222</td>
</tr>
<tr>
<td>Initialization</td>
<td>41, 51</td>
</tr>
<tr>
<td>Installation</td>
<td>38, 49</td>
</tr>
<tr>
<td>interfaces.conf Configuration File</td>
<td>237</td>
</tr>
<tr>
<td>Internet, Broadcasting on</td>
<td>208</td>
</tr>
<tr>
<td>IP</td>
<td>14</td>
</tr>
<tr>
<td>IP Datagrams</td>
<td>16</td>
</tr>
<tr>
<td>IP Driver and Descriptor</td>
<td>76</td>
</tr>
<tr>
<td>IP_SS_IOCTL</td>
<td>81</td>
</tr>
<tr>
<td>ipstart</td>
<td>160</td>
</tr>
<tr>
<td>ITEM Addressing</td>
<td>90, 93, 96, 99, 102</td>
</tr>
</tbody>
</table>

**LAN Communications Pak**

- Architecture 34, 35
- Configuration Client Utilities 120
- Descriptors and Drivers 31
- Examples and Utilities 27
- Libraries 30
Using LAN Communications Pak

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

---

Using LAN Communications Pak

Operating System, Target 150
Options 125
Options, Socket Level 213
OS 121

---

Parameters, Capture 179
Parameters, Set 181
Parameters, Toggle 183
Parameters, Mode 180
Parameters, Send 180
ping 172
Point to Point Protocol 45
PPP
  Data Flow 48
  Device Descriptors 54
  Drivers and Descriptors 46
  Modem Dialer Command Line Arguments 63
  Modem Dialer Utility (chat) 62
  Script Commands 58
  Transmission Process 51
PPP Daemon Command Line Arguments 57
PPP Daemon Process, Start 71
PPP Daemon Utility 57
pppauth Setting, Example 68
pppauth Utility 66
pppd Script Commands 58
Processor and Options, Target 151
Programming 189
Programming Overview 190
Programs, Utility 49
Programs, Utility 49
Protocol Drivers 47, 75
Protocol, Bootstrap 108
Protocol, Point to Point 45
Protocols 234
protocols 232, 234
protocols file 234

---

Using LAN Communications Pak

245
| A      | B      | C      | D      | E      | F      | G      | H      | I      | J      | K      | L      | M      | N      | O      | P      | Q      | R      | S      | T      | U      | V      | W      | X      | Y      | Z      |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
|        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        | example entry 234 | Protocols, File Transfer 137 |
| Raw Sockets 193 | Reading Data 42, 53 | Reading Data Using Sockets 204 | Reading Functions 204 | Receiving Process 209 | resolv.conf 232, 236 | resolv.conf Configuration File 236 | Return Codes from API Calls, Application 78, 91, 94, 97, 100 | routes 173 | routed 175 | routes.conf Configuration File 238 | RPC Configuration File 238 | Run Chat Script 72 |
| Script Commands, pppd 58 | Sections 123 | Send Parameters 180 | Sending a Host Name 116 | Sending Data 41, 53 | Serial Line Internet Protocol (SPSLIP) Driver 37 | Server Steps 194 | Server Utilities 109 | Server Utilities, BOOTP 109 | Server, BOOTP 107 | Server, Client and service defined 234 | Services 234 | services 232, 234 | services file 234 | example entry 235 | Setstats for SPENET 104 |
Sharing Common Values Between Tags 116
Socket Level Options 213
Socket Operations, Controlling 213
Socket Type/Protocol 192
Socket Types 191
Socket, Connect a 200
Socket, Establishing a 194
Sockets, Datagram 192, 200
Sockets, Raw 193
Sockets, Reading Data Using 204
Sockets, Setting up Non-Blocking 206
Sockets, Stream 192, 194
Sockets, Writing Data Using 205
Software Description 27
SPENET, Getstats for 103
SPENET, Setstats for 104
SPF
   Ethernet (spenet) Protocol Driver 101
   Ethernet Driver and Descriptor 101
   IP (spip) Protocol Driver 76
   RAW (spraw) Protocol Driver 89
   Routing Domain (sproute) Protocol Driver 92
   TCP (sptcp) Protocol Driver 95
   UDP (spudp) Protocol Driver 98
SPF_GS_ARPENT 104
SPF_GS_ARPTBL 104
SPF_GS_SYMBOLS 86
SPF_SS_ADDARP 105
SPF_SS_ATTIF 80
SPF_SS_DELADDR 85, 86
SPF_SS_DELARP 105
SPF_SS_IOCTL Commands 82
SPF_SS_IOCTL Commands, Other Supported 82
SPF_SS_SETADDR 85
SPIP, Drivers above 79
SPIP, Drivers below 79
SPIP, Getstats and Setstats Above 80
SPIP, Getstats and Setstats Below 85
SPRAW Driver and Descriptor 89
SPROUTE Driver and Descriptor 92
spslip Device Descriptor 43
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPSLIP Introduction</td>
<td>38</td>
</tr>
<tr>
<td>Spslip Transmission Process</td>
<td>41</td>
</tr>
<tr>
<td>Start PPP Daemon Process</td>
<td>71</td>
</tr>
<tr>
<td>Steps, Server</td>
<td>194</td>
</tr>
<tr>
<td>Stream Sockets</td>
<td>192, 194</td>
</tr>
<tr>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td>Ethernet Devices for 68000</td>
<td>223</td>
</tr>
<tr>
<td>Ethernet Devices for 80X86</td>
<td>224</td>
</tr>
<tr>
<td>Ethernet Devices for PPC</td>
<td>224</td>
</tr>
<tr>
<td>Syntax</td>
<td>124</td>
</tr>
<tr>
<td>Tags, Bootp</td>
<td>112</td>
</tr>
<tr>
<td>Tags, Sharing Common Values Between</td>
<td>116</td>
</tr>
<tr>
<td>Target Operating System</td>
<td>150</td>
</tr>
<tr>
<td>Target Processor and Options</td>
<td>151</td>
</tr>
<tr>
<td>TCP</td>
<td>15</td>
</tr>
<tr>
<td>TCP Driver and Descriptor</td>
<td>95</td>
</tr>
<tr>
<td>telnet</td>
<td>178</td>
</tr>
<tr>
<td>telnetd</td>
<td>184</td>
</tr>
<tr>
<td>tftpd</td>
<td>187</td>
</tr>
<tr>
<td>tftpdc</td>
<td>188</td>
</tr>
<tr>
<td>Toggle Parameters</td>
<td>183</td>
</tr>
<tr>
<td>Transmission Characteristics, Data Reception And</td>
<td>98</td>
</tr>
<tr>
<td>Transmission Characteristics, Data Reception and</td>
<td>76, 89, 92, 95, 101</td>
</tr>
<tr>
<td>Transmission Process, Spslip</td>
<td>41</td>
</tr>
<tr>
<td>Type/Protocol, Socket</td>
<td>192</td>
</tr>
<tr>
<td>Types, Socket</td>
<td>191</td>
</tr>
<tr>
<td>UDP</td>
<td>16</td>
</tr>
<tr>
<td>UDP Driver and Descriptor</td>
<td>98</td>
</tr>
<tr>
<td>Using Connect</td>
<td>199</td>
</tr>
<tr>
<td>Utilities</td>
<td>57, 119, 123</td>
</tr>
<tr>
<td>Utilities, BOOTP Server</td>
<td>109</td>
</tr>
<tr>
<td>Utilities, Server</td>
<td>109</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Utility Programs</td>
<td>49</td>
</tr>
<tr>
<td>Utility Programs</td>
<td>49</td>
</tr>
<tr>
<td>Utility, PPP Daemon</td>
<td>57</td>
</tr>
<tr>
<td>Utility, pppauth</td>
<td>66</td>
</tr>
<tr>
<td>Writing Data Using Sockets</td>
<td>205</td>
</tr>
<tr>
<td>Writing Functions</td>
<td>205</td>
</tr>
</tbody>
</table>
250 Using LAN Communications Pak
To: Microware Customer Support
FAX: 515-224-1352

From: ____________________________________________
Company: _______________________________________
Phone: __________________________________________
Fax: _______________ Email: _______________________

Product Name: LAN Communications Pak

Description of Problem:
________________________________________________
________________________________________________
________________________________________________
________________________________________________
________________________________________________
________________________________________________
________________________________________________
________________________________________________

Host Platform_____________________________________
Target Platform_____________________________________

MICROWARE™