LAN Communications Pak Programming Reference

Version 3.6

www.radisys.com

World Headquarters
5445 NE Dawson Creek Drive • Hillsboro, OR
97124 USA
Phone: 503-615-1100 • Fax: 503-615-1121
Toll-Free: 800-950-0044

International Headquarters
Gebouw Flevopoort • Televisieweg 1A
NL-1322 AC • Almere, The Netherlands
Phone: 31 36 5365595 • Fax: 31 36 5365620

RadiSys Microware Communications Software Division, Inc.
1500 N.W. 118th Street
Des Moines, Iowa 50325
515-223-8000

Revision A
November 2001

Copyright and publication information

This manual reflects version 3.6 of LAN Communications Pak.
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 RadiSys Microware Communications Software Division, Inc.

Disclaimer

The information contained herein is believed to be accurate as of the date of publication. However, RadiSys Corporation 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. RadiSys Corporation 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 RadiSys Corporation and/or other parties. Unauthorized distribution of software may cause damages far in excess of the value of the copies involved.

November 2001

Table of Contents

Chapter 1: Library Function Reference

This chapter provides you with details of the `netdb.l`, `ndblib.l`, `ppplib.l`, and `socket.l` libraries in the LAN Communications Pak. The following sections are included:

- **Internet Network Database Functions**
- **Point-to-Point Protocol Functions**
- **Structure Definitions**
- **Structure List**
- **Function List**

Internet Network Database Functions

The Internet library (`netdb.l`) provides functions for retrieving information from the Internet data files or embedded in the `inetdb` data module and for Internet address manipulation. Each data access function links to `inetdb` and returns a structure pointing to the appropriate entry from the data files or modules.

There are two methods of linking to `inetdb`:

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

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

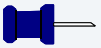
For example, the following program accesses the data module and prints the host entries:

```
listhosts()
{
    struct hostent *host
    sethostent();/* link to inetdb */
    while (host = gethostent()) {
        print_host_entry(host) }
    endhostent();/* unlink from inetdb */
}
```

LAN Communications Pak provides three variations of the Internet library:

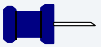
`netdb.l`	This library provides bindings that call the shared `netdb` trap handler for OS-9 systems and subroutines for the OS-9 module.
`netdb_dns.l`	Functions for DNS client support. These functions do not call the `netdb` trap handler.

`netdb_local.l`	Functions for local hostname resolution. These functions do not call the `netdb` trap handler.



Note

References to `netdb.l` also refer to `netdb_dns.l` and `netdb_local.l` in the function definitions.



Note

`netdb.l` links to the `netdb` trap/subroutine module and executes the code in that module.

For application development, you can link with `netdb_local.l` or `netdb_dns.l` and create applications which do not call the `netdb` trap handler. The application size will be larger but the inlined code will execute faster. Also, this may remove the need for loading the `netdb` module.

To link with the `netdb.l` library, a `netdb` trap/subroutine module must be loaded prior to running the application. There are two modules to choose from.

`netdb_dns`	The `netdb_dns` module provides Internet database functionality by using DNS client functionality to contact a name-server if a host name cannot be resolved or found locally within the inetdb data module.
`netdb_local`	The `netdb_local` module provides Internet database functionality by searching the local `inetdb` data module.

The different combinations of library and trap/subroutine modules are identified in the following table.

Table 1-1 Library and Trap/Subroutine Module Combinations

Components	Shared Code via trap module	DNS Client Support
`netdb.l + netdb_dns`	Yes	Yes
`netdb.l + netdb_local`	Yes	No
`netdb_dns.l`	No	Yes
`netdb_local.l`	No	No

netdb.l Function List

The following table lists and describes the functions that compose the `netdb.l` libraries.

Table 1-2 netdb.l Library Functions and Descriptions

Function	Description
`delhostbyname()`	Delete Host Entry by Name
`delintbyname()`	Delete Interface Entry by Name
`delresolvent()`	Delete DNS Resolver Entry
`delroutent()`	Delete Route Entry
`endhostent()`	Unlink from Network Database
`endintent()`	Unlink from Network Database

Table 1-2 netdb.l Library Functions and Descriptions (continued)

Function	Description
`endnetent()`	Unlink from Network Database
`endprotoent()`	Unlink from Network Database
`endresolvent()`	Unlink from Network Database
`endroutent()`	Unlink from Network Database
`endservent()`	Unlink from Network Database
`gethostbyaddr()`	Get Network Host Entry by Address
`gethostbyname()`	Get Network Host Entry by Name
`gethostent()`	Get Network Host Entry
`getinetdent()`	Get Inetd Entry
`getintent()`	Get Interface Entry
`getnetbyaddr()`	Get Network Entry by Address
`getnetbyname()`	Get Network Entry by Name
`getnetent()`	Get Network Entry
`getprotobyname()`	Get Protocol Entry
`getprotobynumber()`	Get Protocol Entry by Number
`getprotoent()`	Get Protocol Entry
`getresolvent()`	Returns Pointer to DNS Structure

Table 1-2 netdb.l Library Functions and Descriptions (continued)

Function	Description
`getservbyname()`	Get Service Entry by Name
`getservbyport()`	Get Service Entry by Port
`getservent()`	Get Service Entry
`htonl()`	Convert 32-Bit Values from Host to Network Byte Order
`htons()`	Convert 16-Bit Values from Host to Network Byte Order
`inet_addr()`	Convert Dot Notation into Network Address
`inet_aton()`	Convert Internet Address to Binary Address
`inet_lnaof()`	Get Local Address
`inet_makeaddr()`	Get Address from Network and Host Address
`inet_netof()`	Get Network Number
`inet_network()`	Interpret Network Number
`inet_ntoa()`	Return Address in Dot Notation
`ip_start()`	Initialize IP stack
`ntohl()`	Convert 32-Bit Values from Network to Host Byte Order

Table 1-2 netdb.l Library Functions and Descriptions (continued)

Function	Description
`ntohs()`	Convert 32-Bit Values from Network to Host Byte Order
`puthostent()`	Add Network Host Entry
`putintent()`	Add Interface Entry
`putnetent()`	Add a network entry to inetdb
`putprotoent()`	Add protocol entry to inetdb
`putresolvent()`	Set the DNS Entry
`putroutent()`	Add Route Entry
`putservent()`	Add interface entry to inetdb
`res_cancel()`	Cancel DNS Client Request
`sethostent()`	Set Host Entry
`setnetent()`	Set Network Entry
`setprotoent()`	Set Protocol Entry
`setservent()`	Set Services Entry

socket.l Function List

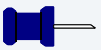
SPF includes an implementation of the Berkeley socket API in the `socket.l` library. The Internet socket library provides a BSD 4.4-like socket API. The following table lists and describes the functions that compose the `socket.l` library:

Table 1-3 socket.l Functions and Descriptions

Function	Description
`accept()`	Accept Connection on Socket
`bind()`	Binds Name to Socket
`connect()`	Initiates Connection on Socket
`gethostname()`	Gets Name of Current Host
`getpeername()`	Get Network Entry
`getsockname()`	Gets Socket Name
`getsockopt()`	Get Socket Options
`listen()`	Listen for Connections on Socket
`recv()`	Receives Message From Connected Socket
`recvfrom()`	Receives Message from Socket
`send()`	Sends Message to Connected Socket
`sendto()`	Sends Message to Socket
`sethostname()`	Set Name of Current Host

Table 1-3 socket.l Functions and Descriptions (continued)

Function	Description
`setsockopt()`	Set Options on Sockets
`shutdown()`	Shut Down Part of Full-Duplex Connection
`socket()`	Creates Endpoint for Communication



Note

When creating socket applications, the `ndblib.l` and `item.l` libraries must also be linked.

Other OS-9 Functions

The following generic OS-9 system calls are also supported by sockets.

```
_os_close()
_os_gs_popt()
_os_gs_ready()
_os_read()
_os_ss_popt()
_os_ss_relea()
_os_ss_sendsig()
_os_write()
ioctl()
select()
```

For More Information

Please refer to the ***Ultra C Library Reference*** for the syntax and functionality of these calls.

ndblib.l Function List

The `ndblib.l` library can be used to dynamically create an `inetdb` data module. This enables an application to configure a system before the IP stack is initialized.

The `netdb.l` library may call functions defined in `ndblib.l`. Therefore, when linking with `netdb.l`, `ndblib.l` must also be linked. The following table lists and describes the functions that compose the `ndblib.l` library.

Table 1-4 netdb.l Functions and Descriptions

Function	Description
`ndb_create_ndbmod()`	Create Network Database Module

For More Information

See **Appendix B: Dynamic Configuration of the inetdb Data Module** for more information about dynamic configuration.

Prototypes

Prototypes for these calls are declared in:

```
MWOS/SRC/DEFS/SPF/BSD/sys/socket.h.
MWOS/SRC/DEFS/SPF/BSD/netdb.h.
MWOS/SRC/DEFS/SPF/BSD/netinet/in.h
```

The `sockaddr` structure is defined in `sys/socket.h` as:

```
struct sockaddr
{
   u_char sa_len;    /* total length */
   u_char sa_family; /* address family */
   char sa_data [14];/* up to 14 bytes of direct address */
};
```

This is a generic socket address meant to accompany various types of layer-three protocols. When using the IP protocol, only addresses belonging to the INET protocol family are allowed.

The `AF_INET` type address is declared in

`MWOS/SRC/DEFS/SPF/BSD/netinet/in.h`

as follows:

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

The `in_addr` structure is defined in the same file as:

```
struct in_addr
{
    u_long           s_addr;
};
```

Point-to-Point Protocol Functions

The Point-to-Point Protocol (PPP) Application Programming Interface (API) provides four types of function calls: stack configuration, CHAT scripting, authentication database, and connection/disconnect. In addition, this API defines structures that provide error reporting and other functionalities between the PPP stack and the software using the API.

Stack Configuration

Stack configuration consists of deciding which, if any, default options within the stack need to be modified, and then modifying those values. The calls that provide this functionality include those listed below:

`ppp_get_options()`	Get negotiable stack options.
`ppp_set_options()`	Set negotiable stack options.
`ppp_option_block()`	Communicate the current, desired values between the stack and the API.

The API also provides functions to get and set the asynchronous parameters of the PPP link. These parameters include, among others, baud rate, parity, and word size. The calls that provide this functionality are listed below:

`ppp_get_asynch_params()`	Get asynchronous parameters of the PPP link.
`ppp_set_asynch_params()`	Set aynchronous parameters of the PPP link.
`ppp_modem_p()`	Communicate the current, desired values between the stack and the API.

CHAT Scripting

CHAT scripting is the process of setting up the data link between the PPP client and server. This may involve sending or receiving commands, logging onto a UNIX shell account and running a PPP startup command, or performing messaging before the client and server exchange PPP configuration packets. The calls that provide this functionality are listed below:

`ppp_chat_open()`	Open a raw CHAT path.
`ppp_chat_close()`	Close CHAT path.
`ppp_chat_write()`	Write data to CHAT path.
`ppp_chat_read()`	Read data from CHAT path.
`ppp_chat_script()`	Run a CHAT script.
`ppp_connect()`	Run optional CHAT script and establish a PPP link.

The `ppp_chat_open()`, `ppp_chat_close()`, `ppp_chat_write()`, and `ppp_chat_read()` functions are low-level functions provided so that you can implement your own version of a CHAT scripting engine. In most cases, the API's built-in engine is adequate; thus, the `ppp_chat_script()` or `ppp_connect()` function is used.

For More Information

For more information on specific CHAT scripting commands, see **Chapter 4** of the ***Using LAN Communications Pak*** manual included with this CD.

Authentication Database

The authentication database is a memory module referenced by the PPP stack during the authentication phase of a PPP connection. If no authentication is needed, this database may be nonexistent. The currently supported authentication protocols are PAP (Password Authentication Protocol) and CHAP (Challenge-Handshake Authentication Protocol). Using the authentication calls listed below, the administrative program may store the PAP/CHAP information needed to log onto any number of servers.

`ppp_auth_create_mod()`	Create a new authentication module (database).
`ppp_auth_link_mod()`	Link to an existing authentication module.
`ppp_auth_unlink_mod()`	Unlink from an authentication module.
`ppp_auth_get_cur_chap()`	Get CHAP name/secret for currently set peer.
`ppp_auth_get_cur_pap()`	Get PAP name/secret for currently set peer.
`ppp_auth_get_peer_name()`	Get name of currently set peer.
`ppp_auth_set_peer_name()`	Set current peer.
`ppp_auth_add_chap()`	Add a new CHAP entry.
`ppp_auth_add_pap()`	Add a new PAP entry.
`ppp_auth_del_chap()`	Delete an existing CHAP entry.
`ppp_auth_del_pap()`	Delete an existing PAP entry.

Connect/Disconnect

Connect/disconnect calls cause the PPP stack to begin the negotiation and establishment of a PPP link, or the termination of an existing PPP link. The calls that provide this functionality are listed below:

`ppp_connect()`	Run optional CHAT script & establish PPP link.
`ppp_start()`	Establish a PPP link.
`ppp_disconnect()`	Terminate current PPP link.

In addition, there are other miscellaneous functions provided by this API. (Some of these functions are required and some are optional, as indicated.) The calls are listed below:

`ppp_init()`	Initialize the PPP API. (required)
`ppp_term()`	Terminate the PPP API. (required)
`ppp_open()`	Open the PPP stack. (required)
`ppp_close()`	Close the PPP stack. (required)
`ppp_get_params()`	Obtain negotiated stack parameters. (optional)
`ppp_get_statistics()`	Obtain current stack statistics. (optional)
`ppp_reset_statistics()`	Reset stack statistics. (optional)

ppplib.l Function List

The PPP API functions are listed in the `ppplib.l` library. The following table lists and describes the API functions that compose the `ppplib.l` file:

Table 1-5 ppplib.l Library Functions and Descriptions

Function	Description
ppp_init()	Initialize the PPP API.
`ppp_term()`	`Terminate the PPP API`.
`ppp_open()`	`Open the PPP Stack`.
`ppp_close()`	Close the PPP Stack.
`ppp_get_asynch_params()`	Get Asynchronous Link Parameters.
`ppp_get_params()`	Obtain Negotiated Stack Parameters.
`ppp_get_options()`	Get Negotiable Stack Options.
`ppp_set_asynch_params()`	Set Asynchronous Link Parameters.
`ppp_set_options()`	Set Negotiable Stack Options.
`ppp_connect()`	Run Optional CHAT Script and Establish PPP Link.
`ppp_disconnect()`	Terminate Current PPP Link.
`ppp_get_statistics()`	Obtain Current Stack Statistics.

Table 1-5 ppplib.l Library Functions and Descriptions (continued)

Function	Description
`ppp_reset_statistics()`	Reset Stack Statistics.
`ppp_auth_create_mod()`	Create a New Authentication Module (Database).
`ppp_auth_link_mod()`	Link to Existing Authentication Module.
`ppp_auth_unlink_mod()`	Unlink from an Authentication Module.
`ppp_auth_get_cur_chap()`	Get CHAP Name/Secret for Currently Set Peer.
`ppp_auth_get_cur_pap()`	Get PAP Name/Secret for Currently Set Peer.
`ppp_auth_get_peer_name()`	Get Name of Currently Set Peer.
`ppp_auth_set_peer_name()`	Set Current Peer.
`ppp_auth_add_chap()`	`Add CHAP Entry to Authentication Module`.
`ppp_auth_add_pap()`	Add PAP Entry to Authentication Module.
`ppp_auth_del_chap()`	Delete CHAP Entry from Authentication Module.
`ppp_auth_del_pap()`	Delete PAP Entry from Authentication Module.
`ppp_chat_open()`	Open a Raw CHAT Path.

Table 1-5 ppplib.l Library Functions and Descriptions (continued)

Function	Description
`ppp_chat_close()`	Close CHAT Path.
`ppp_chat_write()`	Write Data to CHAT Path.
`ppp_chat_read()`	Read Data from CHAT Path.
`ppp_chat_script()`	Run a CHAT Script.
`ppp_start()`	Establish a PPP Link.

Structure Definitions

This section includes the structure definitions in `netdb.h`, `resolve.h`, and `ppplib.h`. Each structure is described in the tables below.

netdb.h and resolve.h

`netdb.h` and `resolve.h` are used for Internet data module activities. The structures are listed and described in the following table.

Table 1-6 netdb.h and resolve.h Structures and Descriptions

Structure	Description
`hostent`	Retrieve/Add/Remove Host Entry.
`n_ifaliasreq`	Add/Remove Interface Address Structure.
`n_ifnet`	Add Interface Structure.
`netent`	Retrieve/Add/Remove Network Entry.
`protoent`	Retrieve/Add/Remove Protocol Entry.
`resolvent`	Retrieve/Add/Remove DNS Client Entry.
`rtreq`	Add/Remove Route Structure.
`servent`	Retrieve/Add/Remove Services Entry.

ppplib.h

There are five important structures that allow interaction between the calling application and the PPP stack. These structures are defined in the `ppplib.h` header file and are also listed below. In all structures (except `auth_handle`), there are fields reserved for future use. It is important for the application to “zero” out all of these fields to ensure compatibility with future versions of drivers and this API. There are two easy ways to accomplish this: set the structure equal to zero or use the `memset()` function.

```
ppp_option_block       pppopts = {0};
```

or

```
ppp_option_block       pppopts;
memset(&pppopts, 0, sizeof(pppopts));
```

Table 1-7 ppplib.h Structures and Descriptions

Structure	Description
`auth_handle`	Allocated by Authentication Database Functions.
`ppp_hdlc_stats`	Allow HDLC Driver to Return Current Receive and Transmit Statics.
`ppp_option_block`	Obtain or Change Desired PPP Negotiation Options.
`ppp_param_block`	Obtain Stack Parameters.
`ppp_conninfo`	Record PPP Connection Error Information.

Table 1-7 ppplib.h Structures and Descriptions (continued)

Structure	Description
`ppp_error`	Store Error Information.
`ppp_modem_p`	Get and Set Asynchronous Parameter or PPP Link.

Structure List

This section includes the structure definitions for the `netdb.h`, `resolv.h`, and `ppplib.h` header files, in alphabetical order. Each function definition includes the following subsections:

- Declaration

  The **Declaration** information details how the structure is defined in the header file.

- Description

  The **Description** area defines the declaration and location of the structure.

- Fields

  The **Fields** section defines the fields for a given structure.

auth_handle

Allocated by Authentication Database Functions

Declaration

```
typedef struct _auth_handle

{

   mh_data              *mod_hdr;

   Auth_data            data;

}

auth_handle,*Auth_handle;
```

Description

The `auth_handle` structure is used by the Authentication database functions. Applications do not have to interpret the internal portions of this structure (`mod_hdr`, `data`). However, they need to allocate `auth_handle` and pass a pointer to the Authentication functions that fill out and manage the structure.

hostent

Retrieve/Add/Remove Host Entry

Declaration

The `hostent` structure is declared in the file `netdb.h` as follows:

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

Description

Returns and sets host entries in the Internet data module. This structure is used by the functions:

```
gethostbyaddr()
gethostbyname()
puthostent()
gethostent()
```

Fields

`h_name`	A pointer to the official name of the host.
`h_aliases`	A pointer to a null-terminated array of pointers to alternate names for the host.
`h_addrtype`	The type of address (`AF_INET` only) returned.
`h_length`	The address length in bytes.
`h_addr`	A pointer to the network address for the host. Host addresses are returned in network-byte order.
`h_addr_list`	A list of addresses from name server.

n_ifaliasreq

Add/Remove Interface Address Structure

Declaration

The `n_ifaliasreq` is defined in `netdb.h` as follows:

```
struct n_ifaliasreq
{
   char                 ifra_name[16];
   struct sockaddr      ifra_addr;
   struct sockaddr      ifra_broadaddr;
  #define ifra_dstaddr ifra_broadaddr
   struct sockaddr      ifra_mask;
} n_ifaliasreq;
```

Description

`n_ifaliasreq` is the interface structure for adding an interface. This structure is used in the function `putintent()`.

Fields

`ifra_name`	Interface name
`ifra_addr`	IP address
`ifra_broadaddr`	Broadcast address or destination address
`ifra_mask`	subnet mask

n_ifnet

Add Interface Structure

Declaration

The `n_ifnet` structure is defined in `netdb.h` as follows:

```
struct n_ifnet {
     char         if_name[16];
     char         if_stack_name[30];
     short        if_flags;
     struct n_if_data {
                  u_long    ifi_type;
                  u_long    ifi_addrlen;
                  u_long    ifi_mtu;
                  u_long    ifi_metric;
     } if_data;
     u_long       extra[6];
} n_ifnet;
```

Description

The `n_ifnet` structure adds interface structures to the Internet data module. See `putintent()`.

Fields

`if_name`	Interface name
`if_stack_name`	Binding Device path.
`if_flags`	Interface state: See `MWOS/SRC/DEFS/SPF/BSD/net/if.h` for possible values.
`ifi_type`	Type of interface (not used, set to zero)
`ifi_addrlen`	Hardware address length (not used, set to zero).
`ifi_mtu`	Maximum transfer unit. Largest piece of data handled by interface.
`ifi_metric`	Weighted routing metric. Usually zero.
`extra`	For future use. Set to zero.

netent

Retrieve/Add/Remove Network Entry

Declaration

The `netent` structure is declared in the file `netdb.h` as follows:

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

Description

The `netent` structure obtains network entries in the Internet date module.

See:

`getnetbyaddr()`

`getnetbyname()`

`getnetent()`

Fields

`n_name`	A pointer to the official name of the network.
`n_aliases`	A pointer to a null-terminated list of pointers to alternate names for the network.
`n_addrtype`	The type of the network number (`AF_INET` only) returned.
`n_net`	The network number. Network numbers are returned in host-byte order.

ppp_conninfo

Record PPP Connection Error Information

Declaration

```
struct _ppp_conninfo{
        signal_code     sig_lcp_up;
        signal_code     sig_lcp_down;
        signal_code     sig_ipcp_up;
        signal_code     sig_ipcp_down;

        u_int32         flags;
        error_code      last_err;
        u_int32         max_errors;
        ppp_error       error_array;

        signal_code     sig_lcp_finish;
        signal_code     sig_ipcp_finish;
        u_int32         rsvd[4];
}
Ppp_conninfo, *Ppp_conninfo;
```

Description

The `ppp_conninfo` structure is allocated by the calling application and must be retained for the life of a PPP link. The `ppp_conninfo` area is where the PPP stack records error information regarding the PPP connection. Detailed error information is recorded in the `error_array`, which is a separately allocated array of `ppp_error` structures.

The following source code illustrates how to allocate a `ppp_conninfo` structure and details an error array that can hold `MAX_ERRORS` entries (error detection not shown):

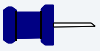
```
ppp_conninfo          ci = {0};
u_int32               size;
ci.max_errors = MAX_ERRORS;
size = sizeof(ppp_error) * MAX_ERRORS;
ci.error_array = malloc(size);
memset(ci.error_array, 0, size);
```

ppp_error

Store Error Information

Declaration

```
typedef struct _ppp_error {
   u_int32              layer;
      union {
         struct {
         error_code err;
         u_int32         line;
         u_int32         abort_line;
         } chat;

         struct {
         u_int32         option;
         u_int32         my_request;
         u_int32         his_request;
         } lncp;
      } err_info;
} ppp_error, *Ppp_error;
```



Note

Layer values may be PPP_LAYER_IPCP, PPP_LAYER_LCP, PPP_LAYER_HDLC, or PPP_LAYER_CHAT.

Description

When a significant error occurs within the PPP stack and an application has provided an error reporting array, information about the error will be stored in the first empty `ppp_error` slot. An empty slot is defined by the “layer” field having a value of `0`.

The manner in which the information within the `ppp_error` structure is decoded depends on which layer recorded the error. For CHAT errors, the OS-9 error value is saved along with the offending CHAT line number. If an `abort` string has caused the CHAT script to terminate, the line number within the script, in which the abort string was defined, is also recorded. For LCP and IPCP layers, the desired option number, local request, and remote request are all recorded.

ppp_hdlc_stats

Allow HDLC Driver to Return Current Receive and Transmit Statics

Declaration

```
typedef struct _ppp_hdlc_stats{
        u_int32         rx_bytes;
        u_int32         rx_frames;
        u_int32         rx_frames_compressed;
        u_int32         rx_frames_dropped;
        u_int32         rx_frames_overrun;
        u_int32         rx_errors;
        u_int32         rx_fcs_errors;

        /*Transmit statistics */
        u_int32         tx_bytes;
        u_int32         tx_frames;
        u_int32         tx_frames_compressed;
        u_int32         tx_frames_dropped;
        u_int32         tx_frames_overrun;
        u_int32         tx_errors;
        u_int32         rsvd[3];
} ppp_hdlc_stats, *Ppp_hdlc_stats;
```

Description

The `ppp_hdlc_stats` structure is allocated by the calling application and referred to in the `ppp_get_statistics()` function. This structure allows the HDLC driver to return the current receive and transmit statics to the calling application.

For both the receive and transmit, the total number of bytes (bytes on the wire, including all HDLC framing bytes), errors (includes internal OS-9/SPF errors), and frames, including the Van-Jacobson compressed frames, dropped frames, and overrun frames (frames exceeding MTU/MRU) are part of this structure. For receive, there is also a field that tabulates the total number of frames having an invalid FCS (Frame Check Sequence); this is similar to a CRC check.

ppp_modem_p

Get and Set Asynchronous Parameter or PPP Link

Declaration

```
typedef struct ppp_modem_p{
        char        rx_dev_name[MAX_NAME_LEN];
        char        tx_dev_name[MAX_NAME_LEN];
        u_int8      baud_rate;
        u_int8      parity;
        u_int8      word_size;
        u_int8      stop_bits;
        u_int8      rts_enable;
        u_int32     rx_bufsize;
        u_int32     tx_bufsize;
} ppp_modem_p, *Ppp_modem_p;
```

Description

This structure is defined in ppp.h. It is used by the PPP function to get and set the asynchronous parameter or a PPP link, such as baud rate.

ppp_option_block

Obtain or Change Desired PPP Negotiation Options

Declaration

```
typedef struct _ppp_option_block{
        u_int32          ppp_mode;
        u_int32          ipcp_timeout;
        u_int32          ipcp_max_configure;
        u_int32          ipcp_max_terminate;
        u_int32          ipcp_max_failure;
        struct           pppopt_ui32ipcp_accept_local;
        struct           pppopt_ui32ipcp_accept_remote;
        struct           pppopt_ui32rx_ip_cproto;
        struct           pppopt_ui32tx_ip_cproto;
        struct           pppopt_ui32rx_ipcp_cslot;
        struct           pppopt_ui32rx_ipcp_mslot;
        u_int32          lcp_timeout;
        u_int32          lcp_max_configure;
        u_int32          lcp_max_terminate;
        u_int32          lcp_max_failure;
        struct           pppopt_ui32rx_accm;
        struct           pppopt_ui32_8 tx_accm;
        struct           pppopt_ui32rx_acfc;
        struct           pppopt_ui32rx_pfc;
        struct           pppopt_ui32rx_mru;
        struct           pppopt_ui32tx_mru;
        struct           pppopt_ui32auth_challenge;
        u_int32          ipcp_default_route;
        u_int32          rsvd[5];
} ppp_option_block, *Ppp_option_block;
```

Description

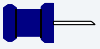
This structure is used to obtain or change the desired PPP negotiation options via the `ppp_get_options()` and `ppp_set_options()` functions, respectively. Many of these options are specified using a `pppopt_ui32` structure, which looks similar to the following code:

```
struct pppopt_ui32

{

   u_int32        priority;/* DEFAULT, DESIRED, or REQUIRED */

   u_int32        value;

};
```

The `tx_accm` must have a size of 256-bits; therefore, it uses the following structure:

```
struct pppopt_ui32_8

{

u_int32           priority;

u_int32           value[8];

};
```

These structures do not allow individual PPP options to be negotiated (DEFAULT), requested (DESIRED), or forced (REQUIRED), which causes a failure if the option cannot be negotiated. Priority levels are currently not supported, and all stack options are treated as DESIRED. The “value” field depends on the option specified.



Note

Most of these options are specified within the LCP and IPCP descriptors. The descriptor values will be used by default unless altered by a `ppp_set_options()` call.

ppp_param_block

Obtain Stack Parameters

Declaration

```
typedef struct _ppp_param_block
{
        /*Generic stack parameters*/
        u_int32          ppp_mode;

        /*IPCP-specific stack parameters*/
        u_int32          rx_ip_cproto;
        u_int32          tx_ip_cproto;
        u_int32          rx_ipcp_cslot;
        u_int32          tx_ipcp_cslot;
        u_int32          rx_ipcp_mslot;
        u_int32          tx_ipcp_mslot;

        /*LCP-specific stack parameters*/
        u_int32          rx_accm;
        u_int32          tx_accm[8];
        u_int32          rx_acfc;
        u_int32          tx_acfc;
        u_int32          rx_pfc;
        u_int32          tx_pfc;
        u_int32          rx_mru;
        u_int32          tx_mru;
        u_int32          local_magic;
        u_int32          remote_magic;
```

```
        /*I/O enabled flags*/
        u_char              hdlc_io_enabled;
        u_char              lcp_io_enabled;
        u_char              ipcp_io_enabled;

        u_char              rsvd1;
        u_int32             rsvd2[3];
} ppp_param_block, *Ppp_param_block;
```

Description

The `ppp_param_block` structure is allocated by the calling application and used to obtain the stack parameters after LCP and IPCP negotiation has completed (both LCP and IPCP are I/O-enabled). Both transmit and receive parameters are returned at the same time. See the parameter descriptions for the `ppp_option_block` for more information on what each parameter represents. The `local_magic` and `remote_magic` parameters are the negotiated "magic numbers" for the PPP link.

For More Information

For more information on these parameters, refer to the ***Request for Comment*** (RFC) 1661.

protoent

Retrieve/Add/Remove Protocol Entry

Declaration

The `protoent` structure is declared in the file `netdb.h` as follows:

```
struct protoent {
     char         *p_name;
     char         **p_aliases;
     int          p_proto;
};
```

Description

The `protoent` structure obtains protocol information in the Internet data module.

See:

`getprotobyname()`
`getprotobynumber()`
`getprotoent()`

Fields

`p_name`	A pointer to the official name of the protocol.
`p_aliases`	A pointer to a null-terminated list of pointers to alternate names for the protocol.
`p_proto`	The protocol number in host-byte order.

resolvent

Retrieve/Add/Remove DNS Client Entry

Declaration

The `resolvent` structure is declared in the file `resolv.h` as follows:

```
struct   resolvent {
     char         *domain;
     char         *nameservers [MAXNS+1];
     char         *search [MAXDNSRCH+1]
}
```

Description

The `resolvent` structure is used for updating and obtaining DNS client resolving information in the Internet data module:

See:

`getresolvent()`
`putresolvent()`

Fields

`domain`	A pointer to the local domain name.
`nameservers`	Ordered list of `nameservers` (in dot notation).
`search`	Search list for host-name lookup.

rtreq

Add/Remove Route Structure

Declaration

The `rtreq` structure is declared in `netdb.h` as follows:

```
struct rtreq {
     int          req;
     int          flags;
     struct sockaddr dst;
     struct sockaddr gateway;
     struct sockaddr netmask;
};
```

Description

The `rtreq` structure updates and obtains routing information in the Internet data module. See:

`putroutent()`

`delroutent()`

Fields

`req`	Request type (`RTM_ADD`).
`flags`	Type of route. `RTF_HOST` for a host route.
`dst`	Destination address.
`gateway`	Gateway address.
`netmask`	Network mask address.

For More Information

See the file `MWOS/SRC/DEFS/SPF/BSD/net/route.h` for additional `req` and `flags` settings.

servent

Retrieve/Add/Remove Services Entry

Declaration

The `servent` structure is declared in the file `netdb.h` as follows:

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

Description

The `servent` structure obtains service information from the Internet data module. See:

`getservbyport()`

`getservbyname()`

`getservent()`

Fields

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

Function List

This section includes library function definitions for `netdb.l`, `ppplib.l`, and `socket.l` in alphabetical order. Each function definition includes the following subsections:

- Syntax

  The syntax information shows how the function and parameters look when written as a C function definition, even if the actual function is a macro or is written in assembly language.

  For example, the syntax for `socket()` appears as follows:

  ```
  #include <sys/types.h>
  #include <sys/socket.h>
  int socket(int af, int type, int protocol)
  ```

  This indicates `socket()` requires the `<sys/types.h>` and `<sys/socket.h>` header files, returns an integer, and requires three integer parameters. The parameter names are suggestions only; you can use any name.

- Libraries

  The **Library** field identifies the library or library group in which the function is included. Functions included in the various libraries are identified accordingly.

- Description

  The description section defines the syntax and processing of the function.

- Attributes

  The **Attributes** section lists various attributes of each function in relation to OS-9—including whether the function is compatible with OS-9 and/or OS-9 for 68K; whether the function is in user state and/or system state; and whether the function is safe for use in a threaded application.

- Errors

  When an error occurs, C functions may return an error code in the global variable `errno`. You must include the file `<errno.h>` in C programs to declare `errno`.

  The file `<sys/errno.h>` contains several UNIX errors that have been defined for OS-9. These are provided for source code compatibility with existing UNIX network code. For example, a non blocking `connect()` may check for `EINVAL`. On OS-9 this error is `EOS_ILLARG`, however if `<sys/errno.h>` is included, the check for `EINVAL` does not need to change.

- See Also—References to Other Calls

  Some functions have a **See Also** section. Functions listed in the **See Also** section are related functions or are called by the function being described.

accept()

Accept Connection on Socket

Syntax

```
#include <sys/socket.h>
int accept(
     int                s,
     struct sockaddr    *addr,
     int                *addrlen)
```

Libraries

`socket.l`
`item.l` (for 68K)

Description

`accept()` takes the first connection on the queue of pending connections and creates a new socket with the same properties as socket `s`. It allocates and returns a new socket descriptor. This new socket reads and writes data to and from the socket to which it is connected. It does not accept more connections. The original socket, `s`, remains open for accepting further connections.

If pending connections are nonexistent on the queue and the socket is configured for blocking, `accept()` blocks the caller until a connection is present.

If the socket is configured as non-blocking and pending connections are nonexistent on the queue, `accept()` returns `EWOULDBLOCK`.

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

`accept()` returns `-1` on error and `errno` is set to the error value. If successful, it returns a non-negative integer path number.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`s`	Is the original socket path. It is created by `socket()`, bound to an address with `bind()`, and is listening for connections after a `listen()`.
`addr`	Is a pointer returning the address of the peer as known to the communications layer. `addr` is returned in `AF_INET` format.
`addrlen`	Is a pointer to a value-result parameter used to pass the amount of space pointed to by `addr`. It returns the actual length in bytes of the address returned in `AF_INET` format.

Errors

`EINVAL`	The socket must be listening to call `accept()`.
`EOPNOTSUPP`	The referenced socket type or option is not supported.
`EWOULDBLOCK`	The socket is non-blocking and no connections are waiting to be accepted.

See Also

`bind()`
`connect()`
`listen()`
`socket()`

bind()

Binds Name to Socket

Syntax

```
#include <sys/socket.h>
int bind(
     int                s,
     struct sockaddr    *name,
     int                namelen)
```

Libraries

```
socket.l
```

Description

`bind()` assigns a name to an unnamed socket. When `socket()` creates a socket, it exists in a name space (address family) but has no assigned name. `bind()` requests the name pointed to by the parameter `name` be assigned to the socket. Only names belonging to the `AF_INET` family are supported.

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

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`s`	Specifies the path number of the socket.
`name`	Points to the socket address.
`namelen`	Specifies the length of the assigned name.

Errors

EADDRINUSE	The specified address name is already in use.
EADDRNOTAVAIL	The specified address name is not available on the local machine.
EINVAL	The socket is already bound to an address name.
EOS_PERMIT	The user is asking for a reserved port and the user is not super user.

See Also

connect()
getnetbyaddr()
socket()

connect()

Initiates Connection on Socket

Syntax

```
#include <sys/socket.h>
int connect(
     int                s,
     struct sockaddr    *name,
     int                namelen)
```

Libraries

```
socket.l
```

Description

If `s` is socket of type `SOCK_STREAM` (TCP), `connect()` attempts to connect to a listening socket. If the socket is a datagram socket such as `SOCK_DGRAM` (UDP) or `SOCK_RAW` (RAW), `connect()` stores the destination address locally. A successful connection returns 0. Otherwise, it returns -1 with the appropriate error code in `errno`.

If `s` is a non-blocking socket the initial `connect()` call returns `EINPROGRESS`. Subsequent calls return `EALREADY` until the connection is established, at which point an `EISCONN` error is returned.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

s	Specifies the path number of the socket to connect. If s is of type SOCK_STREAM, connect() attempts to connect to another socket.
name	Points to the other socket.
namelen	Specifies the length of the assigned name.

Errors

EADDRINUSE	The address is already in use.
EADDRNOTAVAIL	The specified address is not available from this machine.
EAFNOSUPPORT	Address in the specified address family cannot be used with this socket.
EALREADY	The socket is non-blocking and a previous connection attempt has not been completed.
ECONNREFUSED	The attempt to connect was forcefully rejected.
EINPROGROSS	A non-blocking socket connection cannot be completed immediately.
EISCONN	The socket is already connected.
EHOSTONREACH	No route to the host exists
ETIMEDOUT	An attempt to establish a connection timed out without establishing the connection.

See Also

accept()
getnetbyaddr()
socket()

delhostbyname()

Delete Host Entry by Name

Syntax

```
#include <netdb.h>
error_code
delhostbyname(
        char      *name)
```

Libraries

`netdb.l`

Description

`delhostbyname()` removes the host pointed to by `name` from the host section of the applicable `inetdb` data module. If successful, `delhostbyname()` returns 0; otherwise it returns an error.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Errors

`ENOMEM`	Insufficient RAM (POSIX).
`EOS_PARAM`	Bad parameter.
`EOS_PERMIT`	No permission.
`EOS_PNNF`	Attempted to delete an entry that does not exist.
`EOS_TRAP`	`netdb` module not found.

delintbyname()

Delete Interface Entry by Name

Syntax

```
#include <netdb.h>
error_code
delintbyname(
        char      *name)
```

Libraries

`netdb.l`

Description

`delintbyname()` removes the interface pointed to by `name` from the interface section of the `inetdb` data module. If successful, `delintbyname()` returns 0; otherwise it returns an error.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Errors

`EOS_PARAM`	Bad parameter.
`EOS_PERMIT`	No permission.
`EOS_PNNF`	Attempted to delete an entry that does not exist.
`EOS_TRAP`	`netdb` module not found.

delresolvent()

Delete DNS Resolver Entry

Syntax

```
#include <netdb.h>
error_code
delresolvent(void)
```

Libraries

`netdb.l`

Description

`delresolvent()` removes the current DNS resolver entry from the appropriate `inetdb` module. If successful, `delresolvent()` returns `0`; otherwise it returns an error.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Errors

`EOS_MNF`	`netdb_dns` module not found. Also returns this error when `netdb_local` module is loaded.
`EOS_PARAM`	Bad parameter; or not linked to an `inetdb` module.
`EOS_PNNF`	Attempted to delete an entry that does not exist.
`EOS_TRAP`	`netdb` module not found.

See Also

endresolvent()
getresolvent()
putresolvent()

delroutent()

Delete Route Entry

Syntax

```
#include <netdb.h>
error_code
delroutent(
     const struct rtreq      *route_ptr)
```

Libraries

`netdb.l`

Description

`delroutent()` removes the route entry matching `route_ptr` from the route section of the `inetdb` data module. Both the destination and gateway must match. If successful, `delroutent()` returns 0; otherwise it returns an error.

The `rtreq` structure (defined in `netdb.h`) is described in the **Structures** section.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Errors

`EOS_PARAM`	Bad parameter.
`EOS_PERMIT`	No write access.
`EOS_PNNF`	Attempted to delete an entry that does not exist.
`EOS_TRAP`	`netdb` module not found.

See Also

`putroutent()`

endhostent()

Unlink from Network Database

Syntax

```
#include <netdb.h>
int endhostent(void)
```

Libraries

`netdb.l`

Description

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

`endhostent()` returns `0` if successful; otherwise it returns an error.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Errors

`ENOMEM`	Insufficient RAM (POSIX).
`EOS_PARAM`	Bad parameter.

See Also

`gethostbyaddr()`
`gethostbyname()`
`gethostent()`
`sethostname()`

endintent()

Unlink from Network Database

Syntax

```
#include <netdb.h>
error_code
endintent(void)
```

Libraries

`netdb.l`

Description

`endintent()` indicates the process is finished using the interface section of the `inetdb` data module. The link count of `inetdb` is decremented. If successful, `endintent()` returns 0; otherwise it returns an error.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Errors

`ENOMEM`	Insufficient RAM (POSIX).
`EOS_PARAM`	Bad parameter.

endnetent()

Unlink from Network Database

Syntax

```
#include <netdb.h>
int endnetent(void)
```

Libraries

`netdb.l`

Description

`endnetent()` indicates the process is finished using the network section of the `inetdb` data module. The link count of `inetdb` is decremented. `endnetent()` returns `0` if successful; otherwise it returns an error.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Errors

`ENOMEM`	Insufficient RAM (POSIX).
`EOS_PARAM`	Bad parameter.

See Also

`getnetbyaddr()`
`getnetbyname()`
`getnetent()`

endprotoent()

Unlink from Network Database

Syntax

```
#include <netdb.h>
int endprotoent(void)
```

Libraries

`netdb.l`

Description

`endprotoent()` indicates the process is finished using the protocol section of the `inetdb` data module. The link count of `inetdb` is decremented. `endprotoent()` returns `0` if successful; otherwise it returns an error.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Errors

`ENOMEM`	Insufficient RAM (POSIX).
`EOS_PARAM`	Bad parameter.

See Also

`getprotobyname()`
`getprotoent()`

endresolvent()

Unlink from Network Database

Syntax

```
#include <netdb.h>
endresolvent(void)
```

Libraries

`netdb.l`

Description

`endresolvent()` indicates the process is finished using the resolve section of the `inetdb` data module. The link count of `inetdb` is decremented. `endresolvent()` returns `0` if successful; otherwise it returns an error.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Errors

`ENOMEM`	Insufficient RAM (POSIX).
`EOS_PARAM`	Bad parameter.

See Also

`getresolvent()`

endroutent()

Unlink from Network Database

Syntax

```
#include <netdb.h>
error_code endroutent(void)
```

Libraries

`netdb.l`

Description

`endroutent()` indicates the process is finished using the route section of the `inetdb` data module. The link count of `inetdb` is decremented. If successful, `endroutent()` returns 0.; otherwise it returns an error.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Errors

`ENOMEM`	Insufficient RAM (POSIX).
`EOS_PARAM`	Bad parameter.

endservent()

Unlink from Network Database

Syntax

```
#include <netdb.h>
int endservent(void)
```

Libraries

`netdb.l`

Description

`endservent()` indicates the process is finished using the services section of the `inetdb` data module. The link count of `inetdb` is decremented. `endservent()` returns `0` if successful; otherwise it returns an error.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Errors

`ENOMEM`	Insufficient RAM (POSIX).
`EOS_PARAM`	Bad parameter.

See Also

`getnetbyaddr()`
`getservbyname()`
`getservent()`
`setservent()`

gethostbyaddr()

Get Network Host Entry by Address

Syntax

```
#include <netdb.h>
struct hostent *gethostbyaddr(
     const char   *addr,
     int          len,
     int          type)
```

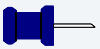
Libraries

```
netdb.l
```

Description

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

`gethostbyaddr()` returns a pointer to a `hostent` structure in the `inetdb` data module. `hostent` structure is defined in the **Structures** section. A null pointer (0) returns on `EOF` or an error is returned and `errno` is set to the error value.



Note

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

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`addr`	A pointer to the Internet address of the host to get in network order.
`len`	Specifies the length of the address in bytes.
`type`	Specifies the `AF_INET` address type.

Errors

`EOS_MNF`	`inetdb` module could not be found.

See Also

`gethostbyname()`
`gethostent()`
`sethostname()`

gethostbyname()

Get Network Host Entry by Name

Syntax

```
#include <netdb.h>
struct hostent *gethostbyname(const char *name)
```

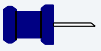
Libraries

`netdb.l`

Description

`gethostbyname()` sequentially searches from the beginning of the `hosts` entries of `inetdb` until a matching host name or alias is found, or until `EOF` is encountered.

`gethostbyname()` returns a pointer to a hostent structure in the `inetdb` data module. `hostent` structure is defined in the **Structures** section, under `hostent`. A null pointer (0) returns on `EOF` or error and `errno` is set to the error value.



Note

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

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`name`	A pointer to the name of the host.

Errors

EOS_MNF	netdb module could not be found.

See Also

endhostent()
gethostbyaddr()
gethostent()

gethostent()

Get Network Host Entry

Syntax

```
#include <netdb.h>
struct hostent *gethostent(void)
```

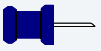
Libraries

`netdb.l`

Description

`gethostent()` reads the next host entry from `inetdb`. it returns a pointer to a `hostent` structure in the `inetdb` data module. (The `hostent` structure is defined in the **Structures** section.)

A null pointer returns on `EOF` or error.



Note

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

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

See Also

`endhostent()`
`gethostbyaddr()`
`gethostbyname()`
`sethostent()`

gethostname()

Gets Name of Current Host

Syntax

```
#include <sys/socket.h>
int gethostname(
     char         *name,
     int          namelen)
```

Libraries

`socket.l`

Description

`gethostname()` returns the host name for the current device. The returned name is null-terminated string.

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

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`name`	Points to the standard host name.
`namelen`	Specifies the size of the name array.

Errors

`EOS_MNF`	`netdb` module could not be found.

getinetdent()

Get Inetd Entry

Syntax

```
#include <netdb.h>
struct inetdent *getinetdent(void)
```

Libraries

`netdb.l`

Description

`getinetdent()` gets the next `inetd` entry. On success, `getinetdent()` returns a pointer to a `inetdent` structure in the `inetdb` module. A null pointer (0) returns on error and `errno` is set to the error value.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Errors

`EOS_MNF`	`netdb` module could not be found.
`EOS_PARAM`	Bad parameter.

getintent()

Get Interface Entry

Syntax

```
#include <netdb.h>
char *getintent(void)
```

Libraries

`netdb.l`

Description

`getintent()` gets the next interface entry. On success, `getintent()` returns a pointer that is cast to a `n_ifnet` structure. Immediately following the `n_ifnet` structure is a `u_int32`, which indicates how many `n_ifaliasreq` structures follow it. Otherwise, it returns a null pointer (0) with the appropriate error code placed in the global variable `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Errors

`EOS_MNF`	`netdb` module could not be found.
`EOS_PARAM`	Bad parameter.

getnetbyaddr()

Get Network Entry by Address

Syntax

```
#include <netdb.h>
struct netent *getnetbyaddr(
     long         net,
     int          type)
```

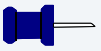
Libraries

`netdb.l`

Description

`getnetbyaddr()` sequentially searches from the beginning of the `networks` entries of `inetdb` until a matching `net` address and `type` is found, or until `EOF` is encountered.

`getnetbyaddr()` returns a pointer to a `netent` structure in the `inetdb` data module. A null pointer (0) returns on `EOF` or error and `errno` is set to the error value. `netent` structure is defined in the **Structures** section.



Note

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

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`net`	The network number (network byte order).
`type`	The network number type (network byte order).

Errors

`EOS_MNF`	`netdb` module could not be found.
`EOS_PARAM`	Bad parameter.

See Also

`endnetent()`
`getnetbyname()`
`getnetent()`
`setnetent()`

getnetbyname()

Get Network Entry by Name

Syntax

```
#include <netdb.h>
struct netent *getnetbyname(const char *name)
```

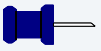
Libraries

`netdb.l`

Description

`getnetbyname()` sequentially searches from the beginning of the `networks` entries of `inetdb` until a matching name or alias is found, or until `EOF` is encountered.

`getnetbyname()` returns a pointer to a `netent` structure in the `inetdb` data module. A null pointer (0) returns on `EOF` or error and `errno` is set to the error value. `netent` structure is defined in the **Structures** section.



Note

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

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`name`	A pointer to the network name.

Errors

EOS_MNF	netdb module could not be found.
EOS_PARAM	Bad parameter.

See Also

endnetent()
getnetbyaddr()
getnetent()
setnetent()

getnetent()

Get Network Entry

Syntax

```
#include <netdb.h>
struct netent *getnetent(void)
```

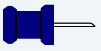
Libraries

`netdb.l`

Description

`getnetent()` reads the next `inetdb` network entry.

`getnetent()` returns a pointer to a `netent` structure in the `inetdb` data module. A null pointer (0) on `EOF` or error and `errno` is set to the error value. `netent` structure is described in the **Structures** section.



Note

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

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Errors

`EOS_MNF`	`netdb` module could not be found.
`EOS_PARAM`	Bad parameter.

See Also

endnetent()
getnetbyaddr()
getnetbyname()
setnetent()

getpeername()

Get Network Entry

Syntax

```
#include <sys/socket.h>
int getpeername(
     int          s,
     struct sockaddr *name,
     int          *namelen)
```

Libraries

```
socket.l
```

Description

`getpeername()` returns the name of the remote node (peer) connected to socket `s`. If successful, `getpeername()` returns `0`. Otherwise, it returns `-1` with `errno` set to the error value.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`s`	Specifies the path number of the socket.
`name`	A pointer to the socket address.
`namelen`	Before calling, initialize `namelen` to indicate the amount of space pointed to by name. On return, it contains the actual size of the name returned in bytes.

Errors

`ENOTCONN`	The socket is not connected.

See Also

bind()
getsockname()
socket()

getprotobyname()

Get Protocol Entry

Syntax

```
#include <netdb.h>
struct protoent *getprotobyname(const char *name)
```

Libraries

`netdb.l`

Description

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

`getprotobyname()` returns a pointer to a `protoent` structure in the `inetdb` data module. `getprotobyname()` returns a null pointer (0) on `EOF` or error and places the error value in errno. `protoent` structure is defined in the **Structures** section.

Note

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

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`name`	A pointer to the name of the protocol.

Errors

EOS_MNF	netdb module could not be found.
EOS_PARAM	Bad parameter.

See Also

endprotoent()
getprotobynumber()
getprotoent()
setprotoent()

getprotobynumber()

Get Protocol Entry by Number

Syntax

```
#include <netdb.h>
struct protoent *getprotobynumber(long proto)
```

Libraries

`netdb.l`

Description

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

`getprotobynumber()` returns a pointer to a protoent structure in the `inetdb` data module. `getprotobynumber()` returns a null pointer (0) on EOF or error and sets `errno` to the error value. `protoent` structure is defined in the **Structures** section.

Note

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

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`proto`	Specifies the protocol number in host byte order.

Errors

EOS_MNF	`inetdb` module could not be found.
EOS_PARAM	Bad parameter.

See Also

endprotoent()
getprotobyname()
getprotoent()
setprotoent()

getprotoent()

Get Protocol Entry

Syntax

```
#include <netdb.h>
struct protoent *getprotoent(void)
```

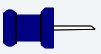
Libraries

`netdb.l`

Description

`getprotoent()` reads the next protocols entry of `inetdb`.

If successful, `getprotoent()` returns a pointer to a `protoent` structure in the `inetdb` data module. On `EOF` or error It returns a null pointer (0) and sets `errno` to the error value. `protoent` structure is defined in the **Structures** section.



Note

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

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Errors

`EOS_MNF`	`inetdb` module could not be found.
`EOS_PARAM`	Bad parameter.

See Also

endprotoent()
getprotobyname()
getprotobynumber()
setprotoent()

getresolvent()

Returns Pointer to DNS Structure

Syntax

```
#include <netdb.h>
struct resolvent *getresolvent(void)
```

Libraries

```
netdb.l
```

Description

`getresolvent()` returns a pointer to the resolver structure used by the DNS in the `netdb` data module. It returns the “last” one it finds. For example, the following modules are searched in this order (when x<4):

```
inetdbx
inetdbx-1
inetdb2
inetdb
```

The last valid pointer in the nameservers and search arrays is followed by a null pointer to indicate the end of valid data. On success, `getresolvent()` returns a pointer to a `resolvent` structure in the `inetdb` data module. On failure, it returns a null pointer and sets `errno` to the error value. `resolve` structure is defined in the **Structures** section.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Errors

`EOS_MNF`	`inetdb` module could not be found.
`EOS_PARAM`	Bad parameter.

See Also

delresolvent()
endresolvent()

getservbyname()

Get Service Entry by Name

Syntax

```
#include <netdb.h>
struct servent *getservbyname(
     const char   *name,
     const char   *proto)
```

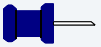
Libraries

`netdb.l`

Description

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

`getservbyname()` returns a pointer to a `servent` structure in the inetdb data module. `getservbyname()` returns a null pointer (0) on `EOF` or error and sets `errno` to the error value. `servent` is defined in the **Structures** section.



Note

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

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`name`	A pointer to the name of the service.
`proto`	A pointer to the name of the protocol.

Errors

`EOS_MNF`	`inetdb` module could not be found.
`EOS_PARAM`	Bad parameter.

See Also

`endservent()`
`getservbyport()`
`getservent()`
`setservent()`

getservbyport()

Get Service Entry by Port

Syntax

```
#include <netdb.h>
struct servent *getservbyport(
     long         port,
     const char   *proto)
```

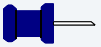
Libraries

`netdb.l`

Description

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

`getservbyport()` returns a pointer to a `servent` structure in the `inetdb` data module. `getservbyport()` returns a null pointer (0) on `EOF` or error.The `servent` structure is defined in the **Structures** section.



Note

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

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`port`	Specifies the service port number in network byte order.
`proto`	A pointer to the protocol name.

Errors

`EOS_MNF`	`inetdb` module could not be found.
`EOS_PARAM`	Bad parameter.

See Also

`endservent()`
`getservbyname()`
`getservent()`
`setservent()`

getservent()

Get Service Entry

Syntax

```
#include <netdb.h>
struct servent *getservent(void)
```

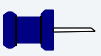
Libraries

`netdb.l`

Description

`getservent()` reads the next `services` entry of `inetdb`.

`getservent()` returns a pointer to an a servent structure in the inetdb data module. It returns a null pointer (0) on `EOF` or error and sets `errno` to the error value. `servent` structure is defined in the **Structures** section.



Note

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

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Errors

`EOS_MNF`	`inetdb` module could not be found.
`EOS_PARAM`	Bad parameter.

See Also

endservent()
getservbyname()
getservbyport()
setservent()

getsockname()

Gets Socket Name

Syntax

```
#include <sys/socket.h>
int getsockname(
     int          s,
     struct sockaddr *name,
     int          *namelen)
```

Libraries

```
socket.l
```

Description

`getsockname()` returns the current local node name for the specified socket. It returns `0` if the call succeeds. Otherwise, it returns `-1` and sets `errno` to the error value.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`s`	Specifies the path number of the socket.
`name`	Points to the socket address.
`namelen`	Before calling, initialize `namelen` to indicate the number of bytes pointed to by name. On return, it contains the actual size in bytes of the name returned.

Errors

ENOBUFS	Insufficient resources are available in the system to perform the operation.
EBUFTOOSMALL	Return buffer (name) is too small.

See Also

bind()
getpeername()
socket()

getsockopt()

Get Socket Options

Syntax

```
#include <sys/socket.h>
int getsockopt(
     int          s,
     int          level,
     int          optname,
     void         *optval,
     int          *optlen)
```

Libraries

`socket.l`

Description

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

If successful, the call returns `0`. Otherwise, it returns `-1` and sets `errno` to the error value.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Options

`socket.h` contains definitions for the socket level options. Options at other protocol levels vary in format and name and are defined in the protocol's header file. Supported socket options are described in **Table 1-8 Socket Level Options**.

Table 1-8 Socket Level Options

Level	optname	Description	Data Type
IPPROTO_IP	IP_MULTICAST_TTL	Get TTL for multicast packets	u_char
IPPROTO_IP	IP_MULTICAST_LOOP	Send multicast packets to the loopback interface	u_char
IPPROTO_IP	IP_MULTICAST_IF	Retrieve interface used for sending multicast packets	in_addr{}
IPPROTO_TCP	TCP_MAXSEG	Get maximum segment size.	int
IPPROTO_TCP	TCP_NODELAY	Do not delay send to coalesce packets.	int
SOL_SOCKET	SO_KEEPALIVE	Keep connection alive by forcing peer to respond periodically.	int
SOL_SOCKET	SO_LINGER	Linger on close if data present.	linger{}
SOL_SOCKET	SO_REUSEADDR	Allow local address reuse.	int
SOL_SOCKET	SO_BROADCAST	Permit sending broadcast datagrams.	int
SOL_SOCKET	SO_OOBINLINE	Leave out-of-band data in normal input queue.	int
SOL_SOCKET	SO_SNDBUF	Get size of send buffer.	int
SOL_SOCKET	SO_RCVBUF	Get size of receive buffer.	int
SOL_SOCKET	SO_SNDLOWAT	Minimum space required in send buffer before to accept more data.	int

Table 1-8 Socket Level Options (continued)

Level	optname	Description	Data Type
SOL_SOCKET	SO_USELOOPBACK	Routing socket receives copy of any data sent (AF_ROUTE only).	int
SOL_SOCKET	SO_TYPE	Get socket type (SOCK_STREAM, SOCK_DGRAM, SOCK_RAW).	int
SOL_SOCKET	SO_ERROR	Retrieve pending socket error	int

Parameters

`s` Specifies the path number of the socket.

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

- To get options at the socket level, specify `level` as SOL_SOCKET.
- To get options at any other level, supply the protocol number of the appropriate protocol controlling the option. For example, IPPROTO_TCP specifies TCP options.

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

`optval` A pointer to the buffer for the requested option. `optval` and `optlen` together identify the buffer in which to return the value for the requested option(s). If an option value is not to be supplied or returned, set `optval` to `0`.

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

Errors

`ENOPROTOOPT`	The option is unknown.

See Also

`setsockopt()`
`socket()`

htonl()

Convert 32-Bit Values from Host to Network Byte Order

Syntax

```
#include <sys/endian.h>
u_long  htonl(u_long hostlong)
```

Libraries

`netdb.l`

Description

`htonl()` converts 32-bit quantities from host to network byte order.

`htonl()` returns the long in a network byte order representation.

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

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

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

See Also

`htons()`
`ntohl()`
`ntohs()`

htons()

Convert 16-Bit Values from Host to Network Byte Order

Syntax

```
#include <sys/endian.h>
u_short htons(u_short hostshort)
```

Libraries

`netdb.l`

Description

`htons()` converts 16-bit quantities from host to network byte order.

`htons()` returns the short in a network byte order representation.

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

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

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

See Also

`htonl()`
`ntohl()`
`ntohs()`

inet_addr()

Convert Dot Notation into Network Address

Syntax

```
#include <netdb.h>
unsigned long inet_addr(char *cp)
```

Libraries

`netdb.l`

Description

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

On error, `inet_addr()` returns `0xFFFFFF` (or `-1` when cast to unsigned).

For More Information

Refer to the ***Using LAN Communications Pak*** manual for more information about Internet addresses.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`cp`	A pointer to the Internet address character string.

See Also

inet_aton()
inet_lnaof()
inet_makeaddr()
inet_netof()
inet_network()
inet_ntoa()

inet_aton()

Convert Internet Address to Binary Address

Syntax

```
#include <netdb.h>
inet_aton(const char *cp, struct in_addr *addr)
```

Libraries

`netdb.l`

Description

`inet_aton()` determines whether the parameter `cp` is a valid ASCII representation of an Internet address and converts it to a binary address.

`inet_aton()` returns 1 if the address is valid and 0 if the address is not valid. This return value is more sophisticated than that of `inet_addr()`, which can not distinguish whether the return is a failure or a broadcast address.

For More Information

Refer to the ***Using LAN Communications Pak*** manual for more information about Internet addresses.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`cp`	A pointer to the Internet address character string.
`addr`	A pointer to the resulting Internet address.

See Also

`inet_addr()`
`inet_lnaof()`
`inet_makeaddr()`
`inet_netof()`
`inet_network()`
`inet_ntoa()`

inet_lnaof()

Get Local Address

Syntax

```
#include <netdb.h>
int inet_lnaof(struct in_addr in)
```

Libraries

`netdb.l`

Description

`inet_lnaof()` returns the host address portion of an Internet address (in host byte order). All host address parts are returned as integer values.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`in`	Specifies the Internet address to break apart.

See Also

`inet_addr()`
`inet_aton()`
`inet_makeaddr()`
`inet_netof()`
`inet_network()`
`inet_ntoa()`

inet_makeaddr()

Get Address from Network and Host Address

Syntax

```
#include <sys/socket.h>
#include <netdb.h>
struct in_addr inet_makeaddr(
     int          net,
     int          lna)
```

Libraries

`netdb.l`

Description

`inet_makeaddr()` takes an Internet network number and a local network address and constructs an Internet address from it.

For More Information

Refer to the ***Using LAN Communications Pak*** manual for more information about Internet addresses.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`net`	Specifies the network number in host byte order.
`lna`	Specifies the host number.

See Also

inet_addr()
inet_aton()
inet_lnaof()
inet_netof()
inet_network()
inet_ntoa()

inet_netof()

Get Network Number

Syntax

```
#include <netdb.h>
int inet_netof(struct in_addr in)
```

Libraries

`netdb.l`

Description

`inet_netof()` takes as input an Internet host address and returns the network number (in host byte order).

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`in`	Specifies the Internet address.

See Also

`inet_addr()`
`inet_aton()`
`inet_lnaof()`
`inet_makeaddr()`
`inet_network()`
`inet_ntoa()`

inet_network()

Interpret Network Number

Syntax

```
#include <netdb.h>
unsigned long inet_network(char *cp)
```

Libraries

`netdb.l`

Description

`inet_network()` interprets character strings representing numbers expressed in the Internet standard "`.`" notation. `inet_network()` returns numbers suitable for use as Internet network numbers. Network numbers are returned as unsigned long values (in host byte order).

On error, `inet_network()` returns `0xFFFFFFFF` (or `-1` when cast to unsigned).

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

cp	A pointer to a character string.

See Also

`inet_addr()`
`inet_aton()`
`inet_lnaof()`
`inet_makeaddr()`
`inet_netof()`
`inet_ntoa()`

inet_ntoa()

Return Address in Dot Notation

Syntax

```
#include <netinet/in.h>
#include <sys/socket.h>
#include <netdb.h>
char *inet_ntoa(struct in_addr in)
```

Libraries

`netdb.l`

Description

`inet_ntoa()` takes an Internet address and returns a pointer to a string in the Internet standard dot notation.

For More Information

Refer to the ***Using LAN Communications Pak*** manual for more information about Internet addresses.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`in`	Specifies the Internet address to be converted to a string in network byte order.

See Also

inet_addr()
inet_aton()
inet_lnaof()
inet_makeaddr()
inet_netof()
inet_network()

ip_start()

Initialize IP stack

Syntax

```
#include <sys/socket.h>
error_code ip_start(void)
```

Libraries

`socket.l`

Description

`ip_start()` initializes the IP stack and all the configured drivers. If successful, `ip_start()` returns 0. Otherwise, it returns the appropriate error code.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Errors

`EOS_MNF`	Module not found.

listen()

Listen for Connections on Socket

Syntax

```
#include <sys/socket.h>
int listen(
     int          s,
     int          backlog)
```

Libraries

```
socket.l
```

Description

`listen()` marks an existing socket as willing to accept incoming connections. The incoming connections are queued until `accept()` is called to retrieve them. The listen call applies only to sockets of type `SOCK_STREAM`.

`listen()` returns `0` if successful. Otherwise, it returns a `-1` with the appropriate code in `errno`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`s`	Specifies the path number of the socket.
`backlog`	Defines the maximum length to which the queue of pending connections may grow. The backlog is limited from 0 to the maximum queue of 128. If the backlog is greater than 128, it defaults to 128. If a connection request arrives with the queue full, the client receives an `ECONNREFUSED` error.

Errors

`EINVAL`	The socket must be bound in order to listen.
`EOPNOTSUPP`	The socket is not of a type supporting the operation `listen()`.

See Also

`accept()`
`connect()`
`socket()`

ndb_create_ndbmod()

Create Network Database Module

Syntax

```
#include <netdblib.h>
error_code
ndb_create_ndbmod(
     char          *modname,
     int           num_files,
     int           *file_sizes,
     u_int32       perm,
     u_int16       rev);
```

Libraries

ndblib.l

Description

create_ndbmod creates the Internet data module modname and reserves space as specified by the parameters passed.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

modname	Name of the module to create. While it can be any string, the ndb_link_ndbmod function (and hence the netdb calls) only check for inetdb, inetdb2, inetdb3, inedb4.

`num_files` The number of different "files" (record types) that this module stores. File numbers 1-32 are assigned as follows:

Table 1-9 File number designations

1	hosts (approx. 25 bytes per host)
2	hosts equiv (not used)
3	networks (approx. 40 bytes per network)
4	protocols (approx. 25 bytes per protocol)
5	services (approx. 25 bytes per service)
6	`inetd` entries (approx. 50 bytes per entry)
7	DNS client configuration (approx. 100 bytes)
8	local host configuration (not used)
9	host interfaces (approx 200 bytes per interface)
10	hostname (>= length of hostname + 1, recommended 65)
11	static routes (approx. 64 bytes per entry)
12-32	Reserved

`file_sizes` An array of size `num_files`, where the value of element N indicates how many bytes to reserve for file N+1. If `file_sizes[6] = 400`, then 400 bytes are reserved for storing resolver information.

`perm` The permission given to the newly created module. If future updates are allowed, the user must have permission to link, read, and write to the module.

`rev` The revision number given to the newly created module. An existing module may be overlaid if a higher revision number is specified.

For More Information

See **Appendix B: Dynamic Configuration of the inetdb Data Module**.

Errors

EOS_PARAM	Bad parameter.

ntohl()

Convert 32-Bit Values from Network to Host Byte Order

Syntax

```
#include <sys/endian.h>
u_long ntohl(u_long netlong)
```

Libraries

`netdb.l`

Description

`ntohl()` converts 32-bit quantities from network to host byte order.

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

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`netlong`	Specifies the network byte order to convert to host long.

See Also

`htonl()`
`htons()`
`ntohs()`

ntohs()

Convert 32-Bit Values from Network to Host Byte Order

Syntax

```
#include <sys/endian.h>
u_short ntohs(u_short netshort)
```

Libraries

`netdb.l`

Description

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

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

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`netshort`	Specifies the network byte order to convert to host short.

See Also

`htonl()`
`htons()`
`ntohl()`

ppp_auth_add_chap()

Add CHAP Entry to Authentication Module

Syntax

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

error_code
ppp_auth_add_chap(
        char        *peer_name,
        char        *id,
                     char *secret,
        auth_handle *hndl);
```

Libraries

ppplib.l

Description

This function adds new CHAP peer/ID/secret group to the authentication module. The peer name must be unique; each peer may only have only one CHAP entry and one PAP entry. If a peer already has a CHAP entry within the database, the existing entry is overwritten.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe

Parameters

`peer_name`	pointer to new peer name This string should not exceed `PPP_MAX_PEER_NAME` in length.
`id`	pointer to new name This string should not exceed `PPP_MAX_NAME` bytes in length.
`secret`	pointer to new CHAP secret This string should not exceed `PPP_MAX_SECRET` bytes in length.
`hndl`	handle to the authentication module This parameter is obtained from `ppp_auth_create_mod()` or `ppp_auth_link_mod()`.

Errors

`EOS_NOTRDY`	returned when the API has not been initialized
`EOS_FULL`	no more free entries within the authentication module
`EOS_ILLPRM`	illegal parameter

ppp_auth_add_pap()

Add PAP Entry to Authentication Module

Syntax

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

error_code
ppp_auth_add_pap(
        char        *peer_name,
        char        *id,
                    char *secret,
        auth_handle *hndl);
```

Libraries

ppplib.l

Description

This function adds new PAP peer/ID/secret group to the authentication module. The peer name must be unique in that each peer may only have one CHAP entry and one PAP entry. If a peer already has a PAP entry within the database, the existing entry is overwritten.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe

Parameters

`peer_name`	pointer to new peer name This string should not exceed `PPP_MAX_PEER_NAME` in length.
`id`	pointer to new name This string should not exceed `PPP_MAX_NAME` bytes in length.
`secret`	pointer to new PAP secret This string should not exceed `PPP_MAX_SECRET` bytes in length.
`hndl`	handle to the authentication module. This parameter is obtained from `ppp_auth_create_mod()` or `ppp_auth_link_mod()`.

Errors

`EOS_NOTRDY`	returned when the API has not been initialized
`EOS_FULL`	allows no more free entries within the authentication module
`EOS_ILLPRM`	illegal parameter

ppp_auth_create_mod()

Create a New Authentication Module (Database)

Syntax

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

error_code
ppp_auth_create_mod(
        u_int16       max_entries,
        auth_handle   *hndl);
```

Libraries

`ppplib.l`

Description

This function creates a new authentication module. This module is used to store authentication information when PAP or CHAP authentication is being used to connect to a remote peer.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe

Parameters

`max_entries`	maximum of host/ID/secret groups to be stored in the module
`hndl`	location where `ppp_auth_create_mod()` will store a handle to the new authentication module This handle will be used in all subsequent calls to the authentication functions in this library.

Errors

`EOS_KWNMOD`	signifies that the authentication module already exists If this error appears, a call to `ppp_auth_link_mod()` should be made.

See Also

`ppp_auth_link_mod()`

`auth_handle structure`

ppp_auth_del_chap()

Delete CHAP Entry from Authentication Module

Syntax

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

error_code
ppp_auth_del_chap(
        char            *peer_name,
        auth_handle     *hndl);
```

Libraries

ppplib.l

Description

This function deletes existing CHAP entry for specified peer from the authentication module.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe

Parameters

peer_name	pointer to peer name This string should not exceed PPP_MAX_PEER_NAME in length.
hndl	handle to the authentication module. This parameter is obtained from ppp_auth_create_mod() or ppp_auth_link_mod().

Errors

EOS_NOTRDY	returned when the API has not been initialized
EOS_PNNF	no CHAP entry for specified peer

ppp_auth_del_pap()

Delete PAP Entry from Authentication Module

Syntax

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

error_code
ppp_auth_del_pap(
        char            *peer_name,
        auth_handle     *hndl);
```

Libraries

```
ppplib.l
```

Description

This function deletes existing PAP entry for specified peer from the authentication module.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe

Parameters

`peer_name`	pointer to peer name This string should not exceed `PPP_MAX_PEER_NAME` in length.
`hndl`	handle to the authentication module. This parameter is obtained from `ppp_auth_create_mod()` or `ppp_auth_link_mod()`.

Errors

EOS_NOTRDY	returned when the API has not been initialized
EOS_PNNF	no PAP entry for specified peer

ppp_auth_get_cur_chap()

Get CHAP Name/Secret for Currently Set Peer

Syntax

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

error_code
ppp_auth_get_cur_chap(
        char            *name,
        char            *secret,
        auth_handle     *hndl);
```

Libraries

`ppplib.l`

Description

This call gets the CHAP secret needed to connect to the current host.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe

Parameters

`name`	location of a character array containing the CHAP name This NULL-terminated character array should no more than `PPP_MAX_NAME` bytes in length.
`secret`	location of a character array where the CHAP secret will be placed This character array should be at most `PPP_MAX_SECRET` bytes in length.

`hndl`	handle to the authentication module This parameter is obtained from `ppp_auth_create_mod()` or `ppp_auth_link_mod()`.

Errors

`EOS_NOTRDY`	returned when the API has not been initialized
`EOS_BNAM`	signifies that the secret could not be found for the current host and name

See Also

`ppp_auth_set_peer_name()`

`ppp_auth_get_peer_name()`

ppp_auth_get_cur_pap()

Get PAP Name/Secret for Currently Set Peer

Syntax

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

error_code
ppp_auth_get_cur_pap(
        char            *name,
        char            *secret,
        auth_handle     *hndl);
```

Libraries

ppplib.l

Description

This call gets the PAP secret needed to connect to the current host.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe

Parameters

name	location of a character array containing the PAP name This character array must be no more than PPP_MAX_NAME bytes in length.
secret	location of a character array where the PAP secret will be placed This character array should be at least PPP_MAX_SECRET bytes in length.

`hndl`	handle to the authentication module. This parameter is obtained from `ppp_auth_create_mod()` or `ppp_auth_link_mod()`.

Errors

`EOS_NOTRDY`	returned when the API has not been initialized
`EOS_BNAM`	secret could not be found for the current host and name

See Also

`ppp_auth_set_peer_name()`

`ppp_auth_get_peer_name()`

ppp_auth_get_peer_name()

Get Name of Currently Set Peer

Syntax

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

error_code
ppp_auth_get_peer_name(
        char            *name,
        auth_handle     *hndl);
```

Libraries

`ppplib.l`

Description

This call gets the current remote peer name from the authentication module, as was set with a previous `ppp_auth_set_peer_name()` call.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe

Parameters

`name`	location to store the current peer name This buffer should be at least `PPP_MAX_PEER_NAME` bytes in length
`hndl`	handle to the authentication module This parameter is obtained from `ppp_auth_create_mod()` or `ppp_auth_link_mod()`

Errors

`EOS_NOTRDY`	returned when the API has not been initialized

See Also

`ppp_auth_set_peer_name()`

ppp_auth_link_mod()

Link to Existing Authentication Module

Syntax

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

error_code
ppp_auth_link_mod(
        auth_handle      *hndl);
```

Libraries

```
ppplib.l
```

Description

This function links to an existing authentication module. It is used to store authentication information when PAP or CHAP authentication is being used to connect to a remote peer.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe

Parameters

`hndl`	location where `ppp_auth_create_mod()` will store a handle to the new authentication module This handle will be used in all subsequent calls to the authentication functions in this library.

Errors

`EOS_NOTRDY`	returned when the API has not been initialized

EOS_MNF	signifies that the authentication module was not found in memory If this error appears, a call to ppp_auth_create_mod() should be made.

See Also

ppp_auth_create_mod()

auth_handle structure

ppp_auth_set_peer_name()

Set Current Peer

Syntax

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

error_code
        ppp_auth_set_peer_name(
        char            *name,
        auth_handle     *hndl);
```

Libraries

`ppplib.l`

Description

This function sets the current remote peer name in the authentication module. This determines which values (such as PAP/CHAP names and secrets) will be get/set in subsequent authentication calls.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe

Parameters

`name`	location of the new remote peer name This character array should be no more than `PPP_MAX_PEER_NAME` bytes in length.
`hndl`	handle to the authentication module This parameter is obtained from `ppp_auth_create_mod()` or `ppp_auth_link_mod()`.

Errors

`EOS_NOTRDY`	returned when the API has not been initialized

See Also

`ppp_auth_get_peer_name()`

ppp_auth_unlink_mod()

Unlink from an Authentication Module

Syntax

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

error_code
ppp_auth_unlink_mod(
        auth_handle     *hndl);
```

Libraries

```
ppplib.l
```

Description

This call unlinks from an authentication module previously linked to with `ppp_auth_link_mod()`, or created with `ppp_auth_create_mod()`.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe

Parameters

`hndl`	handle of the authentication module to unlink

Errors

`EOS_NOTRDY`	returned when the API has not been initialized
`EOS_MNF`	signifies that the authentication module was not found in memory

See Also

`ppp_auth_create_mod()`

`ppp_auth_link_mod()`

`auth_handle structure`

ppp_chat_close()

Close CHAT Path

Syntax

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

error_code
ppp_chat_close(
         path_id      chat_path);
```

Libraries

`ppplib.l`

Description

This function closes the path opened by `ppp_chat_open()`.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe

Parameters

`chat_path`	path identifier that was returned from a successful call to `ppp_chat_open()`

Errors

`EOS_NOTRDY`	returned when the API has not been initialized
`EOS_BPNUM`	returned when this not a valid path identifier

See Also

ppp_open()

ppp_chat_write()

ppp_chat_read()

ppp_start()

ppp_chat_open()

Open a Raw CHAT Path

Syntax

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

error_code
ppp_chat_open(
        char        *hdlc_name,
        path_id     *chat_path);
```

Libraries

`ppplib.l`

Description

This function opens a path to the HDLC driver and places the driver into CHAT mode. When in this mode, the caller can use the `chat_path` to read/write directly from or to the communications device below the HDLC layer.

The name of the HDLC descriptor should be the same as the one used within the PPP stack name. For example, the PPP stack name of `</dev>/hdlc0/lcp0/ipcp0` would use `/hdlc0` for the `hdlc_name` field.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe

Parameters

`stack_name`	name of the HDLC descriptor contained within the PPP stack name.
`chat_path`	returns the path of the HDLC layer at this location

Errors

`EOS_NOTRDY`	returned when the API has not been initialized
`EOS_BPNUM`	returned when this not a valid PPP stack path identifier

See Also

`ppp_open()`

`ppp_chat_write()`

`ppp_chat_read()`

`ppp_chat_close()`

`ppp_start()`

ppp_chat_read()

Read Data from CHAT Path

Syntax

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

error_code
ppp_chat_read(
        path_id m   chat_path,
        void         *buffer,
        u_int32      *count);
```

Libraries

ppplib.l

Description

This function reads user data from the path. Use this call after ppp_chat_open() to read data from the data port without HDLC decoding.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe

Parameters

chat_path	path identifier that was returned from a successful call to ppp_chat_open()
buffer	pointer to the caller's character buffer
count	pointer to the number of bytes in the caller's character buffer After the call completes, the number of bytes actually read is returned here.

Errors

EOS_NOTRDY	returned when the API has not been initialized
EOS_BPNUM	returned when this not a valid PPP stack path identifier

For More Information

For more information on errors associated with `ite_data_read()` refer to the ***SPF Programming Reference Manual*** included with this CD.

See Also

`_os_read()`

`ppp_chat_write()`

ppp_chat_script()

Run a CHAT Script

Syntax

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

error_code
ppp_chat_script(
        path_id         chat_path,
        path_id         log_path,
        u_int8          chat_type,
        char            *chat_name,
        ppp_conninfo    *ci);
```

Libraries

`ppplib.l`

Description

This function runs a CHAT script from either a text file or a data module. An optional logging path may be specified where CHAT commands and responses will be echoed. A log path of `PPP_NOLOG` denotes no logging is desired.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe

Parameters

`chat_path`	CHAT path identifier that was returned from a successful call to `ppp_chat_open()`
`log_path`	path where CHAT commands and responses will be echoed for logging purposes A `log_path` of `PPP_NOLOG` may be used to prevent logging.
`chat_type`	specifies the container of the CHAT script commands: `PPP_CHAT_TYPE_MODULE` or `PPP_CHAT_TYPE_FILE`.
`chat_name`	name of the CHAT module or file
`ci`	pointer to structure allocated by the application that contains connection information for the life of the connection Applications must keep this memory structure to prevent errors from resulting.

Errors

`EOS_NOTRDY`	returned when the API has not been initialized
`EOS_BPNUM`	CHAT or log path is invalid
`EOS_ILLFNC`	unknown command in CHAT script
`EOS_ILLPRM`	bad argument in CHAT script
`EOS_NORAM`	unable to allocate CHAT engine memory
`EOS_PPP_CHAT_BADSTR`	Malformed string in CHAT script
`EOS_PPP_CHAT_ABORT`	script aborted due to reception of an ABORT string

EOS_PPP_CHAT_APPABORT	script aborted due to application setting the PPP_CIFLAG_CHATABORT flag within the ppp_conninfo data structure
ETIMEDOUT	communication with the remote peer timed out

See Also

ppp_chat_open()

ppp_chat_close()

ppp_start()

ppp_start()

Establish a PPP Link

Syntax

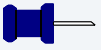
```
#include <SPF/ppplib.h>

error_code
ppp_start(
        path_id         stack_path,
        ppp_conninfo    *connection_info);
```

Libraries

```
ppplib.l
```

Description

This function completes the PPP connection after the CHAT operation has been performed. This call does not use the `chat_path`, rather the `stack_path`, which contains the complete PPP signaling stack.



Note

The `chat_path` must have been closed prior to making this call in order to allow the HDLC layer to begin HDLC framing.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe

Parameters

`stack_path`	path identifier that was returned from a successful call to `ppp_open()`
`connection_info`	pointer to structure allocated by the application that contains connection information for the life of the connection. Applications must keep this memory structure to prevent errors from resulting.

Errors

`EOS_NOTRDY`	returned when the API has not been initialized
`EOS_BPNUM`	returned when this not a valid PPP stack path identifier
`EOS_PARAM`	error in the chat commands
`ETIMEDOUT`	communication with the remote peer timed out

See Also

`ppp_open()`

`chat discussion`

ppp_chat_write()

Write Data to CHAT Path

Syntax

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

error_code
ppp_chat_write(
        path_id     chat_path,
        void        *buffer,
        u_int32     *count);
```

Libraries

```
ppplib.l
```

Description

This function writes user data down the CHAT path. Use this function after `ppp_chat_open()` to send data out the data port without HDLC framing.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe

Parameters

`chat_path`	path identifier returned from a successful call to `ppp_chat_open()`
`buffer`	pointer to the caller's character buffer
`count`	pointer to the number of bytes in the caller's character buffer After the call is completed, the number of bytes actually written is returned here.

Errors

EOS_NOTRDY	returned when the API has not been initialized
EOS_BPNUM	returned when this not a valid PPP stack path identifier

See Also

_os_write()

ppp_chat_read()

ppp_close()

Close the PPP Stack

Syntax

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

error_code
ppp_close(
     path_id    stack_path);
```

Libraries

```
ppplib.l
```

Description

This function closes the PPP stack associated with the `stack_path` handle.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe

Parameters

`stack_path`	path identifier returned from a successful call to `ppp_open()`

Errors

`EOS_NOTRDY`	returned when API has not been initialized
`EOS_DEVBSY`	returned if PPP link is still established
`EOS_BPNUM`	returned when not a valid PPP stack path identifier

See Also

ppp_open()

ppp_connect()

Run Optional CHAT Script and Establish PPP Link

Syntax

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

error_code
ppp_connect(
        path_id         stack_path,
        path_id         log_path,
        char            *hdlc_name,
        u_int8          chat_type,
        char            *chat_name,
        ppp_conninfo    *connection_info);
```

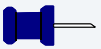
Libraries

```
ppplib.l
```

Description

This function connects an open PPP stack to a remote peer. It can be direct connect or dial-up. If the connection is dial-up, the function can parse a data module or disk file (a CHAT script) into send/expect command pairs, which are sent/received to/from a modem to establish the connection with the remote peer.

This function will return when the modems have connected and HDLC is enabled, when the protocol has timed out, or when the function encounters an error in the CHAT commands. If `stack_up_sig` and `stack_down_sig` in the `connection_info` structure are non-zero, the application will receive one or the other when the connection attempt resolves. If the signals are zero, the calling process will be sent to no signals.



Note

If it is necessary to run a customized chat script, use the `chat_open()`, `chat_write()`, `chat_read()`, `chat_close()` and `ppp_start()` calls instead of the `ppp_connect()` call.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe

Parameters

`stack_path`	path identifier returned from a successful call to `ppp_open()`
`log_path`	path where CHAT commands and responses will be echoed for logging purposes A `log_path` of `PPP_NOLOG` may be used for this parameter to prevent logging.
`hdlc_name`	name of the HDLC descriptor contained within the PPP stack name For example, the PPP stack name of `</dev>/hdlc0/lcp0/ipcp0` would use `/hdlc0` for this parameter.
`chat_type`	Specify the container of the send/expect commands: `PPP_CHAT_TYPE_MODULE`, `PPP_CHAT_TYPE_FILE`, or `PPP_CHAT_TYPE_NONE` (if no CHAT script is required).
`chat_name`	name of the chat module/file (or NULL pointer if chat_type is `PPP_CHAT_TYPE_NONE`).

connection_info	pointer to structure allocated by the application containing connection information for the life of the connection Applications must keep this memory structure to prevent errors from resulting.

Errors

EOS_NOTRDY	returned when the API has not been initialized
EOS_DEVBSY	returned when a PPP link is established
EOS_BPNUM	returned when this not a valid PPP stack path identifier
EOS_PARAM	there was an error in the chat commands
EOS_MNF	module not found
EOS_PNNF	disk file not found
ETIMEDOUT	Communication with the remote peer timed out.

See Also

ppp_open()

ppp_get_params()

ppp_set_params()

_os_intercept()

ppp_chat_open()

ppp_chat_write()

ppp_start()

ppp_disconnect()

Terminate Current PPP Link

Syntax

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

error_code
ppp_disconnect(
       path_id      stack_path);
```

Libraries

`ppplib.l`

Description

This call specifies a disconnect from the remote peer. This is done by sending a terminate request message to the remote peer. Also, a request is made to drop the modem carrier to the driver below the HDLC layer. After this call, it is safe for the application to return the memory for the `ppp_conninfo` structure passed in from the `ppp_connect()` or `ppp_start()`.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe

Parameters

`stack_path`	path identifier returned from a successful call to `ppp_open()`

Errors

EOS_NOTRDY	returned when the API has not been initialized or if a PPP link is not established
EOS_BPNUM	returned when this not a valid PPP path identifier

See Also

ppp_open()

ppp_connect()

ppp_get_asynch_params()

Get Asynchronous Link Parameters

Syntax

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

error_code
ppp_get_asynch_params(
        path_id       stack_path,
        ppp_modem_p   *params);
```

Libraries

```
ppplib.l
```

Description

This call gets the parameter values for the asynchronous PPP link. The desired options are returned in the `ppp_modem_p` structure.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe

Parameters

`stack_path`	path identifier that was returned from a successful call to `ppp_open()`
`params`	pointer to the location where `ppp_get_asynch_params ()` will store the current asynchronous parameter values for the PPP link

Errors

EOS_NOTRDY	returned when the API has not been initialized
EOS_BPNUM	returned when this not a valid PPP stack path identifier
EOS_PERMIT	invalid pointer
EOS_ILLPRM	illegal parameter

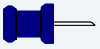
See Also

ppp_open()

ppp_set_asynch_params()

ppp_connect()

ppp_modem_p structure



Note

This call is valid only for asynchronous PPP links.

ppp_get_options()

Get Negotiable Stack Options

Syntax

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

error_code
ppp_get_options(
        path_id             stack_path,
        ppp_option_block    *options);
```

Libraries

```
ppplib.l
```

Description

This call gets the negotiable options of the drivers in an open PPP stack. The options will be stored in the specified `ppp_option_block`.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe

Parameters

`stack_path`	path identifier that was returned from a successful call to `ppp_open()`
`options`	pointer to the location at which `ppp_get_options()` will store the current options for the PPP stack

Errors

EOS_NOTRDY	returned when the API has not been initialized
EOS_BPNUM	returned when this not a valid PPP stack path identifier
EOS_PERMIT	invalid pointer
EOS_ILLPRM	illegal parameter

See Also

ppp_open()

ppp_set_options()

ppp_connect()

ppp_option_block structure

ppp_get_params()

Obtain Negotiated Stack Parameters

Syntax

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

error_code
ppp_get_params(
        path_id          stack_path,
        ppp_param_block  *params);
```

Libraries

`ppplib.l`

Description

This call obtains the current negotiated link parameters of the drivers in an open PPP stack. This information will be placed at the location of the indicated `ppp_param_block` structure.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe

Parameters

`stack_path`	path identifier that was returned from a successful call to `ppp_open()`
`params`	pointer to the location where `ppp_get_params()` will store the current negotiated parameters of the PPP stack

Errors

EOS_NOTRDY	returned when the API has not been initialized
EOS_BPNUM	returned when this not a valid PPP stack path identifier

See Also

ppp_open()

ppp_get_options()

ppp_set_options()

ppp_param_block structure

ppp_get_statistics()

Obtain Current Stack Statistics

Syntax

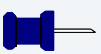
```
#include <SPF/ppplib.h>

error_code
ppp_get_statistics(
        path_id          stack_path,
        ppp_ipcp_stats   *ipcp_stats,
        ppp_lcp_stats    *lcp_stats,
        ppp_hdlc_stats   *hdlc_stats);
```

Libraries

```
ppplib.l
```

Description

This function queries the drivers in the specified PPP stack and returns current statistics for each layer. A NULL pointer may be passed in for any layer in which the caller is not interested.



Note

The LCP and IPCP layers do not currently support statistics, and therefore must be NULL.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe

Parameters

`stack_path`	path identifier that was returned from a successful call to `ppp_open()`
`ipcp_stats`	pointer to a user-allocated statistics structure where `ppp_get_statistics()` will return the IPCP-layer statistics
`lcp_stats`	pointer to a user-allocated statistics structure where `ppp_get_statistics()` will return the LCP-layer statistics
`hdlc_stats`	pointer to a user-allocated statistics structure where `ppp_get_statistics()` will return the HDLC-layer statistics

Errors

`EOS_NOTRDY`	returned when the API has not been initialized
`EOS_PERMIT`	invalid pointer was passed in
`EOS_BPNUM`	returned when this not a valid PPP stack path identifier

See Also

`ppp_open()`

`ppp_stats structure`

ppp_init()

Initialize the PPP API

Syntax

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

error_code
ppp_init(
     void     *rsvd);
```

Libraries

```
ppplib.l
```

Description

This call initializes the PPP API. This call must be made before any other calls are allowed. Currently, the single `rsvd` parameter is reserved for future use and must be NULL.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe

Parameters

`rsvd`	reserved for possible future use must be set to NULL

Errors

`EOS_DEVBSY`	returned when the API has been initialized

See Also

`ppp_close()`

`ppp_disconnect()`

ppp_open()

Open the PPP Stack

Syntax

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

error_code
ppp_open(
        char     *stack_name,
        path_id  *stack_path);
```

Libraries

```
ppplib.l
```

Description

This is the PPP function that opens the PPP stack.

Note

This function merely opens a path to the stack. To actually connect to a remote peer use `ppp_connect()` or `ppp_start()` after the `ppp_open()`. The path identifier that is returned from a successful `ppp_open()` should eventually be used in a call to `ppp_close()`.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe

Parameters

`stack_name`	NULL terminated string containing the name of the PPP stack (`/hdlc0/lcp0/ipcp0`)
`stack_path`	pointer to the location where `ppp_open()` will store the path identifier of the opened path

Errors

`EOS_NOTRDY`	returned when the API has not been initialized
`EOS_MNF`	returned when the specified descriptors are not found in memory

See Also

`ppp_close()`

`ppp_connect()`

ppp_reset_statistics()

Reset Stack Statistics

Syntax

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

error_code
ppp_reset_statistics(
        path_id   stack_path,
        u_int16   layers);
```

Libraries

`ppplib.l`

Description

This function resets the statistics in the specified PPP stack. `layers` is a bitmask that determines which layers should reset their statistics. Currently, only the HDLC layer supports statistics; therefore, the only valid value for layers is `PPP_LAYER_HDLC`.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe

Parameters

`stack_path`	path identifier that was returned from a successful call to `ppp_open()`
`layers`	bitmask that determines which layers should reset their statistics. Layer bit values are defined in `SPF/ppplib.h` (`PPP_LAYER_IPCP`, `PPP_LAYER_LCP`, and `PPP_LAYER_HDLC`).

Errors

EOS_NOTRDY	returned when the API has not been initialized
EOS_BPNUM	returned when this not a valid PPP stack path identifier

See Also

ppp_open()

ppp_stats structure

ppp_set_asynch_params()

Set Asynchronous Link Parameters

Syntax

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

error_code
ppp_set_asynch_params(
        path_id         stack_path,
        ppp_modem_p     *params);
```

Libraries

`ppplib.l`

Description

This call sets the configurable parameter values for the asynchronous PPP link. The desired options are specified in the `ppp_modem_p` structure. This structure may be filled out completely by the user or selected items may be updated in a `ppp_modem_p` returned from a successful call to `ppp_get_asynch_params()`. The configuration must be set before the call to `ppp_connect()` or `ppp_start()` for the given path.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe

Parameters

`stack_path`	path identifier returned from a successful call to `ppp_open()`
`params`	pointer to the location that `ppp_set_asynch_params()`uses to update the asynchronous parameter values for the PPP link

Errors

`EOS_NOTRDY`	returned when the API has not been initialized
`EOS_BPNUM`	returned when this not a valid PPP stack path identifier
`EOS_ILLPRM`	illegal value was specified for one of the options

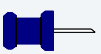
See Also

`ppp_open()`

`ppp_get_asynch_params()`

`ppp_connect()`

`ppp_modem_p structure`



Note

This call is valid only for asynchronous PPP links. The `rx_dev_name` and `tx_dev_name` fields of the PPP param structure should refer to the same device. The `rx_dev_name` field is used to specify the device that the PPP stack will use for communication. The `tx_dev_name` field is ignored.

ppp_set_options()

Set Negotiable Stack Options

Syntax

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

error_code
ppp_set_options(
        path_id             stack_path,
        ppp_option_block    *options);
```

Libraries

`ppplib.l`

Description

This call sets the negotiable options of the drivers in an open PPP stack. The options will be taken from the specified `ppp_option_block`. This structure can be filled out completely by the user or selected items can be modified in a `ppp_option_block` returned from a successful call to `ppp_get_options()`. The configuration must be set before the call to `ppp_connect()` or `ppp_start()` for the given path.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe

Parameters

`stack_path`	path identifier returned from a successful call to `ppp_open()`
`options`	pointer to the location where `ppp_set_options()` will take the new options for the PPP stack

Errors

EOS_NOTRDY	returned when the API has not been initialized
EOS_BPNUM	returned when this not a valid PPP stack path identifier
EOS_ILLPRM	illegal value was specified for one of the options

See Also

ppp_open()

ppp_get_options()

ppp_connect()

ppp_option_block structure

ppp_term()

Terminate the PPP API

Syntax

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

error_code
ppp_term(
     void    *rsvd);
```

Libraries

```
ppplib.l
```

Description

This call terminates (deinitializes) the use of this API. Currently, the single `rsvd` parameter is reserved for future use and must be NULL.

Attributes

Operating System:	OS-9
State:	User
Threads:	Safe

Parameters

`rsvd`	reserved for possible future use must be set to NULL

Errors

`EOS_NOTRDY`	returned when the API has not been initialized

See Also

`ppp_close()`

`ppp_connect()`

puthostent()

Add Network Host Entry

Syntax

```
#include <netdb.h>
error_code
puthostent(
      cons struct hostent    *hp)
```

Libraries

`netdb.l`

Description

`puthostent()` adds the host entry pointed to by `hp` to the `inetdb` data module or an expansion `inetdb` data module. There must be space available in the data module to hold the host entry.
`hp` is a pointer to a `hostent` structure allocated and set by the user. The `h_aliases` and `h_addr_list` elements are arrays of pointers to name and address information, and must be terminated by a null pointer. The `hostent` structure (defined in `/netdb.h`) is defined in the **Structures** section.

If successful, `puthostent()` returns `0`; otherwise it returns an error.

`puthostent()` supports a single IP address.

For More Information

See **Appendix B: Dynamic Configuration of the inetdb Data Module**.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`hp`	Specifies the host entry to be added.

Errors

`EOS_FULL`	No `inetdb` module found with enough space for the new entry.
`EOS_MNF`	Not linked to a `netdb` module.
`EOS_PARAM`	Bad parameter.
`EOS_TRAP`	`netdb` module not found.

See Also

`gethostent()`

putintent()

Add Interface Entry

Syntax

```
#include <netdb.h>
error_code putintent(
     n_ifnet      *ifp,
     n_ifaliasreq *ia,
     u_int32      ia_cnt)
```

Libraries

`netdb.l`

Description

`putintent()` adds an interface entry to the `inetdb` data module or an expansion `inetdb` data module. There must be space available in the data module to hold the entry. If successful, `putintent` returns `0`.

The `n_ifaliasreq` and `n_ifnet` structures (defined in `/netdb.h`) are described in **Structures** section.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`ifp`	Specifies a pointer to the user allocated and set interface structure.
`ia`	Specifies an array of null-terminated addresses to be associated with the interface.
`ia_cnt`	Specifies the number of addresses contained in the `ia` array, and may be `0`.

For More Information

See **Appendix B: Dynamic Configuration of the inetdb Data Module** for example code.

Errors

`EOS_MNF`	`inetdb` module not found.
`EOS_FULL`	No `inetdb` module found with enough space for the new entry.

See Also

`getnetent()`

.

putnetent()

Add a network entry to inetdb

Syntax

```
#include <netdb.h>
putnetent(const struct netent *np);
```

Libraries

netdb.l

Description

`putnetent()` adds a network entry to the `inetdb` data module or an expansion `inetdb` data module. There must be space available in the data module to hold the entry. If successful, `putnetent` returns 0.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

np	specifies a pointer to the user allocated and initialized network entry structure

Errors

`EOS_MNF`	`inetdb` module not found.
`EOS_PARAM`	Bad parameter.
`EOS_FULL`	No `inetdb` module found with enough space for the new entry.

putprotoent()

Add protocol entry to inetdb

Syntax

```
#include <netdb.h>
putprotoent(const struct protoent *pep);
```

Libraries

netdb.l

Description

`putprotoent()` adds a protocol entry to the `inetdb` data module or an expansion `inetdb` data module. There must be space available in the data module to hold the entry. If successful, it returns 0.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`pep`	specifies a pointer to the user allocated and initialized protocol entry structure

Errors

`EOS_MNF`	`inetdb` module not found.
`EOS_PARAM`	Bad parameter.
`EOS_FULL`	No `inetdb` module found with enough space for the new entry.

putresolvent()

Set the DNS Entry

Syntax

```
#include <resolv.h>
error_code putresolvent(resolvent *res)
```

Libraries

`netdb.l`

Description

`putresolvent()` adds the DNS resolver entry pointed to by `res` to the `inetdb` data module or an expansion `inetdb` data module. There must be space available in the data module to hold the resolver entry.

If successful, `putresolvent()` returns zero.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`res`	A pointer to a `resolvent` structure allocated and set by the user. The name server and search pointer arrays must be terminated by a null pointer. There can be only one active resolver entry. Only the first entry of the highest numbered `inetdbx` module is used. If you have enough space, you can add more than one, but it is ignored. To be safe, use `delresolvent`.

For More Information

See **Appendix B: Dynamic Configuration of the inetdb Data Module** for example code.

Errors

EOS_FULL	No `inetdb` module found with enough space for the new entry.
EOS_MNF	`inetdb` module not found.
EOS_PARAM	Bad parameter.
EOS_TRAP	`netdb` module not found.

See Also

`delresolvent()`
`endresolvent()`
`getresolvent()`

putroutent()

Add Route Entry

Syntax

```
#include <netdb.h>
error_code *putroutent(struct rtreq *route_ptr)
```

Libraries

`netdb.l`

Description

`putroutent()` adds the route entry to the `inetdb` data module. If successful, `putroutent()` returns 0.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Errors

`EOS_MNF`	`inetdb` module not found.
`EOS_PARAM`	Bad parameter.
`EOS_FULL`	No `inetdb` module found with enough space for the new entry.

See Also

For More Information

See **Appendix B: Dynamic Configuration of the inetdb Data Module** for sample code.

putservent()

Add interface entry to inetdb

Syntax

```
#include <netdb.h>
putservent(const struct servent *sp)
```

Libraries

netdb.l

Description

`putservent()` adds an interface entry to the `inetdb` data module or an expansion `inetdb` data module. There must be space available in the data module to hold the entry. If successful, it returns 0.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`sp`	specifies a pointer to the user allocated and initialized interfrace entry structure.

Errors

`EOS_MNF`	`inetdb` module not found.
`EOS_PARAM`	Bad parameter.
`EOS_FULL`	No `inetdb` module found with enough space for the new entry.

recv()

Receives Message From Connected Socket

Syntax

```
#include <sys/socket.h>
int recv(
     int          s,
     char         *buf,
     int          len,
     int          flags);
```

Libraries

`socket.l`

Description

`recv()` receives messages from a socket. Use `recv()` only on connected sockets.

If no data are available at the socket, the call waits for data to arrive, unless the socket is non-blocking. In this case, `-1` is returned with the external variable `errno` set to `EWOULDBLOCK`.

On success, the number of bytes read is returned. This value may be less than the number of bytes requested in `len`. If the socket is of type `SOCK_STREAM`, a return value of 0 indicates the peer has closed its half of the connection and no more data is available to read. On error, `-1` is returned and `errno` is set to the error value.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`s`	Specifies the path number of the socket.
`buf`	Points to the buffer into which the message is received.
`len`	Specifies the length of the buffer.
`flags`	`MSG_PEEK`; `MSG_WAITALL`.

Errors

`ENOTCONN`	The socket is not connected.
`EWOULDBLOCK`	The socket is marked non-blocking and the receive operation would block.

See Also

`connect()`
`recvfrom()`
`send()`
`sendto()`
`socket()`

recvfrom()

Receives Message from Socket

Syntax

```
#include <sys/socket.h>
int recvfrom(
     int          s,
     char         *buf,
     int          len,
     int          flags,
     struct sockaddr *from,
     int          *fromlen);
```

Libraries

```
socket.l
```

Description

`recvfrom()` receives messages from a socket. You may use `recvfrom()` to receive data on a socket in an unconnected state.

`recvfrom()` returns the length of the incoming message. If a message is too long to fit in the supplied buffer, excess bytes may be discarded depending on the type of socket from which the message is received.

If no data are available at the socket, the call waits for data to arrive, unless the socket is non-blocking. In this case, `-1` is returned with the external variable `errno` set to `EWOULDBLOCK`.

On success the number of bytes read is returned. This value may be less than the number of bytes requested in `len`. If the socket is of type `SOCK_STREAM`, a return value of 0 indicates the peer has closed its half of the connection and no more data is available to read. On error, `-1` is returned and `errno` is set to the error value.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`s`	Specifies the path number of the socket.
`buf`	Points to the buffer into which the message is received.
`len`	Specifies the length of the buffer.
`flags`	`MSG_PEEK`; `MSG_WAITALL`.
`from`	Points to a buffer specifying the sender of the message. If `from` is non-zero, the source address of the message is filled in.
`fromlen`	Initialized to the size of the buffer associated with `from`. `fromlen` is modified on return to indicate the actual size of the address stored.

Errors

`EWOULDBLOCK`	The socket is marked non-blocking and the receive operation is blocked.

See Also

`recv()`
`send()`
`sendto()`
`socket()`

res_cancel()

Cancel DNS Client Request

Syntax

```
#include <netdb.h>
void res_cancel(void)
```

Libraries

`netdb.l`

Description

`res_cancel()` sets a flag to cancel a DNS client query (`gethostbyname()`). This stops an application from blocking on a `gethostbyname()` call. `res_cancel()` can only be used from the `netdb_dns.l` library since the trap library is not reentrant.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

send()

Sends Message to Connected Socket

Syntax

```
#include <sys/socket.h>
int send(
     int          s,
     void         *msg,
     int          len,
     int          flags);
```

Libraries

`socket.l`

Description

`send()` transmits a message to another socket. Use `send()` only when the socket is in a connected state.

If the socket is a datagram socket and the message is too long to pass atomically through the underlying protocol, an error is returned and the message is not transmitted.

For reliable protocols, such as TCP, if no message space is available at the socket to hold the transmitted message, `send()` normally blocks, unless the socket has been placed in non-blocking I/O mode.

No indication of failure to deliver is implicit in a send. Return values of `-1` indicate some locally detected errors. On success, `send()` returns the number of characters sent. On error, `-1` is returned and `errno` is set to the error value.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`s`	Specifies the path number of a socket created with `socket()`.
`msg`	Points to the message to send.
`len`	Specifies the length of the message.
`flags`	Not supported. Set `flags` to `0`.

Errors

`EMSGSIZE`	The message is too long.
`EHOSTUNREACH`	No route to host.
`ENOBUFS`	Driver cannot allocate buffer.
`ENOTCONN`	Packets are sent through a non-established socket. The socket must be connected.
`EWOULDBLOCK`	The socket is marked non-blocking and the requested operation would block.

See Also

`recv()`
`recvfrom()`
`sendto()`
`socket()`

sendto()

Sends Message to Socket

Syntax

```
#include <sys/socket.h>
int sendto(
     int          s,
     void         *msg,
     int          len,
     int          flags,
     struct sockaddr *to,
     int          tolen);
```

Libraries

`socket.l`

Description

`sendto()` transmits a message to another socket. You may use the function with unconnected sockets.

For reliable protocols, such as TCP, if no message space is available at the socket to hold the message to transmit, `send()` normally blocks, unless the socket has been placed in non-blocking I/O mode.

No indication of failure to deliver is implicit in a send. Return values of `-1` indicate some locally detected errors. On success, `sendto()` returns the number of characters sent. On error, `-1` is returned and `errno` is set to the error value.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`s`	Specifies the path number of a socket created with `socket()`.
`buf`	Points to the message to send.
`len`	Specifies the length of the message.
`flags`	Not supported. Set `flags` to `0`.
`to`	Points to the address of the target.
`tolen`	Specifies the size of the buffer associated with `to`.

Errors

`EDSTADDRREQ`	The `to` address pointer must be non-null and `tolen` variable must be non-zero.
`EMSGSIZE`	The message is too long.
`EHOSTUNREACH`	No route to host.
`ENOBUFS`	Driver cannot allocate buffer.
`ENOTCONN`	Packets are sent through a non-established socket.
`EWOULDBLOCK`	The socket is marked non-blocking and the requested operation would block.

See Also

`recvfrom()`
`send()`
`socket()`

sethostent()

Set Host Entry

Syntax

```
#include <netdb.h>
int sethostent(int stayopen)
```

Libraries

`netdb.l`

Description

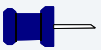
`sethostent()` links the calling process to `inetdb`, if necessary, and resets the pointer to the beginning of the `hosts` entries.

If successful, `sethostent()` returns a value of `0`.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters



Note

LAN Communications Pak ignores the `stayopen` flag. It is included for compatibility only.

Errors

`ENOMEM`	Insufficient RAM (POSIX).
`EOS_MNF`	`inetdb` module could not be found.

See Also

endhostent()
gethostbyaddr()
gethostbyname()
gethostent()
gethostname()
sethostname()

sethostname()

Set Name of Current Host

Syntax

```
#include <sys/socket.h>
int sethostname(
     char         *name,
     int          namelen)
```

Libraries

`socket.l`

Description

`sethostname()` sets the host name for the current host.

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

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`name`	A pointer to a null-terminated string containing the new standard host name for the current processor.
`namelen`	Specifies the size of the name array.

Errors

`EOS_MNF`	`inetdb` module could not be found.

See Also

endnetent()
getnetbyaddr()
getnetbyname()
getnetent()

setnetent()

Set Network Entry

Syntax

```
#include <netdb.h>
int setnetent(int stayopen)
```

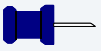
Libraries

`netdb.l`

Description

`setnetent()` links the calling process to `inetdb`, if necessary, and resets the pointer to the beginning of the network entries.

If successful, `setnetent()` returns a value of `0`. Otherwise, it returns `-1` and sets `errno` to the error value.



Note

LAN Communications Pak ignores the `stayopen` flag. It is included for compatibility only.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Errors

`EOS_MNF`	`inetdb` module could not be found.

See Also

`endnetent()`
`getnetent()`

setprotoent()

Set Protocol Entry

Syntax

```
#include <netdb.h>
int setprotoent(int stayopen)
```

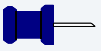
Libraries

`netdb.l`

Description

`setprotoent()` links the calling process to `inetdb`, if necessary, and resets the pointer to the beginning of the `protocol` entries.

If successful, `setprotoent()` returns a value of `0`. Otherwise, it returns `-1` and sets `errno` to the error value.



Note

LAN Communications Pak ignores the `stayopen` flag. It is included for compatibility only.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Errors

`EOS_MNF`	`inetdb` module could not be found.

See Also

`getprotobyname()`
`getprotobynumber()`
`getprotoent()`

setservent()

Set Services Entry

Syntax

```
#include <netdb.h>
int setservent(int stayopen)
```

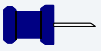
Libraries

`netdb.l`

Description

`setservent()` links the calling process to `inetdb`, if necessary, and resets the pointer to the beginning of the `services` entries.

If successful, `setservent()` returns a value of `0`. Otherwise, it returns `-1` and sets `errno` to the error value.



Note

LAN Communications Pak ignores the `stayopen` flag. It is included for compatibility only.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Errors

`EOS_MNF`	`inetdb` module could not be found.

See Also

endservent()
getservbyname()
getservbyport()
getservent()

setsockopt()

Set Options on Sockets

Syntax

```
#include <sys/socket.h>
int setsockopt(
     int          s,
     int          level,
     int          optname,
     void         *optval,
     int          optlen)
```

Libraries

`socket.l`

Description

`setsockopt()` sets options associated with a socket. Options may exist at multiple protocol levels; they are always present at the uppermost socket level.

`setsockopt()` returns `0` if the call succeeds. Otherwise, it returns `-1` and sets `errno` to the error value.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Options

The include file `sys/socket.h` contains definitions for socket level options. Options at other protocol levels vary in format and name and are defined in the protocol's header file.

Table 1-10 Options

Level	optname	Description	Data Type
IPPROTO_IP	IP_ADD_MEMBERSHIP	Join multicast group	ip_mreq{}
IPPROTO_IP	IP_DROP_MEMBERSHIP	Leave multicast group	ip_mreq{}
IPPROTO_IP	IP_MULTICAST_TTL	Set TTL for multicast packets	u_char
IPPROTO_IP	IP_MULTICAST_LOOP	Send multicast packets to the loopback interface	u_char
IPPROTO_IP	IP_MULTICAST_IF	Select interface for sending multicast packets	in_addr{}
IPPROTO_TCP	TCP_MAXSEG	Set maximum segment size	int
IPPROTO_TCP	TCP_NODELAY	Do not delay send to coalesce packets	int
SOL_SOCKET	SO_KEEPALIVE	Keep connection alive by forcing peer to respond periodically	int
SOL_SOCKET	SO_LINGER	Linger on close if data present	linger{}
SOL_SOCKET	SO_REUSEADDR	Allow local address reuse	int
SOL_SOCKET	SO_BROADCAST	Permit sending broadcast datagrams	int

Table 1-10 Options (continued)

Level	optname	Description	Data Type
SOL_SOCKET	SO_OOBINLINE	Leave out-of-band data in normal input queue	int
SOL_SOCKET	SO_SNDBUF	Set size of send buffer	int
SOL_SOCKET	SO_RCVBUF	Set size of receive buffer	int
SOL_SOCKET	SO_SNDLOWAT	Minimum space required in send buffer to accept more data	int
SOL_SOCKET	SO_USELOOPBACK	Routing socket receives copy of any data sent (AF_ROUTE only)	int

Parameters

`s` Specifies the path number of the socket.

`level` Specifies where the option resides.

- To set options at the socket level, specify `level` as `SOL_SOCKET`.
- To set options at any other level, supply the protocol number of the appropriate protocol controlling the option. For example, `IPPROTO_TCP` specifies TCP options.

`optname` The name of the option to be set.

`optval` The new value of the option.

`optlen` The size of `optval` in bytes.

Errors

ENOPROTOOPT	The option is unknown.
EFAULT	The address pointed to by `optval` is not in a valid part of the process address space.

See Also

`getsockopt()`
`socket()`

shutdown()

Shut Down Part of Full-Duplex Connection

Syntax

```
#include <sys/socket.h>
int shutdown(
     int          s,
     int          how)
```

Libraries

`socket.l`

Description

`shutdown()` shuts down all or part of a full-duplex connection of the socket specified by `s`.

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

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`s`	Specifies the path number of the socket.
`how`	Specifies the method for receiving and sending permissions. • If `how` is `0`, further receives are disallowed. • If `how` is `1`, further sends are disallowed. • If `how` is `2`, further sends and receives are disallowed.

Errors

ENOTCONN	The specified socket is not connected.

See Also

connect()
socket()

socket()

Creates Endpoint for Communication

Syntax

```
#include <sys/types.h>
#include <sys/socket.h>
int socket(
     int          af,
     int          type,
     int          protocol)
```

Libraries

`socket.l`

Description

`socket()` creates an endpoint for communication and returns a path number.

If successful, `socket()` returns the descriptor referencing the socket. Otherwise, it returns `-1` and places the appropriate error code in the global variable `errno`.

SOCK_STREAM Sockets

Sockets of type `SOCK_STREAM` are sequenced, reliable, and full-duplex byte streams, similar to pipes. A stream socket must be in a connected state before any data may be sent or received on it. A `connect()` call creates a connection to another socket.

Once connected, data may be transferred using `_os_read()` and `_os_write()` calls or some variant of the `send()` and `recv()` calls.

SOCK_DGRAM Sockets

SOCK_DGRAM sockets provide an unreliable datagram service. The socket may be either connected or unconnected. If it is connected, data may be sent and received using `read()`, `write()`, `_os_read()`, `_os_write()`, `send()` or `recv()`. If the socket is unconnected, `sendto()` and `recvfrom()` must be used.

SOCK_RAW Sockets

SOCK_RAW allows you to build datagrams, including headers. It is used to send and receive ICMP messages, internal routing requests, and user-defined protocols. Sockets of this type require super-user privileges to create.

Attributes

Operating System:	OS-9 and OS-9 for 68K
State:	User
Threads:	Safe

Parameters

`af` — Specifies an address format with which addresses specified in later operations using the socket should be interpreted. These formats are defined in the include file `socket.h`. Accepted format is:

AF_INET	ARPA Internet address family.
AF_ROUTE	Internal Routing Protocol

`type`	Specifies the semantics of communication. Type values are:
	`SOCK_STREAM` Provides sequenced, reliable, two-way connection based byte streams with an out-of-band data transmission mechanism.
	`SOCK_DGRAM` Supports datagrams (connectionless, unreliable messages of a fixed and typically small maximum length).
	`SOCK_RAW` Supports datagrams (connectionless, unreliable messages of a fixed and typically small maximum length).
`protocol`	Specifies a particular protocol to use with the socket. Normally only a single protocol exists to support a particular socket type, using a given address format. However, many protocols may exist, in which case a particular protocol must be specified in this manner. The protocol number is particular to the communication domain in which communication is to take place.

Errors

`EAFNOSUPPORT`	The specified address family is not supported in this version of the system.
`EPROTONOSUPPORT`	Specified protocol is not supported.
`ENOBUFS`	There is no end-buffer space available. The socket cannot be created.

See Also

accept()
bind()
connect()
getsockname()
getsockopt()
listen()
recv()
send()
shutdown()

Chapter 2: Error Messages

The following messages are extensions to the existing system messages and can be returned by socket access to the internet software. These messages are defined in the `errno.h` header file.

OS-9 Messages

For OS-9 for 68K, the indicated message number is constructed by separating the decimal representation of the upper and lower bytes of the error codes with a colon. For example, message number 007:001 corresponds to a hexadecimal value of 0x0701.

For OS-9, the indicated error number is constructed by separating the decimal representation of the upper and lower words of the error codes with a colon. For example, message number 007:001 corresponds to a hexadecimal value of 0x00070001.

Table 2-1 Messages

Message Number	Description
007:001	**EWOULDBLOCK (I/O operation would block)** An operation that would cause a process to block attempted on a socket in non-blocking mode.
007:002	**EINPROGRESS (I/O operation now in progress)** An operation taking a long time to complete (such as `connect()`) attempted on a socket in non-blocking mode.
007:003	**EALREADY (Operation already in progress)** An operation was attempted on a non-blocking object with an operation in progress.
007:004	**EDESTADDRREQ (Destination address required)** The attempted socket operation requires a destination address.

Table 2-1 Messages (continued)

Message Number	Description
007:005	**EMSGSIZE (Message too long)** A message sent on a socket is larger than the internal message buffer or some other network limit. Messages must be smaller than 32768 bytes.
007:006	**EPROTOTYPE (Protocol wrong type for socket)** A protocol is specified that does not support the semantics of the socket type requested. For example, an AF_INET UDP protocol as SOCK_STREAM is the wrong protocol type for the socket.
007:007	**ENOPROTOOPT (Bad protocol option)** A bad option or level is specified in getsockopt() or setsockopt().
007:008	**EPROTONOSUPPORT (Protocol not supported)** The requested protocol is not available or not configured for use.
007:009	**ESOCKNOSUPPORT (Socket type not supported)** The requested socket type is not supported or not configured for use.
007:010	**EOPNOTSUPP (Operation not supported on socket)** For example, accept() on a datagram socket.
007:011	**EPFNOSUPPORT (Protocol family not supported)**
007:012	**EAFNOSUPPORT (Address family not supported by protocol)**

Table 2-1 Messages (continued)

Message Number	Description
007:013	**EADDRINUSE (Address already in use)** Only one use of each address is normally permitted. Wildcard use and connectionless communication are the exceptions.
007:014	**EADDRNOTAVAIL (Can't assign requested address)** Results from an attempt to create a socket with an address not on this machine.
007:015	**ENETDOWN (Network is down)** The network hardware is not accessible.
007:016	**ENETUNREACH (Network is unreachable)** The network is unreachable. Usually caused by network interface hardware that is operational, but not physically connected to the network. This error can also be caused when the network has no way to reach the destination address.
007:017	**ENETRESET (Network dropped connection on reset)** The host you were connected to crashed and rebooted.
007:018	**ECONNABORTED (Software caused connection abort)** A connection abort was caused by the local (host) machine.
007:019	**ECONNRESET (Connection reset by peer)** A peer forcibly closed a connection. This normally results from a loss of the connection on the remote socket due to a time out or reboot.

Table 2-1 Messages (continued)

Message Number	Description
007:020	**ENOBUFS (No buffer space available)** A socket operation could not be performed because the system lacked sufficient buffer space or a queue is full.
007:021	**EISCONN (Socket is already connected)** A `connect()` request was made on an already connected socket. Also caused by a `sendto()` request on a connected socket to a destination which is already connected.
007:022	**ENOTCONN (Socket is not connected)** A request to send or receive data is rejected because the socket is not connected or no destination is given with a datagram socket.
007:023	**ESHUTDOWN (Can't send after socket shutdown)**
007:024	**ETOOMANYREFS (Too many references)**
007:025	**ETIMEOUT (Connection timed out)** A `connect()` or `send()` request failed because the connected peer did not properly respond after a period of time. The time out period depends on the protocol used.
007:026	**ECONNREFUSED (Connection refused by target)** No connection could be established because the target machine actively refused it. This usually results from trying to connect to a service that is inactive on the target host.
007:027	**EBUFTOOSMALL (Mbuf too small for mbuf operation)**
007:028	**ESMODEXISTS (Socket module already attached)**

Table 2-1 Messages (continued)

Message Number	Description
007:029	**ENOTSOCK (Path is not a socket)**
007:030	**EHOSTUNREACH (Host is unreachable; route not found)**
007:031	**EHOSTDOWN (Host is down)**

Appendix A: Example Programs

This appendix contains a TCP and UDP socket example. Each example includes a client program and a server program. You may use these programs as templates for writing your own programs.

Source code for these examples resides in the following directory: `MWOS/SRC/SPF/INET/EXAMPLES`

The following sections are included in this appendix:

Example One: Datagram Socket Operation

The next two programs, `beam.c` and `target.c`, transfer data using UDP. The beam program sends a user-specified number of 1000-byte datagrams to a particular machine. By default, these are sent to UDP port 20000 but a new port may be specified using the optional third parameter to `beam`.

The `target` program binds to UDP port 20000 (or another port if specified on the command line) and receives datagrams. At the end of the transfer the number of packets and bytes received is printed. Because UDP does not guarantee end-to-end reliability of data delivery the number of packets `target` receives may be less than the number sent by `beam`.

Source code resides in the following directory:

`MWOS/SRC/SPF/INET/EXAMPLES/AF_INET.UDP`

beam.c

```
/* <<<<<<<<<<<<<<<<<<<<<< beam.c >>>>>>>>>>>>>>>>>>>> */
/*
** Syntax:    beam <target> <count> [<port>]
** Function:  sends packets over a datagram socket
*/

/* Header Files */
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <modes.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

/* Macro Definitions */
#define PKT_SIZE    1000   /* packet size */
#define PORT        20000     /* udp port number */
#define START       1         /* packet types */
#define NORMAL      2
```

```
#define END          3

/* Type Definitions */
struct packet {
    u_int32 type;
    u_int32 size;
    u_int32 count;
    char   buf[PKT_SIZE - 12];
} packet, *Packet;

void main(int argc, char* argv[], char* envp[])
{
    int s;
    int i;
    int count;
    struct hostent *host;
    struct sockaddr_in sockname;
    static struct packet pkt;
    /* check for proper number of arguments */
    if ((argc < 3) || (argv[1][0] == '-')) {
        printf("usage: beam <hostname|ip-address> <count> [<port>]\n");
        exit(0);
    }
    /* get number of packets to beam */
    count = atoi(argv[2]);
    /* open up datagram (UDP) socket */
    memset(&sockname, 0, sizeof(sockname));
    sockname.sin_family = AF_INET;
    sockname.sin_port = 0;
    sockname.sin_addr.s_addr = INADDR_ANY;
    if ((s = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
        fprintf(stderr, "beam: socket call failed\n");
        exit(errno);
    }
    /* bind socket (let system pick our port number) */
    if (bind(s, (struct sockaddr*)&sockname, sizeof(sockname)) < 0) {
        fprintf(stderr, "bind failed to host\n");
        _os_close(s);
        exit(errno);
    }
    /* get information concerning the host we'd like to beam to */
    sockname.sin_port = 0;
    sockname.sin_addr.s_addr = INADDR_ANY;
    if ((host = gethostbyname(argv[1])) != (struct hostent *)0) {
        sockname.sin_family = host->h_addrtype;
        memcpy(&sockname.sin_addr.s_addr, host->h_addr, 4);
    } else {
        u_int32 addr = inet_addr(argv[1]);
        sockname.sin_family = AF_INET;
```

```
        memcpy(&sockname.sin_addr.s_addr, &addr, 4);
    }
    endhostent();
    if (argc > 3) {
        sockname.sin_port = htons(atoi(argv[3]));
    } else {
        sockname.sin_port = htons(PORT);
    }
    /* set up socket for send */
    #ifdef USE_CONNECT
        /* connected UDP socket -- we're only talking to this host */
        if (connect(s, (struct sockaddr *)&sockname, sizeof(sockname))
                < 0) {
            fprintf(stderr, "beam: cannot connect\n");
            _os_close(s);
            exit(errno);
        }
    #endif
    printf("beaming...\n");
    /* set up packets for transfer and transfer them all */
    pkt.size = htonl(PKT_SIZE);
    for (i = 0; i <= count; i++) {
        if (i == 0) {
            pkt.type = htonl(START);
        } else if (i >= count) {
            pkt.type = htonl(END);
        } else {
            pkt.type = htonl(NORMAL);
        }
        pkt.count = htonl(i);
        /* send data to target */
        #ifdef USE_CONNECT
            if (send(s, &pkt, ntohl(pkt.size), 0) < 0) {
                fprintf(stderr, "beam: send failed\n");
                _os_close(s);
                exit(errno);
            }
        #else
            if (sendto(s, (char*)&pkt, ntohl(pkt.size), 0,
                    (struct sockaddr*)&sockname, sizeof(sockname)) < 0) {
                fprintf(stderr, "beam: sendto failed\n");
                _os_close(s);
                exit(errno);
            }
        #endif
    }
    _os_close(s);
    exit(0);
}   /* end of main */
```

target.c

```
/* <<<<<<<<<<<<<<<<<<<<<< target.c >>>>>>>>>>>>>>>>>>> */
/*
** Syntax:      target
** Function:    receives packets from beam over a datagram
**              socket and displays number of packets
*/

/* Header Files */
#include <stdio.h>
#include <stdlib.h>
#include <types.h>
#include <errno.h>
#include <string.h>
#include <modes.h>
#include <ctype.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

/* Macro Definitions */
#define PKT_SIZE     1000     /* packet size */
#define PORT         20000    /* udp port number */
#define START        1        /* packet types */
#define NORMAL       2
#define END          3

/* Type Definitions */
struct packet {
    u_int32      type;
    u_int32      size;
    u_int32      count;
    char         buf[PKT_SIZE - 12];
} packet, *Packet;

/* main() : initial program entry point */
void main(int argc, char* argv[], char* envp[])
{
    int s;
    int count;
    int namelen;
    int packetsrecv;
    u_int32 bytesrecv;
    static struct packet pkt;
    struct sockaddr_in name;
```

```
if ((argc < 1) || (argc > 2) || ((argc == 2) &&
        (!isdigit(argv[1][0])))) {
    printf("usage: target [<port>]\n");
    exit(0);
}
/* open up datagram (UDP) socket */
if ((s = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
    fprintf(stderr, "target: socket failed\n");
    exit(errno);
}
/* bind socket (pick proper "well-known" port number) */
memset(&name, 0, sizeof(name));
name.sin_family = AF_INET;
name.sin_addr.s_addr = INADDR_ANY;
if (argc >= 2) {
    name.sin_port = htons(atoi(argv[1]));
} else {
    name.sin_port = htons(PORT);
}
if (bind(s, (struct sockaddr*)&name, sizeof(name)) == -1) {
    fprintf(stderr, "target: bind failed to port
       '%d'\n",ntohs(name.sin_port));
    _os_close(s);
    exit(errno);
}
printf("Waiting for packets...\n");
/* wait for start packet */
while (1) {
    /* get a packet (and find out who sent it to us) */
    namelen = sizeof(name);
    if ((count = recvfrom(s, (char*)&pkt, sizeof(pkt), 0,
                        (struct sockaddr*)&name,&namelen)) == -1) {
       fprintf(stderr, "target: recv failed\n");
       _os_close(s);
       exit(errno);
    }
    if (pkt.type != htonl(START)){
       printf("out of sequence packet received\n");
       continue;
    } else {
       break;
    }
}
bytesrecv = packetsrecv = 0;
/* loop until all packet are receved */
printf("Begin transfer\n");
do {
    namelen = sizeof(name);
    if ((count = recvfrom(s, (char*)&pkt, sizeof(pkt), 0,
```

```
                        (struct sockaddr*)&name,&namelen)) == -1) {
            fprintf(stderr, "target: recv failed\n");
            _os_close(s);
            exit(errno);
         }
         bytesrecv += count;
         packetsrecv++;
      } while (pkt.type == ntohl(NORMAL));
      /* if we didn't get and END packet, print error */
      if (pkt.type != ntohl(END)) {
         printf("expected an END packet\n");
      }
      /* print out summary */
      printf("Transfer complete\n");
      printf("    Packets received:  %d\n",packetsrecv);
      printf("    Bytes received:  %d\n",bytesrecv);
      /* cleanup and exit */
      _os_close(s);
      exit(0);
}  /* end of main */
```

Example Two: Stream Socket Operation

The following programs, `tcpsend.c` and `tcprecv.c` use TCP to reliably transfer a file. The `tcpsend` program connects to TCP port 27000 on the target machine and sends the specified file. Normally `tcpsend` uses blocking I/O. However, if an optional third parameter is used the socket is set to nonblocking.

The `tcprecv` program binds a socket to port 27000 and listens for incoming data connections. When a connection occurs the data is read from the network and written to the output file specified on the command line.

Source code resides in the following directory:

`MWOS/SRC/SPF/INET/EXAMPLES/AF_INET.TCP`

tcpsend.c

```
/* <<<<<<<<<<<<<<<<<<<<<<<<<<<< tcpsend.c >>>>>>>>>>>>>>>>>>>>>>>>>>>> */
/*
**  Syntax:    tcpsend <hostname> <file> <blockflag>
**  Function:  sends a file across a stream (TCP) socket
*/

/* Header Files */
#include <stdio.h>
#include <stdlib.h>
#include <const.h>
#include <signal.h>
#include <types.h>
#include <errno.h>
#include <string.h>
#include <modes.h>
#include <sg_codes.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/errno.h>
#include <netinet/in.h>
#include <netdb.h>
#include <SPF/spf.h>
#include <UNIX/ioctl.h>

/* Macro Definitions */
#define PORT_NUM 27000
#define MAX_LOOPS 200
```

```
#if defined(_OSK)
    #define FMODE(S_IREAD)
    #define _os_sleep(t,s) _os9_sleep(t)
#else
    #define FMODE(S_IREAD|S_IGREAD)
    signal_code sig = 0;
#endif
/* Type Definitions */
struct data {
    int code, count;
    char data[512];
};

/* Global Variables */
struct sockaddr_in ls_addr;
char msgbuf[20480];
char *ptr;

/* Function Prototypes */
void main(int argc, char* argv[], char* envp[]);
int ioctl(unsigned int, unsigned int, caddr_t);

void main(int argc, char* argv[], char* envp[])
{
    int s;
    int totbytes = 0;
    int noblock = 0;
    path_id ifile;
    u_int32 count;
    u_int32 wcount;
    u_int32 wsize;
    u_int32 tries;
    u_int32 tics;
    struct hostent *host;
    if ((argc <= 1) || (argc > 4) || (argv[1][0] == '-')) {
        fprintf(stderr, "tcpsend <hostname> <file> [nonblocking]\n");
        exit(0);
    }
    memset(&ls_addr, 0, sizeof(ls_addr));
    if ((ls_addr.sin_addr.s_addr = inet_addr(argv[1])) == (u_long)-1){
        if ((host = gethostbyname(argv[1])) != NULL){
            memcpy(&ls_addr.sin_addr.s_addr, host->h_addr,
                                    host->h_length);
            ls_addr.sin_family = host->h_addrtype;
        } else {
            fprintf(stderr, "can't resolve name '%s'\n", argv[1]);
            exit(errno);
        }
        endhostent();
    } else {
        ls_addr.sin_family = AF_INET;
    }
    ls_addr.sin_port =  htons(PORT_NUM);
    if ((errno = _os_open(argv[2],FMODE,&ifile)) != SUCCESS) {
```

```
    fprintf(stderr, "can't open file '%s'\n", argv[2]);
    exit(errno);
}
if ((s = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
    perror("socket");
    _os_close(ifile);
    exit(1);
}
if (argc >= 4) {
    printf("using non-blocking sockets\n");
    noblock = IO_ASYNC;
    if (ioctl(s, FIONBIO, (caddr_t)&noblock)) {
        fprintf(stderr, "can't set socket nonblocking\n");
        _os_close(ifile);
        _os_close(s);
        exit(errno);
    }
} else {
    printf("using blocking sockets\n");
}
if (noblock) {
    /*
    ** Non-blocking connect
    */
    tries = MAX_LOOPS;
    while (tries) {
        if (connect(s, (struct sockaddr*)&ls_addr,
                                sizeof(ls_addr)) == -1) {
            if (errno == EISCONN) {
                break;
            }
            if (errno == EINVAL) {
                int error, len;
                len = sizeof(error);
                if (getsockopt(s, SOL_SOCKET, SO_ERROR, &error,
                                &len) < 0) {
                    error = EINVAL;
                }
                errno = error;
                perror("connect");
                _os_close(s);
                _os_close(ifile);
                exit(1);
            }
            tics = 10;
            _os_sleep(&tics, &sig);
        } else {
            break;
        }
        tries--;
    }

    if (tries == 0){
        errno = ETIMEDOUT;
```

```
        perror("connect");
        _os_close(s);
        _os_close(ifile);
        exit(1);
    }
} else {
    /*
    ** Blocking connect
    */
    if (connect(s, (struct sockaddr *)&ls_addr,
                                sizeof(ls_addr)) == -1) {
        perror("connect");
        _os_close(s);
        _os_close(ifile);
        exit(1);
    }
}
printf("Connection established\n");
printf("Sending file '%s'...\n", *argv);
if (noblock) {
    /*
    ** Non-blocking send
    */
    count = sizeof(msgbuf);
    while ((errno = _os_read(ifile,msgbuf,&count)) == SUCCESS) {
        wcount = 0;
        ptr = msgbuf;
        while (wcount < count){
            wsize = count - wcount;
            if ((errno = _os_write(s,ptr,&wsize)) != SUCCESS) {
                if (errno != EWOULDBLOCK) {
                    fprintf(stderr, "socket write error\n");
                    _os_close(s);
                    _os_close(ifile);
                    exit(errno);
                } else {
                    tics = 10;
                    _os_sleep(&tics,&sig);
                }
            } else {
                wcount += wsize;
                ptr += wsize;
            }
        }
        totbytes += count;
        count = sizeof(msgbuf);
    }
    if (errno != EOS_EOF){
        fprintf(stderr, "read error on file\n");
        exit(errno);
    }
} else {
    /*
    ** Blocking send
```

```
        */
        count = sizeof(msgbuf);
        while ((errno = _os_read(ifile, msgbuf, &count)) == SUCCESS) {
            wcount = count;
            ptr = msgbuf;
            while ((errno = _os_write(s, ptr, &count)) == SUCCESS){
                totbytes += count;
                if (count == wcount) {
                    break;
                }
                /*
                ** For some reason all the data was not written,
                ** loop around and try to write the rest.
                */
                ptr += count;
                count = wcount - count;
                wcount = count;
            }
            if (errno){
                fprintf(stderr, "socket write error\n");
                _os_close(s);
                _os_close(ifile);
                exit(errno);
            }
            count = sizeof(msgbuf);
        }
        if (errno != EOS_EOF){
            fprintf(stderr, "read error on file\n");
            exit(errno);
        }
    }
    _os_close(s);
    _os_close(ifile);
    printf("sent %u bytes\n", totbytes);
    exit(0);
}
```

tcprecv.c

```
/* <<<<<<<<<<<<<<<<<<<<<<<<<< tcprecv.c >>>>>>>>>>>>>>>>>>>>>>>> */
/*
**  Syntax:    tcprecv <file>
**  Function:  receives a file across a stream (TCP) socket
*/

/* Header Files */
#include <stdio.h>
#include <stdlib.h>
#include <const.h>
#include <types.h>
#include <errno.h>
```

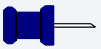
```
#include <string.h>
#include <modes.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

/* Macro Definitions */
#define PORT_NUM 27000
#define INIT 77/* commands */
#define DATA 78
#define END 79
#if defined(_OSK)
    #define FMODE(S_IWRITE)
#elif defined(_OS9000)
    #define FMODE(S_IWRITE|S_IGWRITE)
#endif

/* Global Variables */
char msgbuf[20480];

/* Function Prototypes */
void main(int argc, char* argv[], char* envp[]);

void main(int argc, char* argv[], char* envp[])
{
    int s;
    int sx;
    int size;
    int totbytes = 0;
    path_id ofile;
    u_int32 count = 1;
    struct sockaddr_in ls_addr;
    struct sockaddr_in to;
    if ((argc <= 1) || (argc > 2) || (argv[1][0] == '-')) {
        fprintf(stderr,"tcprecv <file>\n");
        exit(0);
    }
    if ((errno = _os_create(argv[1], FMODE, &ofile,
                            S_IREAD|S_IWRITE)) != SUCCESS) {
        fprintf(stderr, "can't open file '%s'\n",argv[1]);
        exit(errno);
    }
    if ((sx = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        fprintf(stderr,"can't open socket\n", errno);
        _os_close(ofile);
        exit(errno);
    }
    memset(&ls_addr, 0, sizeof(ls_addr));
    ls_addr.sin_family = AF_INET;
    ls_addr.sin_port =  htons(PORT_NUM);
    ls_addr.sin_addr.s_addr = INADDR_ANY;
    if (bind(sx, (struct sockaddr*)&ls_addr, sizeof ls_addr) == -1) {
        fprintf(stderr,"can't bind socket\n");
```

```
        _os_close(sx);
        _os_close(ofile);
        exit(errno);
    }
    if (listen(sx, 1) < 0) {
        fprintf(stderr,"tcp_listen - failed!\n");
        _os_close(sx);
        _os_close(ofile);
        exit(errno);
    }
    size = sizeof(struct sockaddr_in);
    if ((s = accept(sx, (struct sockaddr*)&to, &size)) < 0) {
        fprintf(stderr, "can't accept\n");
        _os_close(s);
        _os_close(sx);
        _os_close(ofile);
        exit(errno);
    }
    _os_close(sx);
    printf("connected to %s port %d\n", inet_ntoa(to.sin_addr),
           ntohs(to.sin_port));
    while (count) {
        count = sizeof(msgbuf);
        if ((errno = _os_read(s,msgbuf,&count)) != SUCCESS) {
            if (errno == EOS_EOF) {
                break; /*at end of file*/
            }
            fprintf(stderr, "can't recv (cnt=%d)\n", count);
            exit(errno);
        } else if (count == 0) {
            break;
        } else {
            if ((errno = _os_write(ofile, msgbuf, &count)) != SUCCESS) {
                fprintf(stderr, "can't write output\n");
                exit(errno);
            }
            totbytes += count;
        }
    }
    _os_close(s);
    _os_close(ofile);
    printf("read %d bytes\n", totbytes);
    exit(0);
}
```

Example Three: Sending Multicast Messages

The example programs `msend.c` and `mrecv.c` send multicast messages over the network, demonstrating the receipt of packets by multiple destinations. Since multicasting is a connectionless protocol, `mrecv.c` may not see every packet sent.

The `msend` program will send either the message entered at the command line or a default message once, then exit. The signal handler in `mrecv.c` processes any system signal as a signal to kill the process. The preferred method for executing this is to use ^C or ^E from the keyboard.



Note

`msend.c` requires a route in the IP routing table that will return as a match for the multicast group being used. This is required even if the interface is explicitly named using the `-i` option. For example, consider the following routing table:

```
$ netstat -rn
```

Internet:

Destination	Gateway	Flags	Refs	Use	Interface
127.0.0.1	127.0.0.1	UH	0	1	lo0
172.16	172.16.2.226	U	1	122	enet1
172.16.2.226	127.0.0.1	UHS	0	0	lo0
192.168.3	192.168.3.19	U	0	0	enet0
192.168.3.19	127.0.0.1	UHS	0	0	lo0

Using `msend.c` to send to group `225.0.0.172` will result in an error since the routing table contains no route that will match that IP address. Running either of the following route commands will allow it to work:

```
route add -net default 172.16.2.250
```

-OR-

```
route add -net 225.0.0.0 172.16.2.226 240.0.0.0
```

Now `msend` will successfully transmit the packet on the `enet1` interface. If a '`-i 192.168.3.19`' command line option is added, the packet will be sent from the `enet0` interface instead.

Source code for both of these files is in the following directory:

`MWOS/SRC/SPF/INET/EXAMPLES/MULTICAST`

msend — Send multicast packet.

Syntax

```
msend [<opts>]
```

Options

```
[-v] [-l] [-t ttl] [-p port] [-g group] [-i
interface] [-m message]
```

`-v` Enable verbose mode. (Default: off)

`-l` Enable loopback reception of packet. (Default: off)

`-t` Set TTL of output packets. (Default: 1)

`-p` Set port of output packets. (Default: 4433)

`-g` Select destination group. (Default:225.0.0.172)

`-i` Select outgoing interface. (Default: route table lookup of group)

`-m` Select message to send. (Default: "This is a test message")

mrecv

Receive multicast packet.

Syntax

```
mrecv [<opts>]
```

Options

```
[-v] [-p port] [-g group] [-i interface]
```

`-v` Enable verbose mode. (Default: off)

`-p` Set port for selecting incoming packets. (Default: 4433)

`-g` Select destination group. (Default: 225.0.0.172)

`-i` Select receiving interface. (Default: route table lookup of group)

Appendix B: Dynamic Configuration of the `inetdb` Data Module

See the ***Using LAN Communications Pak*** manual for information about configuration file contents.

Sample inetdb Module

The `netdb.l` and `ndblib.l` libraries provide functions that enable direct updating of the `inetdb` module.

To create an `inetdb` module through an application, the function `ndb_create_ndbmod()` is used. (See library function for syntax.)

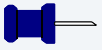
The following example creates an `inetdbX` module with room for additional entries:

```
/* <<<<<<<<<<<<<<<<<<< ndbcreate.c >>>>>>>>>>>>>>>>>>>>>>>>>>>> */
/*
**      Syntax:    ndbcreate <filename>
**      Function:  Create an inetdbX data module
*/
#include <stdio.h>
#include <errno.h>
#include <netdblib.h>

/*
** The following is the creation size array for the configuration
** files.
**
** hosts: 100 bytes (Up to 4 entries)
** hosts.equiv: 0 (not used)
** networks: 80 bytes (Up to 2 entries)
** protocols: 50 bytes (Up to 2 entries)
** services: 50 bytes (Up to 2 entries)
** inetd.conf: 50 bytes (Up to 1 entry)
** resolv.conf: 100 bytes (Up to 1 entry)
** host.config: 0 (not used)
** interfaces.conf: 400 bytes (Up to 2 entries)
** hostname string: 65 bytes (Up to 65 characters)
** routes.conf: 128 bytes (Up to 2 entries)
*/
int size_array[11] = { 100, 0, 80, 50, 50, 50, 100, 0, 400, 65, 128 };

void main(int argc, char* argv[], char* envp[]);

void main(int argc, char* argv[], char* envp[])
{
   char *mod_name;
   int num_files;
   int perm =
        MP_OWNER_READ|MP_OWNER_WRITE|MP_GROUP_READ|MP_WORLD_READ;
```

```
    if (argc == 1) {
        fprintf(stderr, "Module name required (e.g. inetdb2)\n");
        exit(0);
    }
    mod_name = argv[1];
    num_files = 11; /* Create data module with configuration
                       file space */
    if (errno = ndb_create_ndbmod(mod_name, num_files, size_array,
            perm, 0)) {
        fprintf(stderr, "Can't create data module %s\n",mod_name);
        exit(errno);
    }
    exit(0);
}
```

The configuration entries can be updated with the following functions:

- hosts: `puthostent()`, `delhostent()`
- networks: `putnetent()`, `delnetent()`
- protocols: `putprotoent()`, `delprotoent()`
- services: `putservent()`, `delservent()`
- resolv.conf: `putresolvent()`, `delresolvent()`
- interfaces.conf: `putintent()`, `delintent()`
- hostname: `sethostname()`
- routes.conf: `putroutent()`, `delroutent()`



Note

The routes and interfaces entries must be added before the IP stack is initialized. IP reads these entries only at that time. All other entries can be updated after the stack has been brought up.

Manipulating a Host Entry

Following is an example of how to manipulate a host entry:

```
/* <<<<<<<<<<<<<<<<<<<<< hostent.c >>>>>>>>>>>>>>>>>>>>>>>>>>>> */
/*
**      Syntax:    hostent
**      Function:  Manipulate an inetdbX data module host entry
*/

#include <stdio.h>
#include <errno.h>
#include <netdb.h>

void main(int argc, char* argv[], char* envp[]);

void main(int argc, char* argv[], char* envp[])
{
   struct hostent host, *gethost; /* host structure in inetdb */
   static struct in_addr addr; /* IP address of host structure */
   static struct in_addr *h_addrs[2] = {&addr,0}; /* List of IP
                              addresses */
   static char *h_aliases[3] = {"alpha", "beta", 0};

   /*fill in the host entry*/
   addr.s_addr = inet_addr("10.0.0.1");
   host.h_name = "gamma";
   host.h_aliases = h_aliases;
   host.h_addr_list = (char **)h_addrs;
   host.h_length = sizeof(struct in_addr);
   host.h_addrtype = AF_INET;

   /*Insert Host Entry in InetdbX module*/
   if ((errno = puthostent(&host)) == -1) {
      fprintf(stderr,"Error in puthostent!\n");
      exit(errno);
   }

   /*Read Host Entry in InetdbX module*/
   if ((gethost = gethostbyname(host.h_name)) == NULL) {
      fprintf(stderr,"Error in gethostbyname!\n");
      exit(errno);
   }
   printf("Obtained host entry %s\n",gethost->h_name);

   /*Delete Host Entry in InetdbX module*/
   if ((errno = delhostbyname(host.h_name)) == -1) {
      fprintf(stderr,"Error in delhostbyname!\n");
      exit(errno);
   }
   exit(0);
}
```

Changing the DNS Client Entry

Following is an example of how to change the DNS client entry:

```
/* <<<<<<<<<<<<<<<<<<< resolvent.c >>>>>>>>>>>>>>>>>>>>>>>>>>>>>> */
/*
**      Syntax:    resolvent
**      Function:  Manipulate an inetdbX data module DNS client entry
*/
#include <stdio.h>
#include <errno.h>
#include <netdb.h>
#include <resolv.h>
void main(int argc, char* argv[], char* envp[]);

void main(int argc, char* argv[], char* envp[])
{
    struct resolvent res, *getres; /* DNS client resolve structure in
                          inetdb */

    memset(&res, 0, sizeof(res));
    res.domain = "alphatest.com";
    res.nameservers[0] = "10.0.0.1";
    res.nameservers[1] = "10.0.0.2";
    res.search[0] = "alphatest.com";
    res.search[0] = "betatest.com";
    res.search[0] = "gammatest.com";

    /*Insert Resolve Entry in InetdbX module*/
    if ((errno = putresolvent(&res)) == -1) {
        fprintf(stderr,"Error in putresolvent!\n");
        exit(errno);
    }

    /*Read Resolve Entry in InetdbX module*/
    if ((getres = getresolvent()) == NULL) {
        fprintf(stderr,"Error in getresolvent!\n");
        exit(errno);
    }
    printf("Obtained resolve entry for domain %s\n",getres->domain);

    /*Delete Resolve Entry in InetdbX module*/
    if ((errno = delresolvent()) == -1) {
        fprintf(stderr,"Error in delresolvent!\n");
        exit(errno);
    }
    exit(0);
}
```

Adding an Interface Entry

Following is an example of how to add an interface entry:

WARNING

These functions must be called before `ipstart` is run!

```
/* <<<<<<<<<<<<<<<<<<<<<< interface.c >>>>>>>>>>>>>>>>>>>>>>> */
/*
**  Syntax:   interface
**  Function: Manipulate an inetdbX data module hostname and interface
*/
#include <stdio.h>
#include <errno.h>
#include <netdb.h>
#include <net/if.h>

void main(int argc, char* argv[], char* envp[]);

void main(int argc, char* argv[], char* envp[])
{
    struct n_ifnet ifnet, *getint; /*Structure to add Interface*/
    struct n_ifaliasreq ifalias[1];
    struct sockaddr_in *sock_int;

    memset(&ifnet, 0, sizeof(ifnet));
    memset(&ifalias, 0, sizeof(ifalias));

    /*Build interface entry */
    strcpy(ifnet.if_name, "enet0");      /* Interface Name */
    strcpy(ifnet.if_stack_name,
                    "/sple0/enet");    /* Device Path to open */
    ifnet.if_flags = IFF_UP|IFF_BROADCAST       /* Device Flags */
    /*ifnet.mw_flags = IFF_NOMULTICAST;*//* Override driver default */
    strcpy(ifalias[0].ifra_name, "enet0");

    /* Fill in interfaces address information */
    /* IP Address */
    sock_int = (struct sockaddr_in *) &ifalias[0].ifra_addr;
    sock_int->sin_family = AF_INET;
    sock_int->sin_addr.s_addr = inet_addr("10.0.0.1");
    /* Broadcast Address */
    sock_int = (struct sockaddr_in *) &ifalias[0].ifra_broadaddr;
```

```
    sock_int->sin_family = AF_INET;
    sock_int->sin_addr.s_addr = inet_addr("10.255.255.255");
    /* Subnet Mask */
    sock_int = (struct sockaddr_in *) &ifalias[0].ifra_mask;
    sock_int->sin_family = AF_INET;
    sock_int->sin_addr.s_addr = inet_addr("255.0.0.0");

    /* Insert Interface Entry in InetdbX module */
    if ((errno = putintent(&ifnet,ifalias,1)) == -1) {
        fprintf(stderr,"Error in putintent!\n");
        exit(errno);
    }

    /*Read Interface Entry in InetdbX module*/
    if ((getint = (struct n_ifnet *)getintent()) == NULL) {
        fprintf(stderr,"Error in getintent!\n");
        exit(errno);
    }
    printf("Obtained Interface entry for device name %s\n”,
                            getint->if_name);

    /* Delete Interface Entry in InetdbX module */
    if ((errno = delintbyname(getint->if_name)) == -1) {
        fprintf(stderr,"Error in delintbyname!\n");
        exit(errno);
    }
    exit(0);
}
```

Adding, Obtaining, and Deleting a Route Entry

Following is an example of how to add, get, and delete a route entry from the `inetdbx` data module.

WARNING

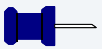
These functions must be called before `ipstart` is run!

```
/* <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< uproute.c >>>>>>>>>>>>>>>>>>>>>>>>>>>>>> */
/*
**      Syntax:    uproute
**      Function:  Manipulate an inetdbX data module route entry
*/


#include <stdio.h>
#include <errno.h>
#include <netdb.h>
#include <net/route.h>

void main(int argc, char* argv[], char* envp[]);

void main(int argc, char* argv[], char* envp[])
{
    struct rtreq route,*getroute;   /*Structure for Route Entry*/

    memset(&route,0,sizeof(route));

    /* The following example adds a host route entry */
    route.req = RTM_ADD;            /*Add Route request*/
    route.flags &= ~RTF_HOST;
    route.flags |= RTF_UP|RTF_GATEWAY;

    /*Fill in destination host route address*/
    ((struct sockaddr_in *) &route.dst)->sin_addr.s_addr =
                          inet_addr("11.0.0.1");
    ((struct sockaddr_in *) &route.dst)->sin_family = AF_INET;
    * (u_char *) &route.dst = sizeof(struct sockaddr_in);

    /*Fill in gateway route address*/
    ((struct sockaddr_in *) &route.gateway)->sin_addr.s_addr =
                          inet_addr("10.0.0.1");
    ((struct sockaddr_in *) &route.gateway)->sin_family = AF_INET;
    * (u_char *) &route.gateway = sizeof(struct sockaddr_in);

    /*Insert Route Entry in InetdbX module*/
    if ((errno = putroutent(&route)) == -1) {
        fprintf(stderr,"Error in putroutent!\n");
        exit(errno);
    }

    /*Read Route Entry in InetdbX module*/
    if ((getroute = getroutent()) == NULL) {
        fprintf(stderr,"Error in getroutent!\n");
        exit(errno);
    }
    printf("Obtained route entry for host %s\n",
               inet_ntoa(((struct sockaddr_in*) &getroute->dst)->sin_addr));

    /*Delete Route Entry in InetdbX module*/
    if ((errno = delroutent(&route)) == -1) {
        fprintf(stderr,"Error in delroutent!\n");
```

```
        exit(errno);
    }
    exit(0);
```

Initializing the IP Stack

Starting the IP stack via an application program can be done with the function `ip_start()` provided in `socket.l`.



Note

The System Memory Buffer handler (`sysmbuf`) must be installed before starting the IP stack.

```
/* <<<<<<<<<<<<<<<<<<<<<<<< ip_start.c >>>>>>>>>>>>>>>>>>>>>>>> */
/*
**      Syntax:    ip_start
**      Function:  Initialize IP stack
*/

#include <types.h>
#include <const.h>
#include <stdio.h>
#include <errno.h>

void main(int argc, char* argv[], char* envp[]);

void main(int argc, char* argv[], char* envp[])
{
    error_code err;
    printf("Initializing IP Stack\n");
    if ((err = ip_start()) !=success) {
        fprintf(stderr,"Error returned from ip_start\n");
        exit(err);
    }
    printf("Initialization complete.\n");
    exit(0);
}
```

Index

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

Product Discrepancy Report

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_____

RadiSys.

MICROWARE SOFTWARE